NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTAMENTO DE INFORMÁTICA

JOSÉ MIGUEL ROSA LEAL

Licenciado em Ciência e Engenharia Informática

E-INVOICING: FATURAS EM TEMPO REAL

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Novembro, 2021

DEPARTAMENTO
DE INFORMÁTICA

E-INVOICING: FATURAS EM TEMPO REAL

JOSÉ MIGUEL ROSA LEAL

Licenciado em Ciência e Engenharia Informática

Orientador: Ricardo Bruno Rodrigues Caetano
Diretor e CTO, Opensoft - Soluções Informáticas, SA

Coorientadora: Maria Cecília Farias Lorga Gomes
Professora Auxiliar, NOVA School of Science and Technology | FCT NOVA

Júri:

Presidente: Hervé Miguel Cordeiro Paulino
Professor Auxiliar, NOVA School of Science and Technology | FCT NOVA

Arguente: Miguel Carlos Pacheco Afonso Goulão
Professor Associado, NOVA School of Science and Technology | FCT NOVA

Orientadores: Ricardo Bruno Rodrigues Caetano
Diretor e CTO, Opensoft - Soluções Informáticas, SA

Maria Cecília Farias Lorga Gomes
Professora Auxiliar, NOVA School of Science and Technology | FCT NOVA

MESTRADO EM ENGENHARIA INFORMÁTICA

Universidade NOVA de Lisboa
Novembro, 2021

***e-Invoicing*: Faturas em tempo real**

Agradecimentos

Em primeiro lugar, gostaria de expressar a minha gratidão ao orientador Eng. Ricardo Caetano por todo o apoio, recomendações e disponibilidade durante a conceção da dissertação.

Em segundo, gostaria de agradecer à coorientadora Dra. Maria Cecília Gomes pela prontidão no seu *feedback* e orientação na construção do documento.

Em terceiro, gostaria de reconhecer o apoio da Opensoft, principalmente os elementos da equipa SAFTPRO, que acompanharam este projeto no seu dia-a-dia.

Além disso, gostaria de agradecer à FCT NOVA pelos 5 anos de conhecimento, amizades e experiências. Queria também agradecer aos colegas que me acompanharam nesta jornada e contribuíram para o meu crescimento pessoal e profissional.

Por último, gostaria de agradecer à minha família, especialmente os meus pais e a minha irmã, que fizeram um esforço financeiro tremendo para eu ter condições para estudar a 300km de casa e deram-me ânimo para dar o meu melhor nesta oportunidade.

Resumo

Atualmente, vários países perdem milhões em receitas devido à fraude e evasão fiscal. Esta perda é contabilizada pela diferença entre as receitas de impostos esperadas e as receitas efetivamente recolhidas. Para além disso, estima-se que a maioria das faturas emitidas a nível mundial ainda são processadas manualmente e que existem geografias onde a faturação ainda não é uma realidade.

A transformação digital e automatização dos processos de faturação tornaram-se uma solução eficaz para estes problemas e surgiram alguns *standards* de dados que facilitam a comunicação de faturas às entidades responsáveis pelo controlo fiscal. Muitos países já os adotam, mas cada um deles introduz pequenas alterações consoante a sua jurisdição.

Existe assim oportunidade para uma organização que possua um sistema que agilize a adoção da fatura eletrónica, ou que se posicione como implementador deste tipo de sistemas, respondendo aos requisitos particulares de cada país. Deste contexto, surgiu o produto SAFTPRO, que possibilita o intercâmbio de dados e viabiliza a operacionalização de um processo de controlo eletrónico das transações comerciais efetuadas num qualquer país. Porém, não suporta a comunicação direta das faturas emitidas, sendo estas registadas posteriormente no sistema.

Pretende-se assim desenvolver um módulo adicional no produto SAFTPRO que permita a submissão, validação e certificação de elementos de faturas, com a possibilidade para se adequar às especificidades introduzidas por cada país, sem ter de reconstruir o produto.

A implementação deste tipo de sistema visa a substituição da fatura em papel, o que traz grandes vantagens na agilização e transparência das relações entre Empresas, Clientes, Fornecedores e Autoridades Tributárias.

Palavras-chave: Fatura Eletrónica; Transformação Digital; SAFTPRO . . .

Abstract

Currently, some countries are losing millions in revenue due to tax evasion and fraud. This loss is established by the difference between the expected tax revenue and the revenue actually collected. In addition, it is estimated that most invoices issued worldwide are still processed manually and in some geographies invoicing is not yet a reality.

The digital transformation and automation of billing processes became an effective solution to these problems and some data standards have emerged to facilitate the communication of invoices to the entities responsible for fiscal control. Many countries already adopt them, but each of them introduces small changes depending on their jurisdiction.

Therefore, exists an opportunity for an organization that has a system that streamlines the adoption of electronic invoices, or that positions itself as an implementer of this type of system, responding to the particular requirements of each country. From this context, the SAFTPRO product arises, which enables the exchange of data and the operationalization of an electronic control process for commercial transactions carried out in any country. However, it does not support the direct communication of invoices issued. These need to be subsequently registered in the system.

Consequently, what is intended to develop is an additional module in the SAFTPRO product that allows the submission, validation and certification of invoice elements, with the possibility to adapt to the specificities introduced by each country and with scalability to adapt to different volume contexts, without having to rebuild the whole product from scratch.

The implementation of this type of system aims to replace paper invoicing, which brings great advantages in the speed and transparency of the relations between Companies, Customers, Suppliers and Tax Authorities.

Keywords: Electronic Invoice, Digital Transformation, SAFTPRO ...

Índice

Índice de Figuras

Índice de Tabelas

Índice de Listagens

Glossário

Bootstrap	Impulsionar o arranque de um projeto, reduzindo os custos inicias de desenvolvimento, através da geração de esqueletos aplicacionais que concentram as funcionalidades mais comuns, para um determinado tipo de aplicação. 5, 27
Business-to-Business	Refere as transações de bens e/ou serviços entre empresas. 8
Business-to-Government	Refere as negociações de produtos e/ou serviços entre empresas e o setor público administrativo. 10
e-Government	Refere-se à utilização das tecnologias da informação e comunicação como plataforma para melhorar a acessibilidade de algumas atividades do setor público, prestação de serviços e troca de informações. 2
e-Procurement	Automatização de todas as etapas necessárias para as compras *Business-to-Business*. 38
Electronic Data Interchange	Permite a troca de documentos normalizados entre sistemas informáticos. É muito utilizado nas relações comerciais, adaptando-se às necessidades dos vários setores. 6
Point of Sale	Local onde ocorre uma transação de venda. 2
Agente Económico	Toda a entidade capaz de influenciar a economia de um país. Nesta definição enquadram-se os consumidores, os comerciantes e os seus fornecedores, assim como algumas instituições públicas e administrativas do estado. No entanto, neste documento a sua utilização abrange apenas os consumidores e os seus respetivos provedores de serviços (comerciantes e fornecedores). 2, 31

Transformação Digital Processo de reavaliação dos modelos de negócio para integrar as novas tecnologias, com a finalidade de alterar a forma como se disponibiliza um determinado serviço e de como se entrega valor aos clientes. 1

Siglas

AP *Access Point* 28

BIS *Business Interoperability Specifications* 28, 42, 77

CFDI *Comprobante Fiscal Digital por Internet* 11
CIUS *Core Invoice Usage Specification* 19, 42, 83
CRUD Create, Read, Update and Delete 61

DOM *Document Object Model* 12
DSL Domain Specific Language 23, 42, 88
DTD *Document Type Definition* 15

ebMS ebXML *Messaging Service* 34
EDI *Electronic Data Interchange* 17

JAXB *Java Architecture for* XML *Binding* 13
JPQL Java *Persistence Query Language* 56

MSH *Message Service Handler* 34

PKI Public Key Infrastructure 162
P-Mode Processing Mode 35

RELAX NG *REgular LAnguage for* XML *Next Generation* 15

SAF-T *Standard Audit File for Tax purposes* 1, 8, 75
SAX *Simple* API *for* XML 13
SML *Service Metadata Locator* 30, 41

SMP *Service Metadata Publisher* 30, 41, 60
StAX *Streaming* API *for* XML 13

TIA *Transport Infrastructure Agreement* 28
TLS *Transport Layer Security* 161

UBL *Universal Business Language* 1, 17, 42, 75, 101
UTD *Universal Transfer Document* 10

XPath *XML Path Language* 15, 88
XSD XML *Schema Definition* 14, 42, 83
XSL *eXtensible Stylesheet Language* 16
XSLT *eXtensible Stylesheet Language Transformations* 17, 51, 84

1

Introdução

1.1 Contexto

Nos dias de hoje, muitos países perdem milhares de milhões em receitas devido à fraude e evasão fiscal [47], contabilizada pela diferença entre as receitas sobre impostos esperadas e as efetivamente recolhidas. Por exemplo, nos Estados Unidos da América este valor pode atingir anualmente 1 trilião de dólares, o que equivale aproximadamente a 846 biliões de euros [8].

Para além disso, estima-se que cerca de 90% de todas as faturas emitidas mundialmente ainda requerem processamento manual e que existem geografias onde a faturação ainda não é uma realidade [29]. Estes problemas e o elevado custo associado à gestão de faturas em papel tornaram-se os principais catalisadores para a Transformação Digital e automatização dos processos de faturação [62].

A fatura eletrónica é uma tecnologia em expansão relativamente rápida em países da América Latina, assim como em muitos países Europeus e Asiáticos. No entanto, a maioria das organizações ainda não acordou num serviço ou sistema único, o que fragmenta o mercado. O lado oposto da moeda também pode ser encontrado em algumas regiões Africanas e outras partes do mundo. Nestes sítios, o desafio fundamental ainda passa por criar faturas compatíveis com os impostos após cada transação, devido à limitação das suas infraestruturas disponíveis, o que nos leva de volta ao problema da fraude e evasão fiscal [29].

Para responder a estas necessidades foram definidos *standards* de dados, como o *Standard Audit File for Tax purposes* (SAF-T)[1] e a *Universal Business Language* (UBL)[2], que facilitam a comunicação de faturas às entidades nacionais responsáveis pelo controlo de taxas e a entidades externas para realização de eventuais auditorias. Muitos países já os utilizam, embora cada um deles introduza pequenas alterações para estar em conformidade com a sua jurisdição.

Existe assim oportunidade para uma organização que possua um sistema que agilize

[1] https://en.wikipedia.org/wiki/SAF-T
[2] http://ubl.xml.org/

a adoção da fatura eletrónica ou que se posicione como implementador desde tipo de sistema, respondendo aos requisitos particulares de cada país.

1.2 Problema

Os sistemas de faturação em papel são compostos por processos monótonos, pouco económicos e sujeitos a erros ou inconsistências humanas. Ao mesmo tempo, são pouco eficazes no combate à fraude e evasão fiscal, originando perdas constantes e significativas para os governos[31]. Nos países que carecem de sistemas de faturação, o problema é a falta de controlo sobre as transações efetuadas e a ausência de receitas provenientes de impostos[29].

Estas situações afetam as Autoridades Tributárias de cada país, mas também possuem outros efeitos negativos. Um deles reflete-se na injustiça para os que pagam mais impostos, por existirem Contribuintes que não cumprem as suas obrigações fiscais e o outro está relacionado com a falta de transparência nas relações entre os Agentes Económicos.

Deste contexto surge o SAFTPRO[3], composto por um portal *Web* e uma REST API, que disponibiliza um conjunto de funcionalidades para a gestão e consulta de ficheiros SAF-T, o que viabiliza a operacionalização de um processo de controlo eletrónico das transações comerciais efetuadas. Os ficheiros submetidos para processamento e validação não têm tamanho fixo e podem ir desde alguns megabytes até centenas de gigabytes.

O SAFTPRO é desenvolvido e mantido pela empresa Opensoft[4], constituída por engenheiros de *software* altamente qualificados, com uma vasta experiência no desenvolvimento de *software* de qualidade, especializada em sistemas críticos com ênfase nas áreas de *e-Government*.

De momento, o produto não suporta a submissão direta das faturas às Autoridades Tributárias. Ou seja, as faturas têm de ser registadas no sistema após a sua emissão. É aqui que entra o tema da dissertação e o seu propósito é responder às necessidades do SAFTPRO, de forma a poder dotá-lo com capacidade a para submeter, validar e certificar elementos de faturas baseadas no modelo normalizado internacional UBL.

Em simultâneo, surgem outros tipos de desafios, tais como:

- Possibilitar a configuração do formato das faturas, para atender às especificidades de cada país, sem ter de reconstruir o produto de raíz;
- Disponibilizar APIs que facilitem a adoção do sistema pelos produtores de software de faturação ou *Points of Sale* (POS);
- Estruturar a solução com elasticidade suficiente, para que esta possa escalar e adaptar-se a diferentes contextos de volume de dados;
- Capacidade de resiliência do sistema.

[3] https://www.saftpro.com/
[4] https://www.opensoft.pt/

Este tipo de sistema tem como finalidade substituir a fatura em papel e prevenir a fraude e evasão fiscal, o que traz grandes benefícios na agilização e transparência das relações entre Empresas, Clientes, Fornecedores e as Autoridades Tributárias. Além disso, permite reduzir os custos das entidades envolvidas e reduz o mercado paralelo, com as subsequentes vantagens no aumento da receita proveniente de impostos.

1.3 Objetivos e Contribuições

Pretende-se investigar uma solução que permita o registo de dados de faturas, integrado num modelo de comunicação misto por *Webservice*, *Webform* ou *File Upload* perante um sistema central com capacidade de validação de faturas e posterior análise de dados de faturação. A solução deverá tornar mais cómoda e eficiente a comunicação detalhada de elementos de faturas, reduzindo a necessidade da disponibilização das faturas em papel ao adquirente pelo comerciante.

O processo de investigação irá culminar no desenvolvimento de uma prova de conceito que demonstre a submissão, validação e certificação de elementos de faturas.

As necessidades que constituem os objetivos a alcançar durante a tese são[5]:

- **ND01:** adaptar o formato das faturas para que estas possam ser processadas, validadas e persistidas pelo SAFTPRO;
- **ND02:** permitir ao consumidor intervir na aprovação da fatura;
- **ND03:** viabilizar, através de APIs e canais de submissão, a adoção do sistema pelos *softwares* de faturação e/ou POS;
- **ND04:** disponibilizar mecanismos que suportem a certificação das faturas, para garantir a sua autenticidade, integridade e não repúdio.

Estas necessidades, foram posteriormente consolidadas e reconhecidas num conjunto de requisitos funcionais e não funcionais, que podem ser encontrados com mais detalhe na secção 3.3.

Relativamente aos requisitos não funcionais, o ênfase assenta sobre a categoria de desempenho, pela sua enorme responsabilidade no desenvolvimento da solução, visto que todo o ciclo de vida da fatura depende da correta implementação do sistema e é necessário garantir que todos os processos despoletados são o mais eficientes possíveis. Seguem-se os requisitos de documentação e manutenção, pois são essenciais para garantir que o que é desenvolvido é possível de manter e estender, consoante as necessidades que possam surgir no futuro. Neste sentido, será documentado todo o processo de configuração da solução, assim como todas as APIs e componentes desenvolvidos.

Por fim, destacam-se os requisitos de segurança que são tão importantes quanto a criticidade da informação a transmitir. O objetivo é garantir que todas as comunicações

[5] **NDXX** - corresponde ao identificador interno atribuído à necessidade (*need*) identificada.

são estabelecidas através de protocolos seguros, para assegurar a integridade da informação transmitida e disponibilizar as funcionalidades implementadas apenas a utilizadores autenticados e autorizados.

1.4 Visão Global da Solução

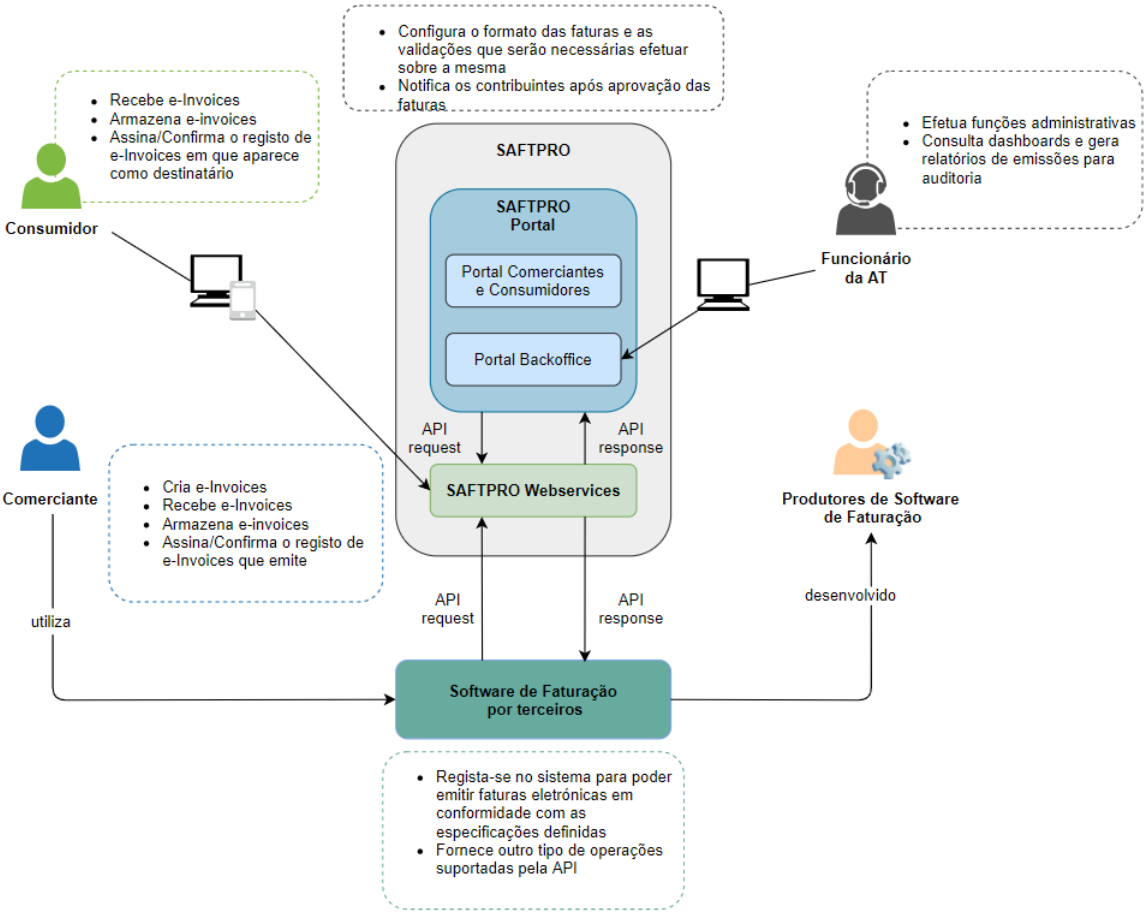


Figura 1.1: SAFTPRO - Visão Global da Solução

Tendo em conta as necessidades identificadas, a solução proposta baseia-se no alargamento das funcionalidades do SAFTPRO *Webservices*.

No capítulo 3, será descrito com mais detalhe a arquitetura idealizada. No entanto, podemos encontrar na figura 1.1 uma representação geral das novas funcionalidades por utilizador, os seus respetivos ambientes de utilização e camadas de acesso.

1.5 Organização do Documento

Capítulo 1: esclarece os objetivos a alcançar, o seu contexto e motivação.

Capítulo 2: descreve o estado da arte e encontra-se estruturado de acordo com o plano de trabalhos inicial. Inicialmente, surgiu a necessidade de identificar com clareza o que é a fatura eletrónica. Posteriormente, foi dado ênfase no processamento e validação de ficheiros XML, pelo facto de ser o formato de representação mais utilizado para este tipo de documentos. De seguida, foi explorado a geração dinâmica de código pela sua importância no *Bootstrap* de novos projetos e/ou módulos. Por último, investigou-se as potencialidades das especificações Peppol.

Capítulo 3: apresenta a arquitetura atual do produto SAFTPRO, seguida pelo conceito aplicacional e descrição da arquitetura proposta para a solução. Para além disso, inclui o modelo de dados e todas as tecnologias consideradas relevantes para o desenvolvimento das funcionalidades.

Capítulo 4: consiste na descrição detalhada da implementação da solução e em certa instâncias apresenta o trabalho que ainda é necessário efetuar, assim como as alternativas consideradas.

Capítulo 5: apresenta mecanismos de validação utilizados e os resultados obtidos na fase de implementação.

2

Contexto e Trabalho Relacionado

As tendências tecnológicas são bastante dinâmicas e estão a evoluir a um ritmo elevado. As soluções que se consideravam revolucionárias há alguns anos atrás, encontram-se agora em risco de ficar obsoletas. O mesmo pode ser dito sobre o mundo tributário e os sistemas que o suportam. A corrida para acabar com a faturação manual e em papel tem vindo a acelerar. Contudo, a grande maioria das empresas ainda recebe faturas num formato que requer a entrada manual de dados nos sistemas de contabilidade [62].

Nos dias de hoje, estima-se que a economia digital global ronde os 6 biliões de utilizadores de *smartphones*, o que impulsiona novos mercados e oportunidades de negócio, mas também fornece benefícios para os governos relativamente à cobrança de impostos, deteção de fraudes e cumprimento de jurisdições.

A fatura eletrónica está em desenvolvimento progressivo para dar uma resposta mais rápida e eficiente a este tipo de situações, pois fornece meios de fiscalização e ajuda a mitigar a perda de receitas.

A partir de 2017, muitos países começaram a adicionar ao seu arsenal a fatura eletrónica, impulsionados também pela globalização e de modo a garantir a conformidade com os impostos indiretos e alfandegários [30].

2.1 Fatura Eletrónica

A fatura eletrónica é um documento semelhante à fatura convencional em formato eletrónico, ou seja, desmaterializado e representa documentos transacionais entre parceiros comerciais, por exemplo um Cliente e um Fornecedor. Geralmente, é composta por um conjunto estruturado de dados que são enviados como uma mensagem *Electronic Data Interchange* (EDI) para posterior validação/autorização pelas Autoridades Tributárias.

O processo de transmissão deste tipo de faturas envolve um canal seguro e uma assinatura digital pelos Contribuintes, para garantir autenticidade, integridade e não repúdio da informação [9].

A faturação eletrónica, ou *e-Invoicing*, permite acompanhar de forma mais precisa o fluxo monetário entre Contribuintes e o pagamento de impostos às Autoridades Tributárias (fig. 2.1, descreve este processo), o que a torna uma excelente arma para combater a fraude fiscal.

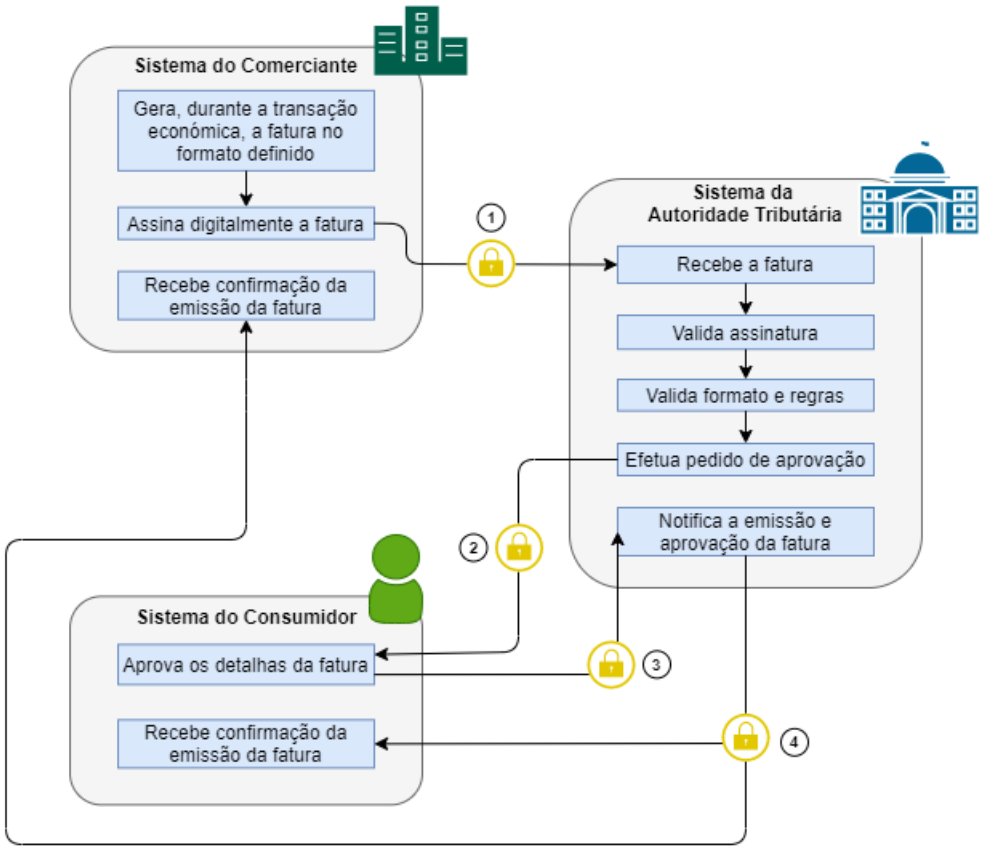


Figura 2.1: Descrição alto nível de um processo de *e-Invoicing*

É comum as Autoridades Tributárias certificarem o formato a ser utilizado (normalmente baseado em XML), assim como o seu conteúdo e representação gráfica. Cabe ainda a estas autoridades, regulamentar o processo de armazenamento e as funcionalidades disponibilizadas sobre as faturas emitidas [14].

Estima-se que é possível economizar cerca de 60% a 80% dos custos, em comparação com o processo manual baseado em papel [29]. Além disso, os tempos de transmissão são mais curtos e a fatura ao ser emitida de forma estruturada reduz o risco de erro humano ou inconsistências, o que aumenta a qualidade, assim como a transparência da informação e permite estabelecer uma relação comercial mais próspera entre as entidades envolvidas.

Por último, a uniformização do formato das faturas facilita o processo de integração com sistemas de gestão e incentiva o seu uso pelos Contribuintes.

2.1.1 Modelos Tributários

No meio tributário existem alguns modelos, que caracterizam o momento em que a informação de uma transação chega às mãos das Autoridades Tributárias. Essencialmente, a sua diferença é o intervalo de tempo disponibilizado para detetar possíveis fraudes ou irregularidades [62, 29].

2.1.1.1 *Post-Audit*

Neste modelo a fatura eletrónica não desempenha um papel vital, uma vez que a sua utilização é voluntária e o seu formato não precisa ser estruturado.

Existem duas tendências dentro deste modelo, a primeira é considerada mais "antiquada" e a sua principal fonte de informação para as Autoridades Tributárias é as declarações de impostos, nomeadamente o IVA.

A segunda, ocorre em países que já adotam a fatura eletrónica, mas com caráter opcional em transações *Business-to-Business* (B2B). Numa versão mais avançada desta segunda tendência, algumas informações são transmitidas em ficheiros SAF-T para as Autoridades Tributárias, mas o prazo de transmissão pode variar entre alguns dias e alguns meses.

Como características deste modelo temos a baixa visibilidade nas transações, auditorias tardias e a predominância da faturação manual, visto que a fatura eletrónica tem caráter voluntário.

2.1.1.2 *Real-Time*

Corresponde a um modelo intermédio entre *Post-Audit* e *Tax-Clearance*, mas no seu cerne pode ser visto como um modelo *Post-Audit*.

Neste modelo as auditorias ocorrem de uma maneira mais reativa. No entanto, ainda é permitido que a transmissão da informação, referente às transações, seja enviada em deferido para as Autoridades Tributárias.

2.1.1.3 *Tax-Clearance*

A comunicação entre Contribuintes e Autoridades Tributárias é estruturada e imediata, o que garante uma visibilidade total e instantânea das transações, viabilizando o processo de validação/aprovação no momento em que a transação está a decorrer. Deste modo, é possível detetar em tempo real qualquer incumprimento e responsabilizar os culpados no ato.

Neste modelo, a fatura eletrónica é fulcral e o seu caráter é obrigatório.

Conclusões

A implementação do *e-Invoicing* permitirá ao SAFTPRO transitar do modelo *Post-Audit*, para um modelo em conformidade com as características do *Tax-Clearance*.

Para as empresas, os benefícios recaem na simplificação dos fluxos de negócios em termos contabilísticos e fiscais. Porém, algumas destas ainda processam a entrada dos dados das faturas de uma forma manual e enfrentam agora uma janela mais apertada para disponibilizar faturas em conformidade com as jurisdições impostas. Se os dados armazenados por uma empresa estiverem errados ou não estruturados, o problema será amplificado quando a informação for inserida numa fatura eletrónica.

Com o modelo *Tax-Clearance* os detalhes internos da cadeia de abastecimento e contabilidade tornam-se mais visíveis para os governos, proporcionando a capacidade de validar de forma mais confiável (ou rejeitar rapidamente) as emissões de faturas e as declarações de impostos, o que eleva o risco de multas por não conformidade [30].

A generalização da fatura eletrónica possui potencial para acabar com as rotinas monótonas da faturação em papel e está a promover o rumo às "*Cashless Societies*" (sociedades sem dinheiro físico) [30].

2.1.2 *e-Invoicing* no Mundo

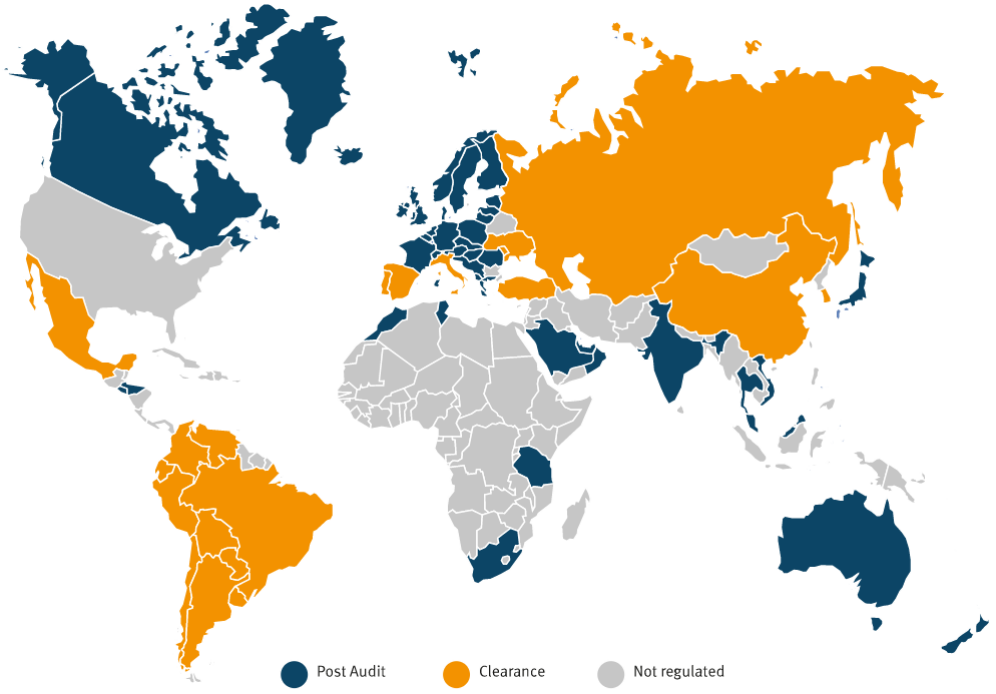


Figura 2.2: Adoção dos diferentes tipos de modelo pelo mundo [22]

2.1.2.1 Situação Europeia

Na União Europeia, a reforma do IVA permitiu acelerar o desenvolvimento da fatura eletrónica. Com ela foram introduzidos o conceito de fatura eletrónica e todas as regras

fundamentais pelas quais esta se rege [61]. Todos os estados membros têm de implementar as regulações definidas na diretiva 2014/55/EU[1], que se limita a definir a semântica da fatura eletrónica e tem como objetivo a sua uniformização, para permitir interoperabilidade em contratos públicos [7, 16, 61].

A maioria dos países já cumpriu, desenvolvendo a sua própria legislação para refletir a diretiva no seu quadro jurídico.

2.1.2.2 Resto do Globo

A adoção do *e-Invoicing* está a crescer na América do Norte, a um ritmo sustentável e os esforços estão concentrados em definir os requisitos da fatura eletrónica para alinhá-los o mais próximo possível com as estruturas estabelecidas ou a estabelecer, de modo a promover interoperabilidade[29, 51].

Na Rússia as atividades de faturação eletrónica começaram relativamente tarde. No entanto, encontram-se agora a desenvolver a um ritmo altamente dinâmico [29]. A sua utilização tornou-se comum nos setores de distribuição em massa, farmacêutico e automóvel. O formato utilizado é o *Universal Transfer Document* (UTD)[2] e é regulamentado para incluir campos definidos pela Autoridade Tributária Russa [5]. A emissão das faturas é feita através de operadores autorizados e as empresas devem obter um certificado expedido por uma autoridade de certificação Russa para poderem assinar digitalmente as faturas emitidas.

Na Ásia e Pacífico a implementação está a ser rápida. As taxas de adoção são superiores a 60% na China e na Índia para transações B2B. Noutros países do Pacífico, a fatura eletrónica é adotada de maneira diferente. Por exemplo, Singapura foi dos primeiros países asiáticos a adotá-la e desde 2008 que existe a sua obrigatoriedade para transações *Business-to-Government* (B2G), mas a regulamentação permanece quase inexistente para os restantes mercados [51]. O caminho a percorrer consiste na uniformização do modelo para os sistemas de faturação. Por exemplo, a Austrália e Nova Zelândia assinaram um contrato para criar e manter um formato comum, de modo a melhorar a sua produtividade e reduzir os custos ao operar num meio único [29].

Por último, temos a América Latina que é provavelmente a região mais avançada em termos de implementação [13], apesar de não possuir um formato uniforme que facilite transações internacionais [51]. A fatura eletrónica faz parte dos procedimentos normais das relações comerciais de cada vez mais países como o Brasil, Chile, Guatemala e México. Em termos de legislação os países mais dominantes são o México e o Brasil, que pretendem agora avançar para outras políticas de cumprimento tributário [13].

[1] https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32014L0055

[2] https://help.sap.com/viewer/293e92f54872474fbed03a40a3262ccb/6.17.17/en-US/822a32543ab11d5de10000000a44538d.html

Conclusões

O objetivo desta secção foi compreender o estado de *e-Invoicing* pelo mundo (fig. 2.2), perceber que problemas são predominantes e identificar com mais clareza as soluções que já existem, como são implementadas e que formatos utilizam.

Foram analisados quatro países com mais detalhe (ver anexo I), o que permitiu extrapolar informação valiosa para a arquitetura da solução e que componentes são necessários para responder aos desafios lançados.

No que diz respeito aos diferentes formatos utilizados, existe uma grande semelhança nos seus campos obrigatórios. Noutros, existem campos que à partida seriam obrigatórios numa fatura, mas como o formato dá suporte à criação de outros tipos de documentos isto não acontece. Por exemplo, o método de pagamento no *Comprobante Fiscal Digital por Internet* (CFDI)[3].

Três dos quatro formatos analisados têm origem nacional e estão muito direcionados para a emissão de faturas dentro do próprio país, o que condiciona a interoperabilidade. A exceção é o Peppol BIS *Billing* 3.0[4], que pode ser utilizado sem qualquer modificação, mas pode ser estendido para acomodar as necessidades de cada país.

Os campos que surgem com mais frequência em todas as representações são:

- Número da Fatura;
- Data/Hora de emissão;
- Comerciante/Fornecedor;
- Consumidor;
- Descontos/Taxas aplicáveis;
- Imposto;
- Método de Pagamento;
- Descrição dos itens faturados;
- Preço total a faturar;
- Preço total líquido;
- Moeda.

No que diz respeito ao modelo de integração, existem essencialmente duas abordagens. Numa delas, a infraestrutura disponibilizada pelas Autoridades Tributárias é um portal *Web* direcionado para a receção de faturas e posterior consulta pelos Contribuintes, assim como alguns processos administrativos. A responsabilidade de emissão, submissão e validação das faturas eletrónicas recai sobre terceiros que são certificados para tal. Estes após completarem o fluxo de processamento da fatura encaminham-na para o sistema da Autoridade Tributária para que possa ser disponibilizada aos Contribuintes [13, 15, 18]. A outra, é mais centralizada, ou seja, a plataforma disponibilizada para consulta e processos administrativos fornece também meios para submissão, validação e autorização

[3] https://edicom.mx/compliance-mexico
[4] https://docs.peppol.eu/poacc/billing/3.0/

das faturas. Os meios de submissão predominantes são o *upload* direto ou através de *webservices* [9, 17].

Para terminar, o modelo de integração deverá ser semelhante ao segundo, uma vez que o SAFTPRO já tem um componente portal *web*, responsável por disponibilizar operações de consulta e controlo administrativo. No entanto, encontra-se limitado ao nível de *e-Invoicing* e ao número de países que suporta, pois sempre que é necessário adotar um novo cenário o produto sofre alterações significativas para gerar as validações e configurar o formato dos ficheiros.

2.2 Processamento e Validação de XML

O XML é um *standard* concebido para facilitar a troca e armazenamento de informações entre sistemas de forma estruturada. É auto-descritivo e o seu processamento visa aceder ou modificar informações presentes no ficheiro.

Existem essencialmente duas abordagens para processar um ficheiro XML. Na primeira, o ficheiro é mapeado integralmente em memória, através da construção de uma árvore que representa o documento (*Parse Tree*). Na segunda abordagem, o ficheiro é processado em *stream* através de *callbacks* e o documento não precisa de ser mapeado totalmente para memória.

Atualmente, o SAFTPRO lida com ficheiros SAF-T que podem chegar até às centenas de gigabytes e portanto a primeira abordagem não é exequível a nível aplicacional. Porém, no âmbito de *e-Invoicing* este obstáculo já não se impõe, porque o formato utilizado é mais leve e representa apenas uma fatura, em vez de um conjunto destas. Logo, terão de ser avaliadas as duas abordagens para perceber a que melhor se enquadra neste caso de uso.

2.2.1 *Standards* de Processamento

As duas metodologias de processamento XML descritas, originaram um conjunto de especificações *standard* que permitem uniformizar as implementações para processamento de XML a nível aplicacional (o anexo II inclui uma comparação das diferentes especificações consideradas).

2.2.1.1 DOM[5]

O processamento ocorre sobre a representação do documento em memória, através da construção do *Document Object Model* (DOM). Inicialmente, o ficheiro é percorrido para gerar todos os objetos, que depois são agregados e disponibilizados em memória numa estrutura hierárquica em árvore.

O acesso aleatório aos elementos e a sua respetiva modificação são suportados de maneira eficiente.

[5] https://dom.spec.whatwg.org/

2.2.1.2 SAX[6]

Disponibiliza uma interface de *stream* que analisa o ficheiro sequencialmente. O processamento inicia no topo do documento e termina no elemento *ROOT*, sem a criação do DOM e à medida que o ficheiro é processado são gerados eventos.

Este tipo de processamento designa-se *Push Model* e tipicamente não permite controlar o fluxo de processamento. Ou seja, quando se começa a percorrer o documento, este é iterado até ao fim e os *handlers* são invocados a cada elemento disponibilizado pelo *parser*.

Esta especificação adequa-se a ficheiros constituídos por elementos que podem ser analisados individualmente ou que não cabem integralmente em memória, como é o caso dos ficheiros SAF-T.

2.2.1.3 StAX

Interpretada como uma especificação intermédia entre a DOM e a SAX, distinguindo-se da última pelo mecanismo que utiliza no processamento de documentos [24, 28].

A StAX utiliza um *Pull Model*, que permite aos *handlers* terem mais controlo no processamento de um documento. Nomeadamente, permite que estes invoquem o *parser*, apenas quando estão prontos para receber o próximo evento, o que lhes confere um maior grau de flexibilidade a nível aplicacional.

2.2.1.4 JAXB

O SAFTPRO é desenvolvido com suporte da linguagem Java, onde a especificação mais utilizada para processamento de ficheiros XML é o JAXB, que se desataca essencialmente por [40]:

- Disponibilizar uma API para processamento através de parsers DOM ou SAX;
- Oferecer um modo de leitura semelhante ao DOM, que evita percorrer a *Parse Tree* para aceder aos elementos;
- Permitir fazer *unmarshalling* de XML, disponibilizando objetos Java às aplicações que refletem os elementos do documento;
- Suportar validações estruturais no processo de *unmarshalling* e gerar árvores mais eficientes, em termos de memória, que as utilizadas pelo DOM.

Unmarshalling corresponde à desserialização dos dados em XML para uma Java *Content Tree*, com a subsequente capacidade para validar a informação durante o processo [19]. O inverso designa-se *Marshalling* e recorre à serialização, que corresponde ao processo de transformação dos dados de um objeto num *array* de *bytes*, consoante o formato selecionado. Por sua vez, este formato pode variar desde um simples *array* até uma representação mais elaborada em XML [1].

Existem diversas implementações fiéis a esta especificação que podem ser *Open-source* ou comerciais, cada uma delas com as suas vantagens e desvantagens.

[6] http://www.saxproject.org/

2.2.2 *Standards* de Validação[7]

Em XML, os documentos corretos sintaticamente denominam-se *Well-Formed* e possuem estrutura suficiente para ser representados numa árvore hierárquica. As especificações da secção 2.2.1 dependem desta premissa para poder processar os documentos.

No entanto, para que um documento seja considerado válido é necessário obedecer também a um conjunto de regras que definem a sua gramática. Os ficheiros XML *Schema Definition* (XSD) são utilizados para expressar esta gramática e servem para validar os documentos, mas existem outros *standards* para validação, cada um com a sua particularidade.

2.2.2.1 XSD

A definição de elementos é suportada por tipos simples e complexos, que são interpretados como uma sequência de outros tipos simples ou complexos. Por omissão, existem tipos simples definidos como `int`, `string` e `date`. Mas, é possível definir outros tipos, mais restritos na gama de valores que aceitam ou no tamanho dos valores em si.

O XSD torna a composição de *schemas* modular, o que permite a sua reutilização noutros contextos, através de *imports*. Em simultâneo, suporta herança nos tipos complexos e permite implementar alguma lógica de validação diretamente no *schema*, contribuindo para a redução do código aplicacional dedicado à validação de ficheiros XML.

A listagem 1, exemplifica a estrutura de um ficheiro XSD. Por sua vez, a listagem 2, representa um documento que obedece a esta estrutura.

```
1  <schema xmlns="http://www.w3.org/2001/XMLSchema"
2      xmlns:lh="http://liquidhub.com/SimpleList"
3      targetNamespace="http://liquidhub.com/SimpleList" elementFormDefault="qualified">
4      <complexType name="SimpleList">
5          <sequence>
6              <element name="Item" type="string" maxOccurs="unbounded" />
7          </sequence>
8          <attribute name="name" type="string" />
9      </complexType>
10     <element name="List" type="lh:SimpleList"/>
11 </schema>
```

Listagem 1: list.xsd

```
1  <?xml version="1.0" ?>
2  <List name="Fruit List" xmlns=http://liquidhub.com/SimpleList
3      xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
4      xsi:schemaLocation="http://liquidhub.com/SimpleList list.xsd">
5      <Item>Apple</Item>
6      <Item>Banana</Item>
7      <Item>Pear</Item>
8  </List>
```

Listagem 2: Documento XML

[7] Subsecção estruturada com base em [38, 60].

2.2.2.2 DTD

O *Document Type Definition* (DTD) permite controlar a estrutura geral de um documento, assim como os seus elementos e atributos.

O poder expressivo dos DTDs é um pouco limitado e existem alguns problemas com *namespaces*. Pois na sua perspetiva, os prefixos de um *namespace* correspondem a uma parte da `String` que compõe o nome de um elemento.

```
<!ELEMENT List (Item+)>
<!ATTLIST List
    name CDATA #IMPLIED >
<!ELEMENT Item (#PCDATA)>
```

Listagem 3: list.dtd

A título de exemplo, o DTD presente na listagem 3, permite validar o seguinte documento:

```
<?xml version="1.0" ?>
<!DOCTYPE List SYSTEM "list.dtd">
<List name="Fruit List">
    <Item>Apple</Item>
    <Item>Banana</Item>
    <Item>Pear</Item>
</List>
```

Listagem 4: Documento XML

2.2.2.3 *Relax* NG[8]

O RELAX NG possui um bom balanço entre poder expressivo e facilidade de utilização. Foi criado para dar resposta à complexidade do XSD, pelo que os seus *schemas* podem ser expressos num formato semelhante ao XSD, ou num formato alternativo mais compacto e simples (listagem 5).

```
element List {
    attribute name { text },
    element Item { text }+
}
```

Listagem 5: Formato compacto list.rng

2.2.2.4 Schematron[9]

O Schematron utiliza um paradigma *rule-based*, que difere um pouco da abordagem utilizada nas gramáticas regulares que caracterizam os *standards* vistos anteriormente. Nomeadamente, permite descrever *schemas* através de asserções *XML Path Language* (XPath)[10]

[8] https://relaxng.org/
[9] https://schematron.com/
[10] https://www.w3.org/TR/1999/REC-xpath-19991116/

(listagem 6), que depois são avaliados sobre os documentos XML [3].

```
<schema xmlns="http://www.ascc.net/xml/schematron">
    <pattern name="No Duplicate Item">
        <rule context="Item">
            <assert test="not(preceding-sibling::Item=.)">
                Duplicate items not allowed!
            </assert>
        </rule>
    </pattern>
</schema>
```

Listagem 6: Regra Schematron

A maioria das restrições expressas em XSD, DTD e Relax NG são suportadas. Mas, existe a possibilidade de expressar outras regras semânticas que não são intuitivas de definir nestes standards. Por exemplo, embora o XML inclua um mecanismo de ID/IREF que permite a referenciação cruzada entre elementos, estes encontram-se ligados de uma maneira fraca. Ou seja, é possível assegurar a unicidade dos IDs e garantir que os IDREF apontam para IDs existentes. No entanto, não existe maneira de garantir que um determinado IDREF aponta para um ID específico, num determinado tipo de elemento [10].

Além disso, os *schemas* definidos por gramáticas regulares também limitam a expressividade relativamente às relações entre elementos. Uma vez que, as restrições que se estabelecem focam apenas na noção de *child* e *sibling*. Contudo, o XPath fornece uma sintaxe que permite permite explorar outros tipos de relações, conhecidas como *axes*, que podem ser estabelecidas entre elementos e atributos de um documento XML [10].

Muitas implementações de validações em Schematron sofrem um processo de transformação e são disponibilizadas sobre *stylesheets* XSL (fig. 2.3).

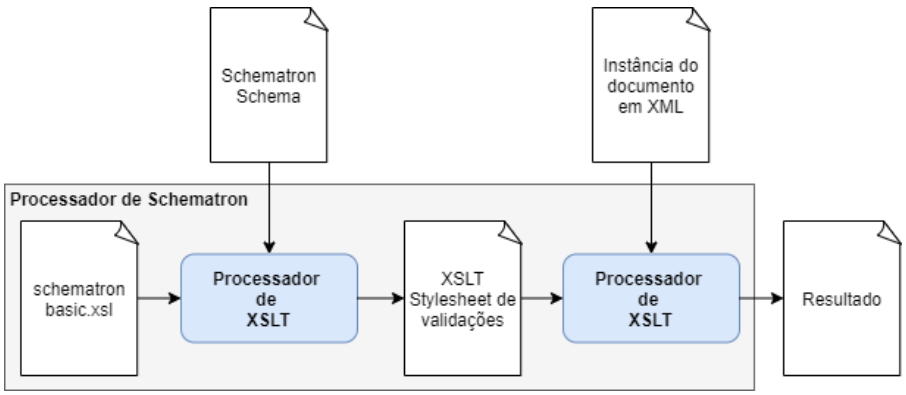


Figura 2.3: Transformação de Schematron em XSL *Transform* [50]

Os ficheiros resultantes são depois utilizados para confrontar as instâncias do documento a validar e na presença de erros produzem uma lista com as suas descrições, em texto simples ou XML.

A *eXtensible Stylesheet Language* (XSL) exerce uma função semelhante à do CSS nos documentos HTML. Ou seja, especifica como o documento XML deve ser representado.

Por exemplo, no HTML existem *tags* pré-definidas customizáveis através de CSS e com base nesta informação o *browser* sabe como as dispor no ecrã.

Por último, as *eXtensible Stylesheet Language Transformations* (XSLT) permitem converter automaticamente documentos XML em outros formatos [6].

2.3 UBL 2.1[11]

O XML é amplamente utilizado na troca de informação em vários setores e originou versões específicas de documentos básicos, como as faturas. Estes documentos específicos permitem otimizar os contextos de negócio, mas a coexistência de formatos para o mesmo propósito, em domínios diferentes, apresenta algumas desvantagens, nomeadamente ao nível de manutenção que é necessário para acomodar todas as versões dos documentos comerciais. Para além da duplicação do esforço, a coexistência de vários formatos dificulta a integração com os sistemas desenvolvidos ou em desenvolvimento.

A *Universal Business Language* foi projetada para endereçar este problema e oferece uma sintaxe de documentos comerciais compreendida universalmente. Além disso, define um formato de intercâmbio genérico que pode ser estendido para acomodar requisitos específicos de cada setor e fornece uma infraestrutura completa que pode estender os benefícios dos sistemas EDI existentes para empresas de todos os tamanhos.

Como biblioteca, disponibiliza *Business Information Entities* que são agrupadas para construir documentos específicos, como o caso das faturas. Posteriormente, estes modelos podem ser transformados desde que respeitem a nomenclatura e as regras de *design* UBL, o que facilita e agiliza a criação de documentos não especificados na biblioteca "comum".

2.3.1 Biblioteca “Comum”

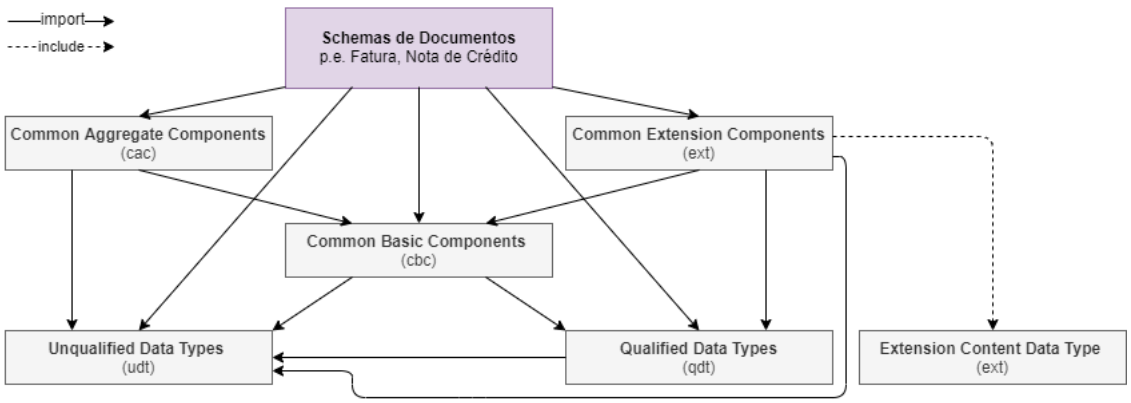


Figura 2.4: Composição da biblioteca UBL 2.1

[11] Secção estruturada com base em [34, 35].

Corresponde à única representação normativa de documentos UBL 2.1 e os seus respetivos componentes, para fins de validação e conformidade (fig. 2.4).

2.3.2 Como projetar para conformidade

2.3.2.1 Subconjuntos

Os tipos de documentos comuns foram projetados para acomodar uma ampla variedade de contextos. Como resultado, se todos os elementos opcionais forem instanciados, o documento resultante será extremamente detalhado. Por exemplo, um documento correspondente a uma encomenda instanciado na totalidade terá aproximadamente 800.000 elementos e atributos. No entanto, a maioria das implementações não precisa de todas as entidades e portanto é possível utilizar subconjuntos de um documento, o que permite remover do modelo quaisquer componentes opcionais que não são necessários para satisfazer os requisitos específicos de uma implementação.

Deve ter-se atenção à utilização de subconjuntos, pois estes só podem ser utilizados para remover elementos opcionais ou alterar a cardinalidade de maneira a que não seja reduzido o número mínimo de ocorrências e/ou estendido o número máximo de ocorrências de um elemento.

2.3.2.2 Restrições de listas de códigos

Restringir valores de uma entidade a um conjunto fixo é um requisito bastante comum e em UBL existem dois tipos de restrições a este nível:

- **Listas de códigos sem valores definidos**: listas sem restrições que podem ter um número infinito de valores, restritos apenas pela sua forma lexical;
- **Listas de códigos com valores definidos**: listas que restringem os valores do conteúdo e que correspondem a especializações das listas de códigos sem valores definidos.

2.3.2.3 Outras restrições

Existem outras situações que requerem customização para restringir valores, nomeadamente as restrições de co-ocorrência, que se aplicam quando valores de um ou mais campos são afetados por valores de outros presentes no documento. Estas situações envolvem a presença ou ausência de um elemento, assim como valores particulares do mesmo, por exemplo:

- O valor total de uma encomenda não pode exceder 100.000€;
- Ou, o endereço de entrega deve ser o mesmo que o endereço de cobrança.

Em UBL, este tipo de restrições é especificado em ficheiros Schematron, porque não é intuitivo defini-las em ficheiros XSD.

2.3.2.4 *Core Invoice Usage Specifications*

Para a representação da fatura em UBL, uma *Core Invoice Usage Specification* (CIUS) corresponde a um subconjunto do modelo e pode ser comparada a uma *guideline* de implementação. Naturalmente, as regras estabelecidas para os subconjuntos devem ser respeitadas, sem adicionar quaisquer termos ainda não definidos. Isto é, que exigiriam uma extensão.

Ao seguir uma CIUS, é importante garantir que a fatura está em conformidade com a sua forma restrita e não restrita (fig. 2.5). Na sua essência, o tipo de restrições que pode ser definido resume-se a proibir elementos com cardinalidade opcional, tornar elementos obrigatórios, limitar a cardinalidade de elementos, modificar o tipo de dados para uma representação mais restritiva e limitar os valores das listas de códigos, adicionar regras de negócio extra ou tornar mais restritivas as que já existem, restringir o tamanho dos campos, o número de casas decimais ou o formato dos valores.

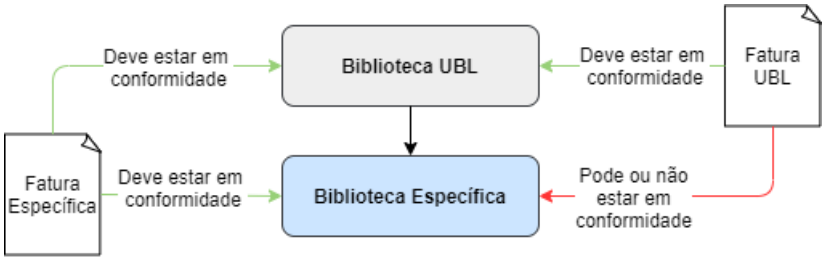


Figura 2.5: Conformidade em UBL 2.1

2.3.3 Como projetar para compatibilidade

O objetivo é reutilizar o máximo possível do modelo UBL, mas quando isto não é possível pode-se criar extensões personalizadas que adicionam novas entidades.

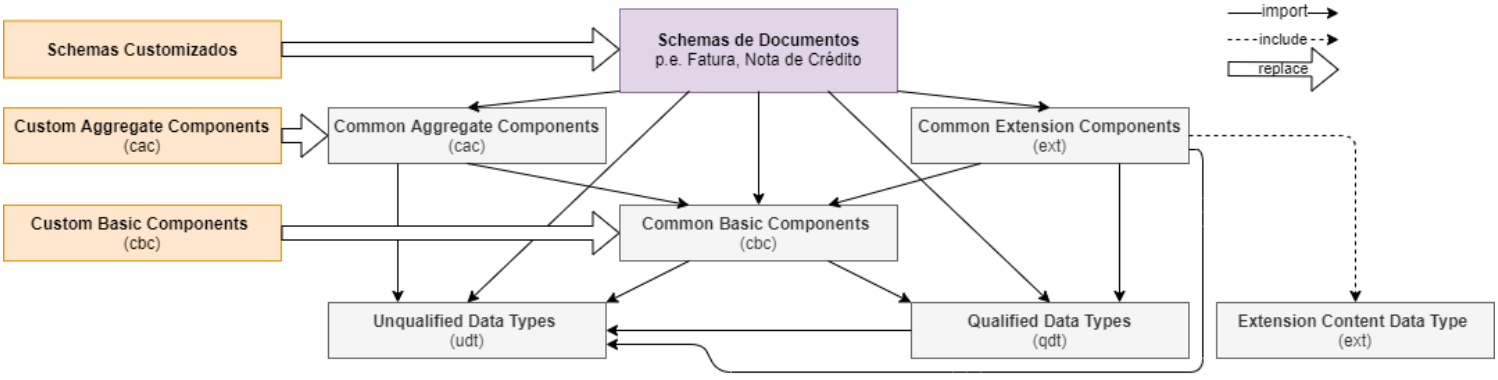


Figura 2.6: Customização UBL 2.1 a nível abstrato

Uma das abordagens para garantir compatibilidade é criar um superconjunto da entidade que queremos estender e incluir a entidade original como associação. Quando uma

entidade não existe, deve-se utilizar a biblioteca de componentes para a criar de modo a assegurar a interoperabilidade.

Para efeitos de customização em subconjuntos, pode-se fazer uma incisão diretamente nos *schemas* UBL e remover todas as entidades desnecessárias. A figura 2.6 apresenta outra abordagem, que consiste em trabalhar a um nível abstrato e sintetizar os fragmentos de um *schema*. Posteriormente, estas modificações são organizadas de modo a garantir que os *schemas* customizados se sobrepõe aos originais no pacote da distribuição UBL.

2.3.4 Validação

Os *schemas* disponibilizados suportam a troca de informação entre vários negócios e é exigido um alto grau de precisão para garantir que o processamento efetuado pelas aplicações e ações comerciais reflitam a finalidade, intenção e conteúdo da informação acordada pelos parceiros comerciais.

A UBL propõe um modelo de validação, dividido em duas etapas, para sistemas que recebem instâncias de documentos baseados na sua biblioteca. As restrições estruturais e lexicais são expressas num ficheiro XSD e as restrições semânticas são expressas num ficheiro XSLT ou Schematron.

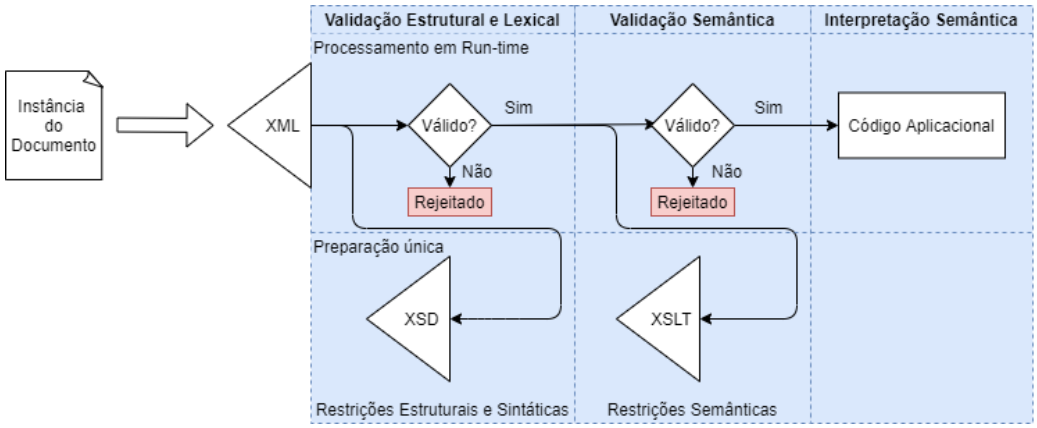


Figura 2.7: Modelo de Validação UBL 2.1

O processo é sequencial visto que só faz sentido validar o conteúdo dos dados se estes estiverem bem estruturados e em qualquer etapa uma falha implica a rejeição do documento. Caso a aplicação necessite de assegurar a validade do documento para carregar estruturas de dados, essa garantia é obtida na primeira etapa.

As validações semânticas desempenhadas na segunda fase evitam que a aplicação precise de saber as restrições aplicadas ao documento, o que permite generalizar a aplicação para suportar diferentes valores possíveis. Isto significa que, a aplicação não precisa de ser modificada caso as restrições sejam alteradas em contextos diferentes.

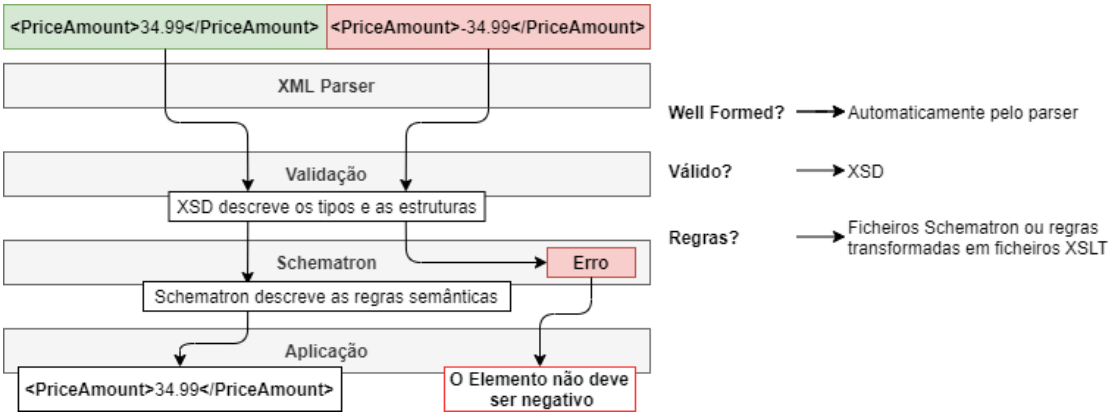


Figura 2.8: Exemplo de validação UBL 2.1 [48]

Conclusões

Nas condições necessárias identificadas para o contexto da tese de *e-Invoicing*, um dos objetivos consiste em gerar através de ficheiros XSD o modelo de dados para suportar o desenvolvimento da solução. O formato base para dar suporte à estrutura das faturas é o UBL 2.1 e foi escolhido pela sua modularidade e mecanismos de extensão.

O UBL foi explorado para entender a sua terminologia e perceber como poderá ser utilizado para atingir os objetivos definidos. Desta análise foi possível concluir que o formato encaixa no contexto SAFTPRO, visto que os documentos em UBL são compostos por *Business Information Entities* que podem ser mapeadas diretamente em objetos Java, o que nos dá imensa flexibilidade e facilita o processo de integração com o modelo de dados. No entanto, os tipos de documentos presentes na biblioteca UBL foram projetados para acomodar uma ampla variedade de contextos. Como resultado, se todos os elementos opcionais forem instanciados, o documento resultante será extremamente detalhado e poderá conter dependências cíclicas, tornando impraticável a geração do modelo de dados a partir da representação base da fatura em UBL.

Subjacente a este objetivo está a integração do modelo base UBL 2.1 com a versão 2.0.0 do *schema* para ficheiros SAF-T. Inicialmente, foi feita uma tentativa de mapeamento (fig. 2.9) para averiguar as implicações da integração dos dois modelos, mas surgiram algumas complicações pelo facto do formato UBL 2.1 e o SAF-T versão 2.0.0 terem contextos de utilização diferentes. O primeiro é mais direcionado para o envio unitário de faturas, enquanto o segundo tende a ser utilizado para acoplar conjuntos destas. Mais, constatou-se que na sua essência existem poucos campos obrigatórios em comum com mapeamento direto. Muitos deles têm cardinalidade diferente ou são incompatíveis estruturalmente, isto é, encontram-se em sítios diferentes nas suas representações, têm nomes diferentes ou os tipos de dados utilizados não são iguais. Portanto, chegou-se à conclusão que tentar integrar dois *standards* diferentes cria uma dependência muito difícil de concretizar num cenário genérico e leva a problemas de manutenção ao longo do tempo.

UBL 2.1 (XPath)	Cardinalidade	SAF-T 1.0 (XPath)
Invoice/cbc:ID	1	InvoiceStructure/InvoiceNo
Invoice/cbc:CopyIndicator	0..1	N/A
Invoice/cbc:UUID	0..1	InvoiceStructure/SystemID
		InvoiceStructure/BatchID
Invoice/cbc:IssueDate	1	invoiceStructure/InvoiceDate
Invoice/cbc:IssueTime	0..1	N/A
Invoice/cbc:DueDate	0..1	N/A
Invoice/cbc:InvoiceTypeCode	0..1	InvoiceStructure/InvoiceType
Invoice/cbc:Note	0..1	N/A
Invoice/cbc:TaxPointDate	0..1	N/A
Invoice/cbc:DocumentCurrencyCode	0..1	HeaderStructure/DefaultCurrencyCode
Invoice/cbc:TaxCurrencyCode	0..1	InvoiceStructure/DocumentTotals/TaxInformationTotals/TaxAmount/CurrencyCode
Invoice/cbc:PricingCurrencyCode	0..1	*** Confirmar este campo ***
Invoice/cbc:PaymentCurrencyCode	0..1	*** Confirmar este campo ***
Invoice/cbc:PaymentAlternativeCurrencyCode	0..1	N/A
Invoice/cbc:AccountingCostCode	0..1	*** Confirmar este campo ***
Invoice/cbc:AccountingCost	0..1	*** Confirmar este campo ***
Invoice/cbc:LineCountNumeric	0..1	N/A
Invoice/cbc:BuyerReference	0..1	N/A
Invoice/cac:InvoicePeriod	0..n	
Invoice/cac:InvoicePeriod/cbc:StartDate	0..1	InvoiceStructure/PeriodYear
Invoice/cac:InvoicePeriod/cbc:StartTime	0..1	N/A
Invoice/cac:InvoicePeriod/cbc:EndDate	0..1	N/A

Figura 2.9: Tentativa de mapeamento de UBL para SAF-T

Para endereçar estes problemas propõe-se a geração distinta dos dois modelos de dados, através da definição do XSD UBL 2.1 e SAF-T, assim como de toda a infraestrutura de suporte para gestão de SAF-Ts e faturas. Segundo esta abordagem, os modelos de dados ficam ligados apenas no momento de submissão do SAF-T, através da identificação das respetivas faturas presentes na representação em UBL 2.1 e no ficheiro SAF-T submetido.

A separação do mapeamento e conversão de UBL 2.1 para SAF-T permite-nos implementar a solução mais rapidamente e evoluir independentemente os ficheiros XSD, para produzir uma arquitetura modular com flexibilidade para desativar o módulo de *e-Invoicing*, caso este não seja preciso para a instância concreta de um país.

Voltando ainda ao formato UBL 2.1, constatou-se que a maioria das implementações não precisa de todas as entidades utilizadas e portanto é necessário utilizar subconjuntos do documento, o que nos permite remover quaisquer componentes opcionais que não são necessários para satisfazer os requisitos específicos de uma implementação.

A secção 2.3.2.4 descreve a solução adotada por muitos países, incluindo Portugal, e consiste em definir uma CIUS, que corresponde a um subconjunto do modelo e pode ser comparada a uma *guideline* de implementação. Para além disso, oferece a garantia de que uma fatura em conformidade com a CIUS estará de certeza em conformidade com o modelo base da fatura em UBL 2.1 (o contrário não é verdade).

Para a definição da CIUS não ser *ad-hoc*, foi sugerido adotar a norma europeia EN16931[12], que na sua essência confere significado aos campos do modelo e especifica quais devem ser utilizados. Neste caso, o problema é que não existe XSD referente à norma EN16931, nem das CIUS que derivam desta, nomeadamente a Portuguesa. Ou seja, o que acontece na realidade é que em termos estruturais os ficheiros de uma CIUS validam estruturalmente com o XSD do modelo base do UBL 2.1 e depois todas as restrições de elementos que devem ou não estar presentes, assim como qualquer lógica de negócio são expressos em ficheiros Schematron, dos quais é impossível gerar o modelo de dados. Consequentemente, percebeu-se que a premissa é apostar na interoperabilidade e adotar "*Open Standards*" como o Peppol[13].

2.4 Geração Dinâmica de Código[14]

Através da geração de código podemos obter pequenas porções de código ou contextos aplicacionais inteiros. A sua utilização simplifica o processo de desenvolvimento e aumenta a produtividade, portabilidade e consistência do código produzido, visto que:

- Ao escrever o gerador de código uma vez, ele pode ser reutilizado vezes sem conta para um determinado contexto, o que permite poupar tempo no desenvolvimento de *software*;
- Depois de desenvolver o gerador de código para uma determinada linguagem de programação é possível trocar ou incrementá-lo para suportar diferentes linguagens e ferramentas;
- Por último, a qualidade do código gerado é sempre consistente, assumindo que não há *bugs* no gerador.

Não obstante, existem alguns inconvenientes para as técnicas de geração de código. Por exemplo, a partir do momento em que utilizamos uma ferramenta o código fica dependente da mesma, o que pode levar a problemas de manutenção no futuro. Além disso, existe a complexidade associada à utilização das ferramentas e o facto das mesmas poderem gerar mais código do que o necessário, ou código que não está otimizado.

As técnicas de geração de código permitem gerar código repetitivo a partir de *schemas* ou qualquer outra fonte de informação. Um exemplo muito comum, consiste em gerar os *Data Access Objects* com base num modelo de dados já existente. Contudo, é possível gerar código mais complexo, como o esqueleto de uma aplicação ou contextos aplicacionais inteiros através de Domain Specific Languages (DSLs).

As DSLs são linguagens adaptadas a domínios aplicacionais específicos, conferindo-lhes ganhos substanciais de expressividade e facilidade de uso, nesses mesmos domínios,

[12] https://standards.cencenelec.eu/dyn/www/f?p=205:110:0::::FSP_ORG_ID,FSP_LANG_ID:1883209,25&cs=18F2559A05E966F8D6BA2CD11622D2977

[13] https://peppol.eu/

[14] Secção estruturada com base em [23, 54].

quando comparadas com as linguagens de programação convencionais. O *trade-off* é essencialmente a generalidade da linguagem pela sua expressividade [33], de modo a criar camadas de abstração que encapsulem as metodologias de desenho e conceção de um determinado domínio técnico [55]. Muitas das vezes os geradores de código utilizam DSLs como *input*, o que lhes permite reutilizar as semânticas de um domínio, mas a sua importância também é ampla na construção de sistemas mais complexos. Nestes cenários, a sua contribuição é permitir a reutilização de artefactos de software [55].

Tipicamente, o objetivo das DSLs é aumentar a produtividade dos engenheiros de software e portanto o seu "sucesso"pode ser quantificado pelo impacto que tem na evolução do processo de desenvolvimento. Primeiro, é importante que as aplicações consideradas adequadas ao domínio possam ser expressas na DSL. Esta métrica, pode ser expressa como a completude da DSL ou a sua cobertura do domínio. Além disso, a construção de uma aplicação com a DSL deve exigir um esforço substancialmente menor do que com outros meios. Esta métrica pode ser aproximada, com o número de linhas de código (LOC) necessárias para construir a aplicação em comparação com o que seria necessário noutra linguagem [55].

2.4.1 Tipos de *Pipeline*

Existem diferentes maneiras de desenhar um *pipeline* para geração de código, mas todas dependem de dois elementos:

- ***Input*:** que corresponde à informação necessária para gerar o código. Os exemplos mais simples são as DSLs, *Data Sources* ou informação obtida através de *Reverse engineering*;
- ***Output*:** que especifica como será obtido o código gerado. Para esta finalidade são utilizadas muitas vezes *template engines*, que disponibilizam APIs para criação de objetos Java programaticamente, preenchimento de HTML, etc.

Um *pipeline* pode ser uma coisa tão simples como um *Parser Generator*, onde o *input* é uma DSL e o *output* é produzido através de um *template engine*, ou o código gerado pelos IDEs, que permitem a geração de *stubs* para implementação de métodos, assim como gualquer *getter* e *setter* de propriedades.

Os tipos de *pipeline* mais relevantes para o tema da tese são o *Model Driven Design* e o *Database-related Code*.

O *Model Driven Design* é disponibilizado habitualmente em *plugins* para IDEs, ou em IDEs *standalone*, que permitem descrever o modelo de uma aplicação a partir do qual é possível gerar o seu esqueleto ou toda a sua infraestrutura.

O *Database-related Code* deriva do *Model Driven Design* e o programador normalmente define o modelo de dados, a partir do qual é gerada a aplicação com as operações CRUD ou código para manipular a base de dados. Algumas ferramentas também disponibilizam

o inverso, ou seja, a partir de uma base de dados é criado o modelo ou o código para a gerir.

2.4.2 *Template Engine*

Um *template engine* é equivalente a um mini-compilador, que processa uma linguagem de *template* simples e assenta em ficheiros com notações especiais que são interpretadas. A funcionalidade mais simples consiste em substituir estas anotações por informação relevante e apropriada, em tempo de execução. A maioria destes mecanismos possui também alguns comandos para controlo de fluxo, por exemplo *loops* e declarações *if-else*, que permitem descrever estruturas simples.

O Apache Velocity[15] é um dos mais utilizados em Java e é composto por diversas ferramentas. No seu *core* está o Velocity *Engine*, que permite gerar código-fonte a partir de um *template* definido na Velocity *Template Language*. Esta linguagem possui uma enorme flexibilidade e é composta por uma sintaxe simples que fornece *references*, como formalismo para aceder a objetos no ambiente Velocity e *directives*, que são um conjunto de instruções utilizadas para controlo de fluxo e ação.

```
1 Hello $name!  Welcome to Velocity!
```

Listagem 7: helloworld.vm

```
1  public class HelloWorld
2  {
3      public static void main( String[] args )
4          throws Exception
5      {
6          VelocityEngine ve = new VelocityEngine();
7          ve.init();
8          Template t = ve.getTemplate( "helloworld.vm" );
9          VelocityContext context = new VelocityContext();
10         context.put("name", "World");
11         StringWriter writer = new StringWriter();
12         t.merge( context, writer );
13         System.out.println( writer.toString() );
14     }
15 }
```

Listagem 8: Velocity Engine

O Velocity é simples de utilizar e permite gerar código para qualquer linguagem de programação, o que pode ser considerado uma desvantagem visto que não possui qualquer conhecimento do que está a ser gerado. Por exemplo, pode estar a utilizar um *template* para gerar um *Controller*, sem ter qualquer conhecimento do conceito *Controller* no padrão *Model-View-Controller*[16].

[15] https://velocity.apache.org/

[16] https://folk.universitetetioslo.no/trygver/themes/mvc/mvc-index.html

2.4.3 Exemplos de ferramentas

Esta subsecção divulga algumas ferramentas que existem para efeitos de contextualização.

2.4.3.1 Acceleo[17]

Plugin para o Eclipse IDE, que fornece ferramentas para gerar código Java a partir de um modelo da ferramenta Eclipse *Modeling Framework*[18], que pode ser definido através de um diagrama UML[19] ou de uma DSL customizada.

Figura 2.10: Acceleo - Visão global [56]

2.4.3.2 Celerio[20]

Ferramenta em Java que possui um *database extractor*, através do qual é possível extrair o modelo de uma base de dados já existente. Posteriormente, o modelo é utilizado em associação com ficheiros de configuração para instanciar o *template engine* e gerar código consoante os *templates* disponibilizados.

Os *templates* utilizados por esta ferramenta são *templates* Velocity.

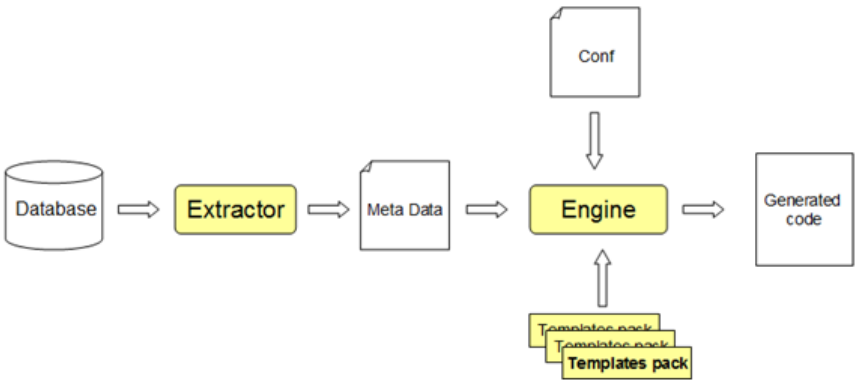


Figura 2.11: Celerio - Visão global [4]

[17] https://www.eclipse.org/acceleo/
[18] https://www.eclipse.org/modeling/emf/
[19] https://www.uml.org/
[20] https://www.jaxio.com/en/celerio.html

2.4.3.3 Telosys[21]

Ferramenta utilizada com frequência para dar *Bootstrap* a projetos.

Possui dois modos diferentes para geração de código, mas que são utilizados de maneira semelhante através de um modelo intermédio genérico.

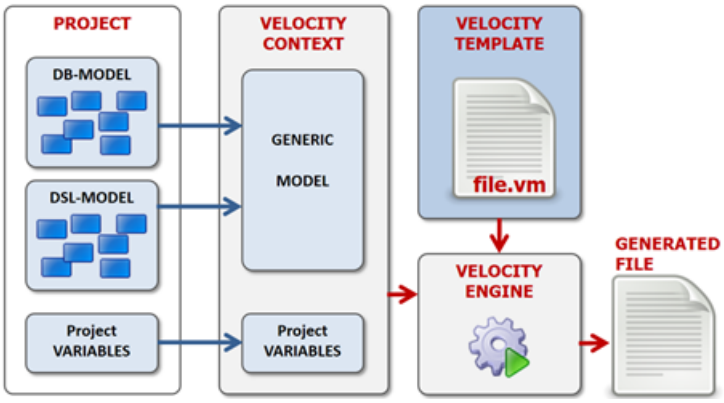


Figura 2.12: Telosys - Visão global [53]

O primeiro modo pode ser utilizado quando já existe uma base de dados relacional. Neste caso, o Telosys exporta diretamente todas as tabelas e respetivas relações, através de uma conexão à base de dados. Posteriormente, esta informação é guardada num ficheiro XML, que faz corresponder uma nova *entity* a cada tabela da base dados.

O segundo consiste em utilizar uma sintaxe muito simples e especifica da ferramenta para descrever as *entities* e as suas relações (a listagem 9 exemplifica a definição de um livro).

```
1  Book {
2    id : int { @Id, @AutoIncremented };
3    title : string { @SizeMax(255) };
4    year : short { @Min(1950), @Max(2025) };
5    price : float { @Min(0), @Max(200) };
6    publisher : Publisher { @NotNull };
7    authors : Author[];
8  }
```

Listagem 9: book.entity

Após definido o modelo, este é carregado juntamente com as variáveis de projeto num formato genérico para o contexto de execução do Velocity, que contêm todos os objetos que podem ser utilizados no *template*. Posteriormente, o gerador combina o *template* e o modelo presente no contexto Velocity para gerar o código-fonte final.

[21] https://www.telosys.org/index.html

2.5 Peppol

O acrónimo "PEPPOL" significa *Pan-European Public Procurement On-Line* e reflete o objetivo inicial do projeto desenvolvido pela associação OpenPEPPOL, em 2008, com o financiamento da Comissão Europeia[58].

Inicialmente, o seu propósito era facilitar o comércio entre Estados membros da União Europeia. Mas, ultrapassou essa expetativa e atualmente pode ser visto como uma estrutura que permite a troca simples e segura de documentos comerciais eletrónicos entre entidades públicas e privadas, um pouco por todo o mundo [2]. Como consequência, o acrónimo foi substituído recentemente pela marca "Peppol" de modo a refletir o seu uso a nível internacional e reforçar o *motto* "Connect once, connect to all" [58].

O Peppol oferece uma rede *e-Delivery* e os respetivos *Transport Infrastructure Agreements* (TIAs) responsáveis pela base legal e técnica para a interoperabilidade, assim como as *Business Interoperability Specifications* (BIS) que correspondem aos tipos de documentos que podem circular na rede. Uma delas é a CIUS Peppol BIS *Billing* 3.0 que deriva da norma europeia EN16931 e facilita a colaboração, assim como a implementação da faturação eletrónica.

Dito isto, não se deve interpretar o Peppol com uma plataforma para a troca de documentos, mas sim como uma ferramenta e infraestrutura que permite ligar todas as plataformas já existentes utilizadas pelas empresas e administrações públicas [2].

Em suma, podemos compartimentalizar o Peppol em três pilares[59]:

- A rede (Peppol *e-Delivery Network*);
- O quadro legal que rege a administração da rede (PEPPOL *Transport Infrastructure Agreements*);
- E as especificações dos documentos (Peppol *Business Interoperability Specifications*).

2.5.1 Peppol *e-Delivery Network*

À semelhança do *CEF e-Delivery*[22], a rede *e-Delivery* do Peppol utiliza um modelo *4-Corners*, onde os participantes utilizam um *Access Point* (AP) à sua escolha para se ligarem à rede, que depois se encarrega da troca de mensagens com o parceiro comercial do participante, através de um outro *Acess Point* escolhido pelo parceiro [49]. Em simultâneo, o Peppol estabelece o conjunto de processos e normas que devem ser cumpridas na rede. Como resultado, obtém-se uma rede segura e interoperável sobre o mesmo protocolo de transporte e formato de mensagens.

Os *Access Points* poderão disponibilizar serviços complementares aos requeridos pelo serviço de *e-Delivery*, tornando a integração com os seus clientes mais cómoda e transparente.

[22] https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eDelivery

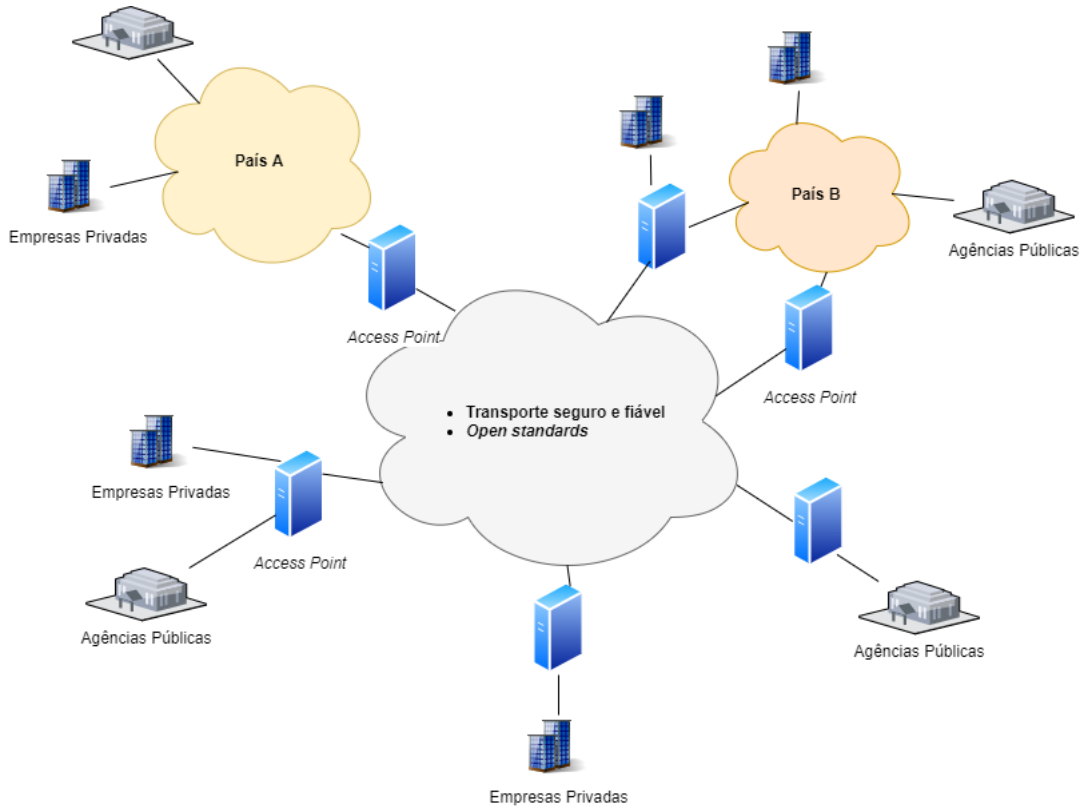


Figura 2.13: Peppol - Visão global da rede

Se ampliarmos o conteúdo da figura 2.13, encontraremos algo semelhante ao que é retratado na figura 2.14. Nesta imagem, é realçada a importância de cada um dos componentes da rede, sendo que cada um tem uma responsabilidade distinta e é indispensável para o seu correto funcionamento.

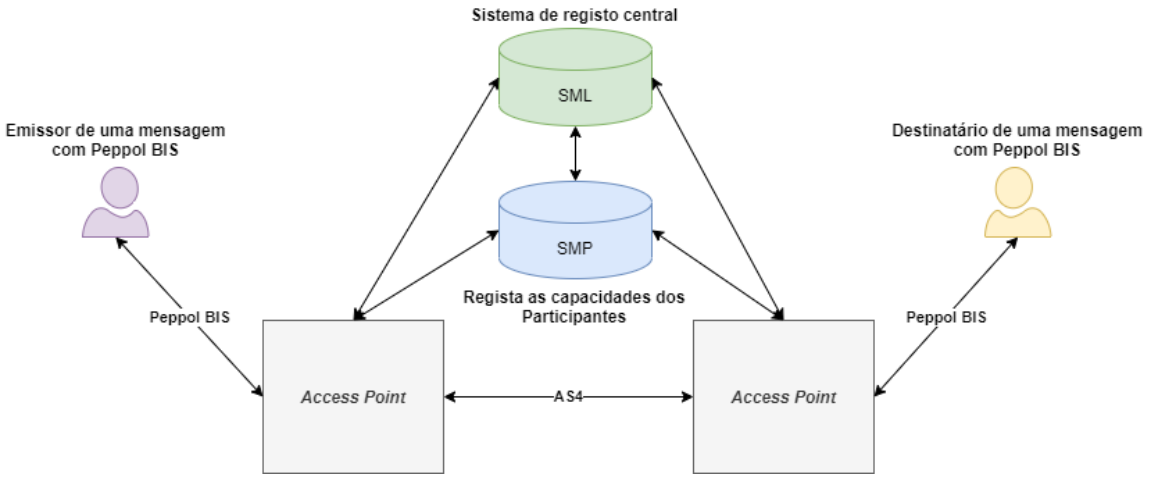


Figura 2.14: Peppol - *4-Corner Model*

Primeiro temos os *Access Points* credenciados, que são capazes de se conectar a qualquer outro *Access Point* da rede Peppol, asseguram que os documentos trocados chegam ao seu destino e estão de acordo com as normas e regras impostas pelo Peppol [26]. De seguida, temos os *Service Metadata Publishers* (SMPs) e o *Service Metadata Locator* (SML).

Os *Service Metadata Publishers* disponibilizam os endereços e metadados dos utilizadores conectados à rede Peppol. Todos os *stakeholders* que fazem parte da rede têm de estar registados num *Service Metadata Publisher*. Tipicamente, este é fornecido como complemento de um *Access Point*, uma vez que são eles que publicam detalhes para os clientes do *Access Point* [49, 46]. No entanto, também podem ser disponibilizados por terceiros.

A grande vantagem de ter um SMP é que o *Access Point* pode gerir a sua própria infraestrutura de identificação[23] na rede Peppol conectada ao SML.

Para a entrega de documentos é fulcral que os *Access Points* tenham acesso a informação sobre todos os outros *Access Points*. Nomeadamente, que participantes são servidos e que tipo de documentos suportam. Esta responsabilidade recai sobre o *Service Metadata Locator*, que corresponde ao único componente central na rede Peppol[49, 45].

No fundo, o *Service Metadata Locator* aponta que *Service Metada Publisher* deve ser consultado para descobrir os detalhes de um participante e pode ser equiparado a um repositório, onde se situam todos os *Access Points* e *Service Metadata Publishers* certificados.

2.5.1.1 *Workflows* Típicos[24]

Na rede Peppol existem três *workflows* diferentes e predominantes:

- Registo de um SMP no SML;
- Registo de um participante na rede Peppol;
- Troca de documentos entre parceiros comerciais.

2.5.1.1.1 Registo de um SMP no SML

De acordo com a figura 2.15, o registo de um SMP divide-se em dois passos:

1. **REGISTER SMP:** de uma perspetiva técnica, consiste em invocar um *webservice* exposto pelo SML, com os metadados do SMP e as credenciais apropriadas (p.e. administrador de sistema);
2. **CREATE SMP:** consiste em criar o DNS *record* para o SMP, com o seu identificador e localização na internet, tornando-o acessível e detectável por terceiros. O DNS *record* (`SMP-ID.publisher.DNSZONE`) pode ser um CNAME *record*, se o URL público for um *domain name*, ou um A *record*, se o URL público for um endereço IP. Em relação à `DNSZONE`, esta corresponde à zona onde o SML opera. Em produção é utilizado o valor "edelivery.tech.ec.europa.eu" e em ambiente de testes o valor é "acc.edelivery.tech.ec.europa.eu".

[23] Consultar anexo V para mais informação sobre os identificadores utilizados no Peppol.
[24] Subsecção estruturada com base em [49, 25].

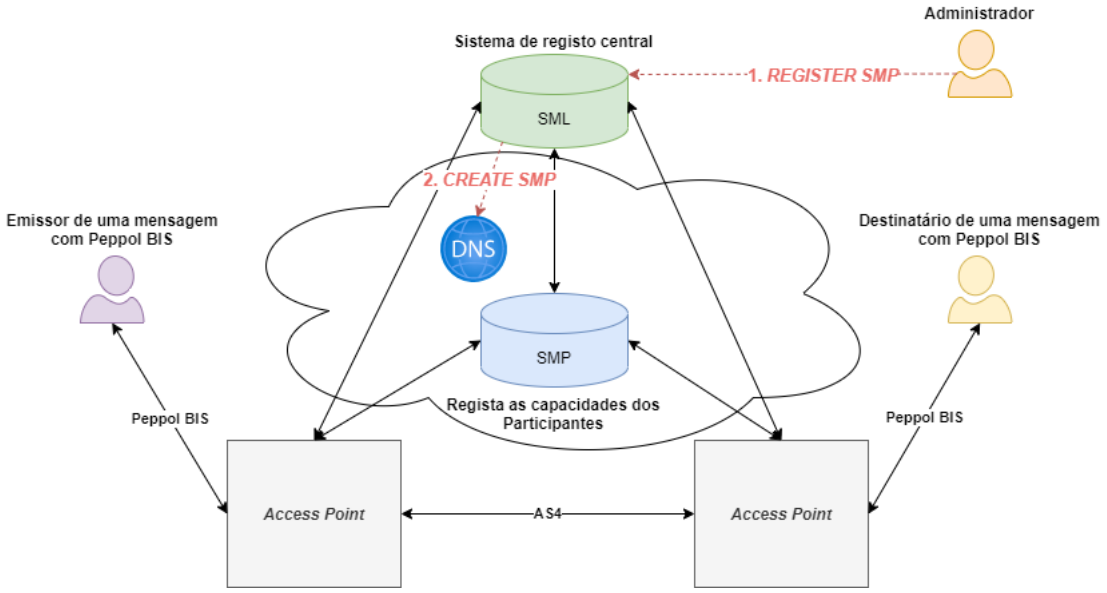


Figura 2.15: Peppol - Registo de um SMP

Este processo de registo ocorre uma única vez e é necessário disponibilizar a seguinte informação:

- Certificado X.509 e identificador do SMP;
- Endereço IP da máquina (endereço físico);
- URL público do SMP (endereço lógico).

2.5.1.1.2 Registo de um participante na rede Peppol

Os Agentes Económicos desempenham o papel de participantes na rede Peppol e uma troca de documentos presume sempre a noção de um *Sender* e um *Receiver*.

No SMP é publicado apenas os documentos suportados pelos *Receivers*. No entanto, o tipo de processo considerado poderá envolver a troca de múltiplos documentos nos dois sentidos. Nestes casos, o *Sender* inicial desempenhará o papel de *Receiver* em algumas interações e portanto também terá de ser servido por um SMP, onde regista as suas capacidades. Contudo, se existir um cenário onde um participante só emite documentos e nunca os recebe pela rede, então não é necessário que este participante se registe perante um SMP.

De acordo com a figura 2.16, o registo de um participante resume-se nos seguintes passos:

1. ***REGISTER RECEIVING PARTY & SUBMIT METADATA*:** Neste passo, o participante e os seus metadados são registados no SMP através de uma invocação REST,

como explicitado na especificação do SMP[25]. Normalmente, os metadados representam um serviço em específico e associam o identificador Peppol do participante com a sua capacidade para receber um determinado tipo de documento, através do protocolo de transporte adequado. Para além disso, os metadados detalham que processos de negócio englobam um determinado documento, os seus restantes dados operacionais e toda a informação necessária para que os *Access Points* possam comunicar entre si;

2. ***CREATE RECEIVING PARTY*:** O passo anterior despoleta automaticamente uma invocação no SML, por parte do SMP, onde é registado o novo participante;
3. ***REGISTER RECEIVING PARTY*:** Por último, dá-se a criação de uma entrada na base de dados do SML e o respetivo DNS *record* para o participante, com a sintaxe `"B-"+hexstring(md5(lowercase(ID-VALUE)))+"."+ID-SCHEME+"."+DNSZONE`. Este aponta para a *publisher entry* que corresponde ao URL, ou endereço IP disponibilizado pelo SMP no seu processo de registo. A `DNSZONE` é semelhante à do processo de registo do SMP, o `ID-SCHEME` corresponde ao *Participant Identifier Meta Scheme* e o `ID-VALUE` é o valor do identificador Peppol do participante.

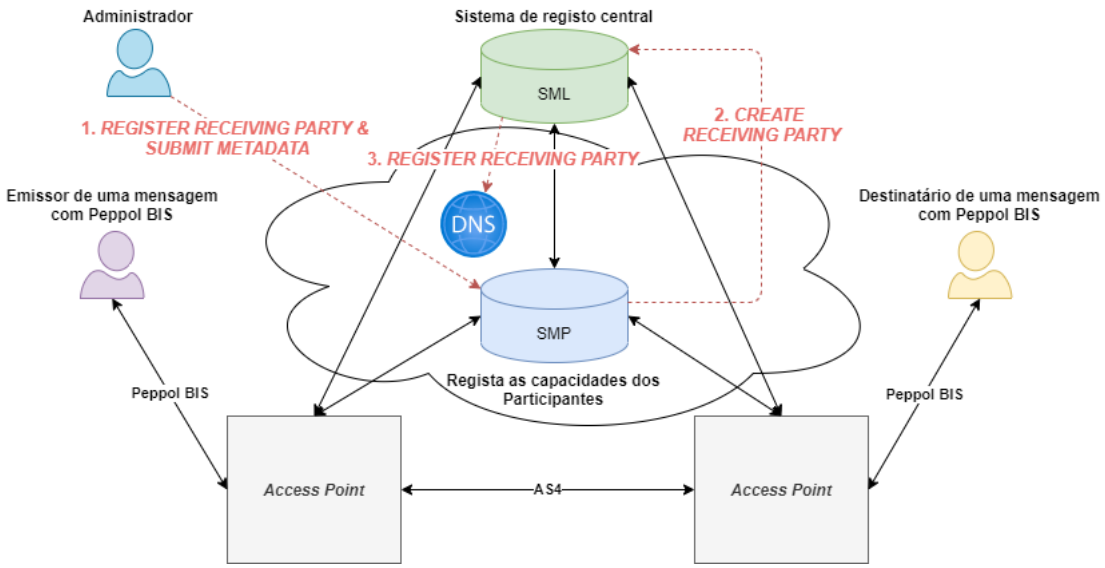


Figura 2.16: Peppol - Registo de um participante

De um modo geral, é necessário disponibilizar a seguinte informação:

- Certificado X.509 e identificador do SMP;
- Identificador Peppol do participante.

[25] Ver [46].

2.5.1.1.3 Troca de documentos entre parceiros comerciais

Antes de iniciar uma troca de documentos é necessário obter a seguinte informação:

- O PPID do *Receiver*, que atualmente é obtido fora de banda, mas pode ser mitigado com a evolução do Peppol Directory[26];
- O SML onde está registado registado o PPID (produção ou ambiente de testes);
- O *Document Type Identifier* e o *Process Identifier*;
- O documento que deve ser trocado;

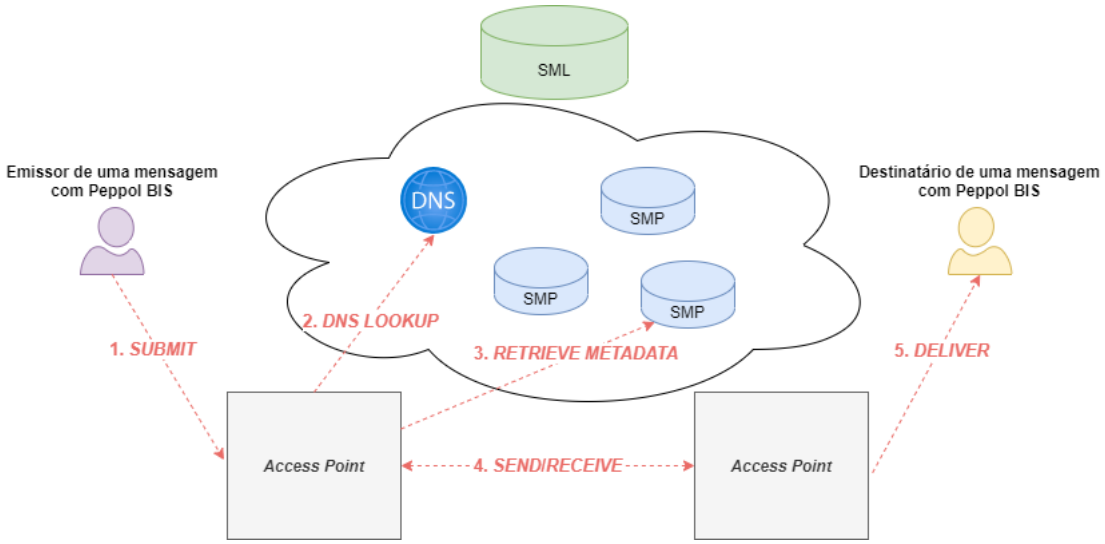


Figura 2.17: Peppol - Troca de documentos

Como exemplo, consideremos o PPID "iso6523-actorid-upis::9915:test" e o SML de testes "DG DIGIT SMK", cuja `DNSZONE` é "acc.edelivery.tech.ec.europa.eu'. Do ponto de vista do *Sender Access Point*, os passos a tomar para a troca de documentos são[27]:

1. Encontrar o SMP onde o *Receiver* registou os seus *endpoints*. Para tal basta construir o URL com base na sintaxe `"B-"+hexstring(md5(lowercase(ID-VALUE)))+"."+ID-SCHEME+"."+DNSZONE`. Neste exemplo em concreto, corresponderia ao URL `http://B-85008b8279e07ab0392da7555856a2.iso6523-actorid-upis.acc.edelivery.tech.ec.europa.eu`, que após resolvido pelo DNS, coincide com o IP do servidor `http://test-infra.peppol.at/` (nome utilizado no processo de registo do SMP);
2. Agora basta interrogar o SMP, através de uma *query* com o tipo de documento desejado através da seguinte sintaxe " `http://smp.url/participant/services/documenttype`". O resultado desta invocação vem assinado e necessita ser processado para

[26] `https://directory.peppol.eu/public`

[27] Os passos descritos omitem as operações de *Submit* e *Deliver* presentes na figura 2.17, como tal a numeração inicia na operação *DNS Lookup*.

obter o *endpoint* correspondente ao *Process Identifier* e protocolo de transporte especificado.

3. O último passo refere-se ao envio do documento para o *endpoint* através do *Access Point* do *Sender*.

É importante realçar que o SML não está envolvido na troca de documentos. Logo, mesmo que este componente centralizado, não esteja a funcionar corretamente, será sempre possível trocar documentos (ver fig.2.17).

Numa rede de qualquer dimensão ou complexidade, é extremamente invulgar encontrar um único componente que, quando falha, quebra as comunicações na rede inteira. Porém, existem dispositivos que são responsáveis por grandes partes da rede.

Um "*Single point of failure*" corresponde a qualquer elemento da rede, que quando falha, impossibilita a comunicação com uma parte da mesma. No entanto, isto só acontece se não existir um sistema de *backup* [11]. Portanto, uma das grandes estratégias para colmatar esta situação é introduzir redundância nos componentes e replicar a informação. Deste modo, após a fase de registo do SMP e dos participantes, o *lookup* é efetuado no DNS, que se encontra replicado na internet e consequentemente ajuda a mitigar a ocorrência de "*Single points of failure*".

2.5.1.2 AS4

O protocolo de comunicação vigente na rede Peppol é o AS4, que corresponde a um perfil da especificação ebXML *Messaging Service* (ebMS) 3.0[28].

As suas principais características são [57]:

- **Interoperabilidade:** introduzido pelo comité técnico da OASIS[29] como um *standard* definido a partir de outros *standards* já existentes, tais como o MIME[30], SOAP[31] e WS-Security[32];
- **Segurança:** utiliza um subconjunto de funcionalidades presentes no WS-Security para assegurar a confidencialidade da informação, mais a sua integridade e não repúdio;
- **Fiabilidade:** utiliza um mecanismo de *acknowledgments* para garantir a entrega das mensagens e evitar duplicados;
- **Flexível ou *Payload-Agnostic*:** permite transportar qualquer tipo de *payload*.

A figura 2.18 retrata de um modo abstrato a troca de mensagens definida na especificação ebMS 3.0. O modelo é bastante modular e promove a separação de responsabilidades entre os diferentes componentes, nomeadamente ao nível dos *Message Service Handlers* (MSHs) responsáveis pela troca das mensagens AS4.

[28] ver [36].
[29] https://www.oasis-open.org/
[30] https://datatracker.ietf.org/doc/html/rfc2045
[31] https://www.w3.org/TR/soap/
[32] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss

Os restantes componentes são o *Producer* que disponibiliza o *payload* a ser trocado e o *Consumer* que o recebe e processa.

Os conceitos *Producer*, *Sender* (*Sending* MSH), *Receiver* (*Receiving* MSH) e *Consumer* não devem ser confundidos com os da topologia *4-Corners* do Peppol.

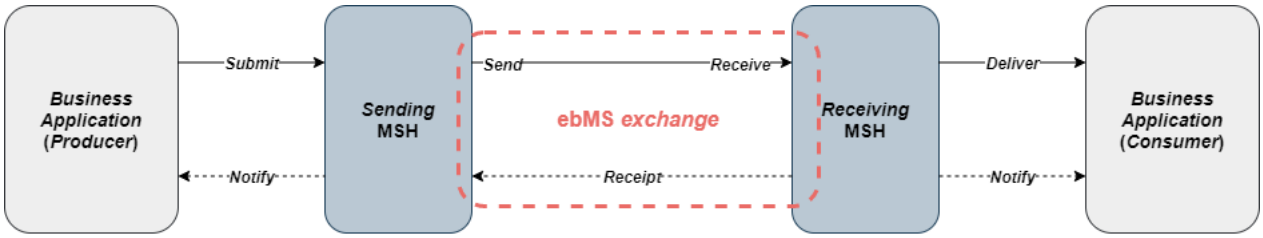


Figura 2.18: ebMS 3.0 - *Abstract Messaging Model*

De acordo com o modelo definido, a troca de mensagens pode ser realizada através de diversos padrões como o *One-Way/Push*, *One-Way/Pull* e *Two-Way/Sync*, que pode ser encarado como uma sequência de dois *One-Way/Push*.

A grande diferença reside no componente que despoleta a troca de mensagens. Isto é, no *One-Way/Push* quem tem iniciativa é o *Sending MSH*, no One-Way/Pull é o *Receiving* MSH e no *Two-Way/Sync* os papéis são invertidos entre trocas, ou seja, após a primeira interação o *Sending* MSH e *Receiving* MSH permutam papéis.

A interação entre os diferentes componentes é estabelecida através de cinco operações abstratas, das quais apenas a *Send* e *Receive* estão no âmbito da especificação ebMS 3.0. A *Submit*, *Delivery* e *Notify* estão relacionadas com a integração dos MSH com as aplicações e portanto poderão ser implementadas livremente pelos provedores de serviços.

Adicionalmente, a especificação ebMS 3.0 define como é que as mensagens devem ser encapsuladas, disponibiliza *guidelines* para o tratamento de erros e questões de segurança[33], o que pressupõe um processo de configuração entre os dois MSH, que antecede a troca de mensagens.

Na versão 3.0 do ebMS, utiliza-se o conceito Processing Modes (P-Modes), que combinam um conjunto de parâmetros para classificar os detalhes da troca de mensagens. Por exemplo, que tipo de identificadores devem ser utilizados para referenciar o emissor e destinatário da mensagem, ou que algoritmo deve ser utilizado para garantir a integridade e autenticidade das mesmas.

De modo a simplificar a interoperabilidade entre sistemas, é possível criar perfis que predefinem alguns dos valores dos parâmetros P-Mode. Deste contexto, surgiu o primeiro perfil AS4[34] definido pela OASIS. Contudo, como acontece no UBL 2.1, os valores predefinidos ainda deixam muitas opções em aberto e portanto é comum que para um

[33] Consultar anexo VI para mais informação sobre estes tópicos.

[34] http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/profiles/AS4-profile/v1.0/AS4-profile-v1.0.html

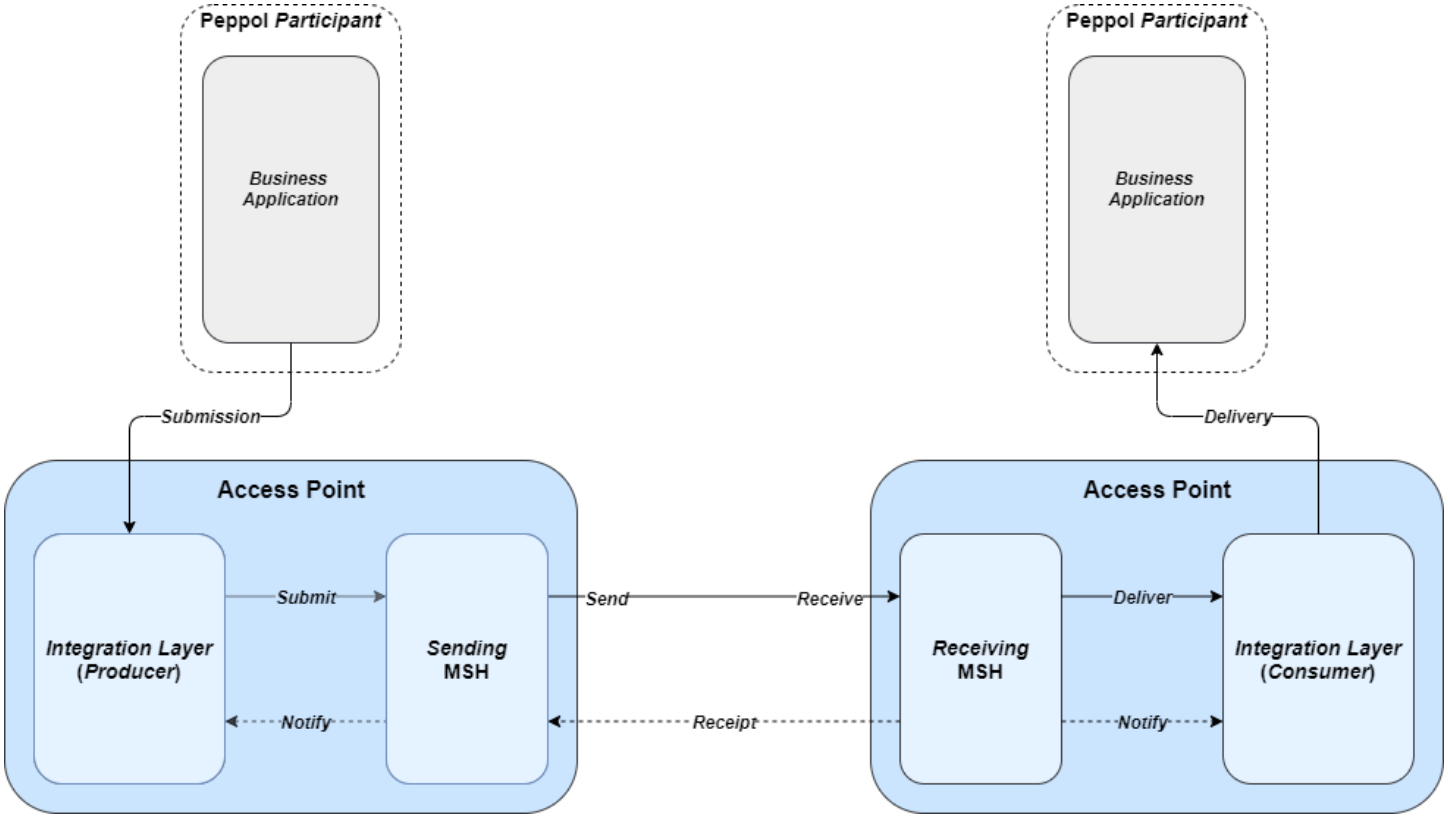


Figura 2.19: Perfil Peppol AS4 - Âmbito considerado

determinado contexto surjam perfis mais restritos, que detalham minuciosamente como é que a troca de mensagens deve ser realizada.

Assim, surgiu o perfil AS4 direcionado para o Peppol que deriva do perfil CEF *e-Delivery* 1.14[35] e abstrai o modelo anterior, considerando os *Access Points* como um único componente (fig. 2.19).

2.5.2 Peppol *Transport Infrastructure Agreements*[36]

A integridade e segurança das transações comerciais na rede Peppol dependem da cooperação entre os vários intervenientes e de uma infraestrutura de chave pública. Para administrar e governar os interesses do Peppol, foram designadas as seguintes autoridades:

- **Peppol *Coordinating Authority*:** tem autoridade sobre todos os componentes centrais da rede PEPPOL (as especificações técnicas e de serviços, o *Service Metadata Locator*, os *Transport Infrastrucure Agreements* e respetivos anexos) e delega autoridade sobre possíveis implementações, assim como a utilização da infraestrutura dentro de um domínio definido a uma Peppol *Authority*.

[35] ver [12].
[36] Secção estruturada com base em [59, 25]

- **Peppol *Authorities*:** garantem que os *Access Points* e *Service Metadata Publishers* disponibilizados estão em conformidade com os padrões técnicos e especificações do serviço que comercializam, celebrando acordos separados com ambos nos seus domínios.

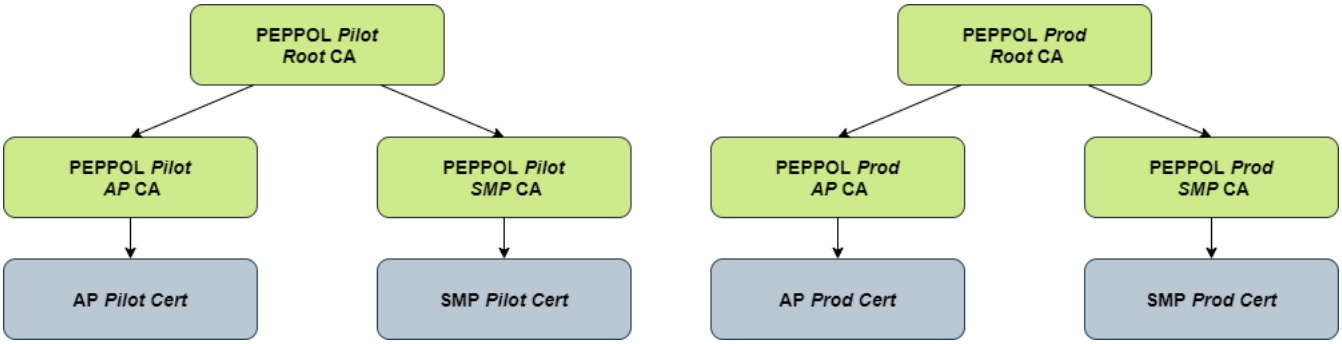


Figura 2.20: Peppol - *PKI version* 3.0

Após assinar os *Transport Infrastructure Agreements*, os *Access Points* e *Service Metadata Publishers* recebem um certificado Peppol direcionado para o ambiente de testes da rede. Para que uma determinada implementação seja comercializada é necessário realizar o OpenPEPPOL *testbed*, cujo objetivo é fazer o *onboarding* de novos provedores de serviços e garantir que as suas implementações estão de acordo com os *Transport Infrastructure Agreements* assinados [26, 37]. Caso os resultados do teste de aceitação sejam positivos, a implementação será reconhecida pela sua legitimidade e será emitido um novo certificado que lhe permitirá entrar em produção (a fig. 2.20 realça esta separação hierárquica).

Os certificados são válidos até que os *Transport Infrastructure Agreements* assinados expirem. Mas, podem ser revogados antes do tempo se os provedores de serviços quebrarem algum acordo.

Utilização de certificados no PEPPOL:

- ***Service Metadata Publisher***
  - Utiliza o certificado `PEPPOL SMP` para assinar os `SignedServiceMetadata`[37];
  - Utiliza o certificado `PEPPOL SMP` quando comunica com o *Service Metadata Locator* para criar ou apagar um `ServiceGroup`;
  - Requer a parte pública dos certificados `AP PEPPOL` que constam nos seus *endpoints* registados.
- ***Receiver Access Point***
  - Verifica se o documento que recebeu está assinado por um certificado `PEPPOL AP` válido;

[37] Modelo de dados SMP (ver secção 4 em [45]).

  - Requer outro certificado para TLS. Este não precisa de ser emitido pela Peppol PKI, mas não pode ser *self-signed*.

- ***Sender Access Point***
  - Utiliza o certificado `PEPPOL AP` para assinar as mensagens AS4 e disponibiliza a sua parte pública nas mensagens que transmite;
  - Também requer um certificado para o protocolo TLS.

2.5.3 Peppol *Business Interoperability Specifications*[38]

O Peppol desenvolveu as *Business Interoperability Specifications* para normalizar a troca e validação de documentos eletrónicos entre *Access Points*, consumidores do setor público e respetivos fornecedores.

As especificações contemplam os processos comerciais mais comuns entre indústrias e asseguram a cobertura de todos os requisitos legais e comerciais dos mesmos, o que permite a um participante emitir documentos, num contexto transfronteiriço, de modo simples. Atualmente, as Peppol BIS que existem são:

- PEPPOL BIS *Billing* 3.0
- PEPPOL BIS *Order Only* 3.1
- PEPPOL BIS *Ordering* 3.1
- PEPPOL BIS *Catalogue only* 3.0
- PEPPOL BIS *Catalogue without response* 3.0
- PEPPOL BIS *Despatch Advice* 3.0
- PEPPOL BIS *Punch Out* 3.0
- PEPPOL BIS *Order Agreement* 3.0
- PEPPOL BIS *Message Level Response* 3.0
- PEPPOL BIS *Invoice Response* 3.0

Concretamente, para o âmbito da tese será utilizada a Peppol BIS *Billing* 3.0 que suporta o cenário de *e-Invoicing*. Porém, as BIS poderão não arcar de raíz dados específicos para determinadas indústrias e certos contextos fora do domínio do *e-Procurement*. Nestes casos, os parceiros comerciais podem dentro do seu domínio estabelecer conteúdos e regras adicionais que lhes permitam satisfazer as suas necessidades, utilizando um BIS já definido como base para a personalização. Geralmente, este processo desencadeia a designação de um novo `CustomizationID` para diferenciar a personalização da especificação base e os seus contextos de transação.

2.5.3.1 Validação de documentos

Um dos princípios que rege o Peppol é "*Only valid documents are to be exchanged over the network*". Isto implica, que todos os documentos trocados devem ser confrontados com os

[38] Secção estruturada com base em [59, 26, 42].

artefactos de validação disponibilizados pela sua BIS.

O *Access Point* utilizado pelo participante que submete o documento é o ponto de entrada para a rede. Consequentemente, recai sobre este a responsabilidade de garantir que não está a trocar mensagens falaciosas com os restantes *Access Points* e participantes. No entanto, é recomendado que todos os provedores de serviço de *Access Points* monitorizem o comportamento dos seus participantes, assim como de todos os outros *Access Points* com quem estabelecem comunicações, para assegurar que estão a operar em conformidade no seu domínio comercial. Se porventura existir algum incumprimento por parte dos participantes, os *Receiver Access Points* e *Sender Access Points* deverão abordá-los para resolver o problema.

Caso os emissores continuem a transmitir documentos inválidos com o consentimento do *Access Point* que têm acordo, o *Receiver Access Point* deverá elaborar um relatório e reportá-lo à PEPPOL *Authority* com a qual assinou os TIAs. A denúncia é de extrema importância e permite à Peppol *Authority* intervir, revogando possivelmente o certificado Peppol do *Sender Access Point*. Para terminar e considerando o contexto de *e-Invoicing*, concerne:

- Ao emissor submeter faturas corretas para que o pagamento seja efetuado;
- Ao *Sender Access Point* garantir que as faturas emitidas estão corretas, para que possa continuar a disponibilizar os seus serviços;
- E ao *Receiver Acess Point* garantir que os participantes que serve não estão a receber documentos inválidos.

3

Solução Proposta

Pretende-se desenvolver uma prova de conceito que demonstre a submissão, validação e certificação de elementos de faturas em tempo real integrado no produto SAFTPRO[1] e em simultâneo possibilitar a configuração dos requisitos legais das faturas, para atender às especificidades de cada país, sem ter de reconstruir o produto de raíz.

A prova de conceito assenta sobre os fundamentos disponibilizados pelo Peppol, porque foi ponderada a importância de *Open-standards* e a normalização do processo de *e-Invoicing*. O Peppol é promissor porque começou como um projeto desenhado para operar em solo Europeu, mas transborda para outros países como a Austrália, o Canadá, a Singapura, a Nova Zelândia e os Estados Unidos da América.

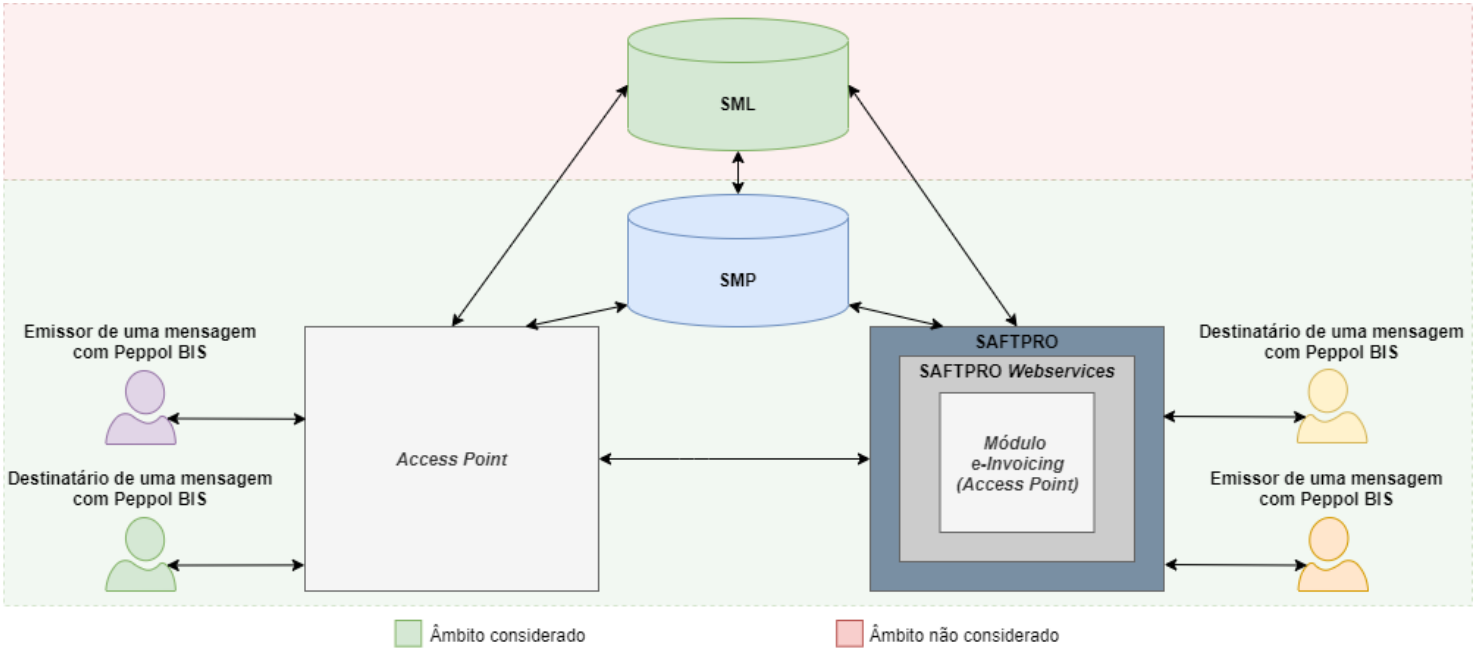


Figura 3.1: Prova de Conceito - Âmbito considerado

[1] Consultar o anexo VII para mais detalhes sobre a visão geral do produto.

A figura 3.1, delimita os componentes ponderados na conceção da solução, à exceção do SML. O SMP também não foi desenvolvido, mas foi necessário recorrer à configuração de uma implementação *Open-source*, para permitir o registo de participantes, sem um certificado Peppol válido. Em paralelo, a ausência deste certificado também impossibilitou a integração do *Service Metada Publisher* com o *Service Metadata Locator*, nos seus diferentes domínios.

3.1 Componentes da Solução

O foco de desenvolvimento é o módulo de *e-Invoicing*, onde residem as funcionalidades que permitem ao SAFTPRO operar como *Access Point* e concedem a caracterização das faturas. Ainda assim, para demonstrar um exemplo de integração das APIs com os utilizadores finais foi contemplada uma pequena aplicação *desktop*.

3.1.1 Módulo de *e-Invoicing*

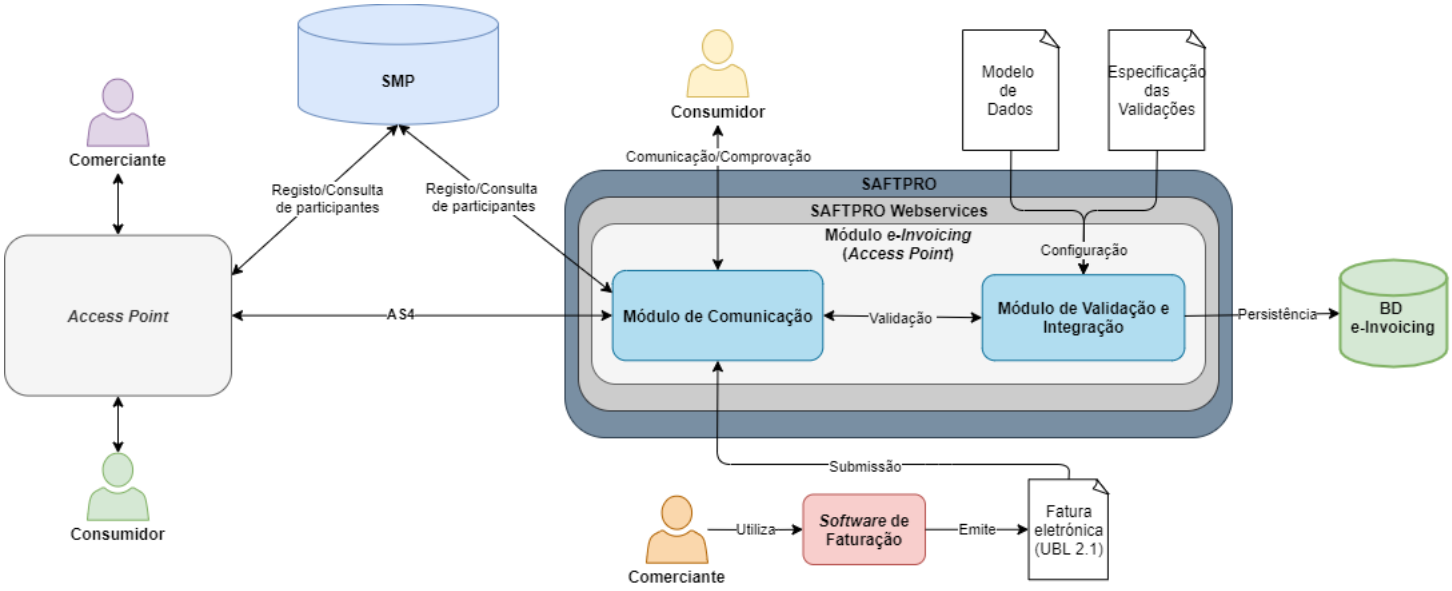


Figura 3.2: SAFTPRO - Conceito

A solução concebida consiste na construção de dois sub-módulos:

- **Módulo de Comunicação**: responsável pelos protocolos de comunicação que suportam o modelo de *e-Invoicing* e por disponibilizar as funcionalidades que facilitam o processamento da informação de modo a possibilitar a automatização e integração da emissão, envio e receção de faturas eletrónicas.
  Após a validação de uma fatura, recai também sobre este módulo a responsabilidade de a comunicar para apurar a veracidade dos seus detalhes. Este fluxo termina

normalmente com a persistência dos dados da fatura na base de dados, ou com a sua rejeição.

- **Módulo de Validação e Integração**: responsável por incorporar o modelo de dados e as validações que devem ser efetuadas sobre as faturas.
  Para o modelo de dados a solução idealizada consiste em definir as entidades e relações de acordo com a norma europeia EN16931, mas com ênfase no mapeamento para o formato UBL 2.1, efetuado pela Peppol BIS *Billing* 3.0. A partir desta noção de modelo base e com o apoio de uma DSL, torna-se possível a geração de código para extrair a informação das faturas e fazer a sua persistência.
  O processo de validação segue o modelo definido pela UBL 2.1, descrito na secção 2.3.4. A validação sintática visa garantir que as faturas eletrónicas recebidas atendem aos requisitos técnicos em vigor para o XSD do UBL 2.1. Em oposição, a validação semântica prende-se mais com o conteúdo do documento e o objetivo é garantir que as faturas respeitam as especificações legais que cada país estabelece. Esta divisão permite-nos trabalhar o nível semântico genericamente e gerar CIUS que derivem da Peppol BIS *Billing* 3.0, sem comprometer a integridade da solução.

3.1.2 Protótipo aplicação *desktop*

Esta secção demonstra o esboço, concebido na ferramenta Balsamiq *Wireframes*[2], direcionado para a aplicação *desktop* disponibilizada aos utilizadores finais como exemplo de integração.

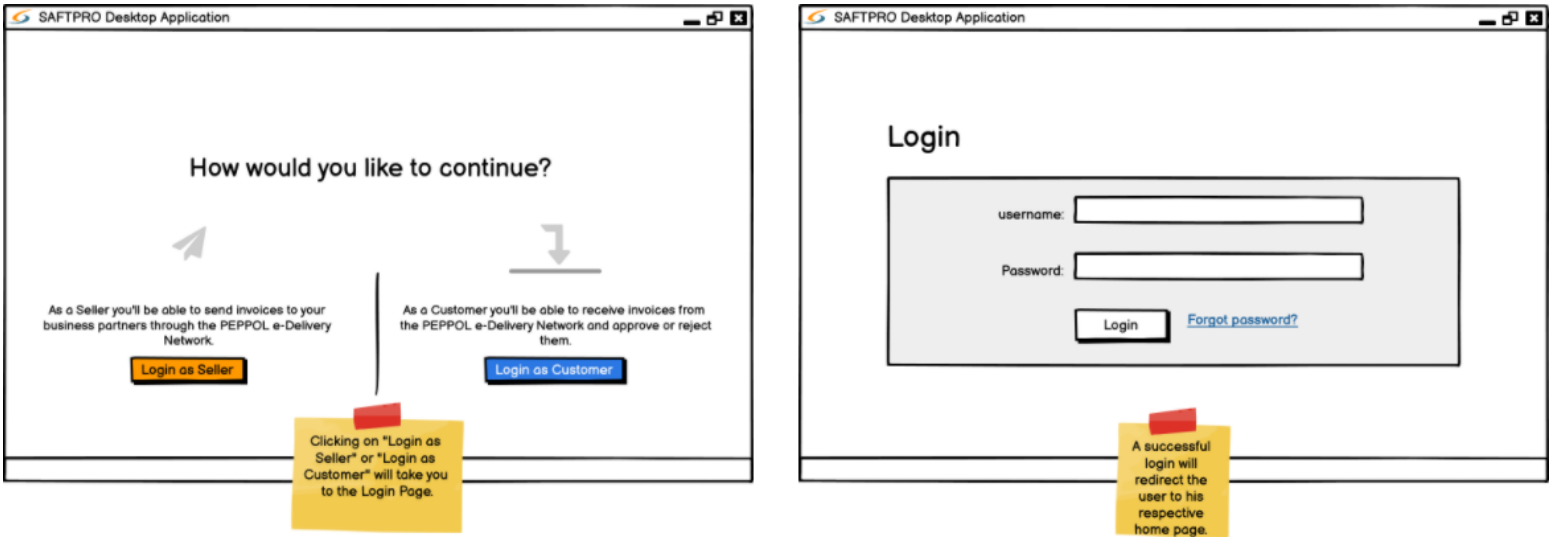


Figura 3.3: Página inicial e *Login*

[2] https://balsamiq.com/wireframes/

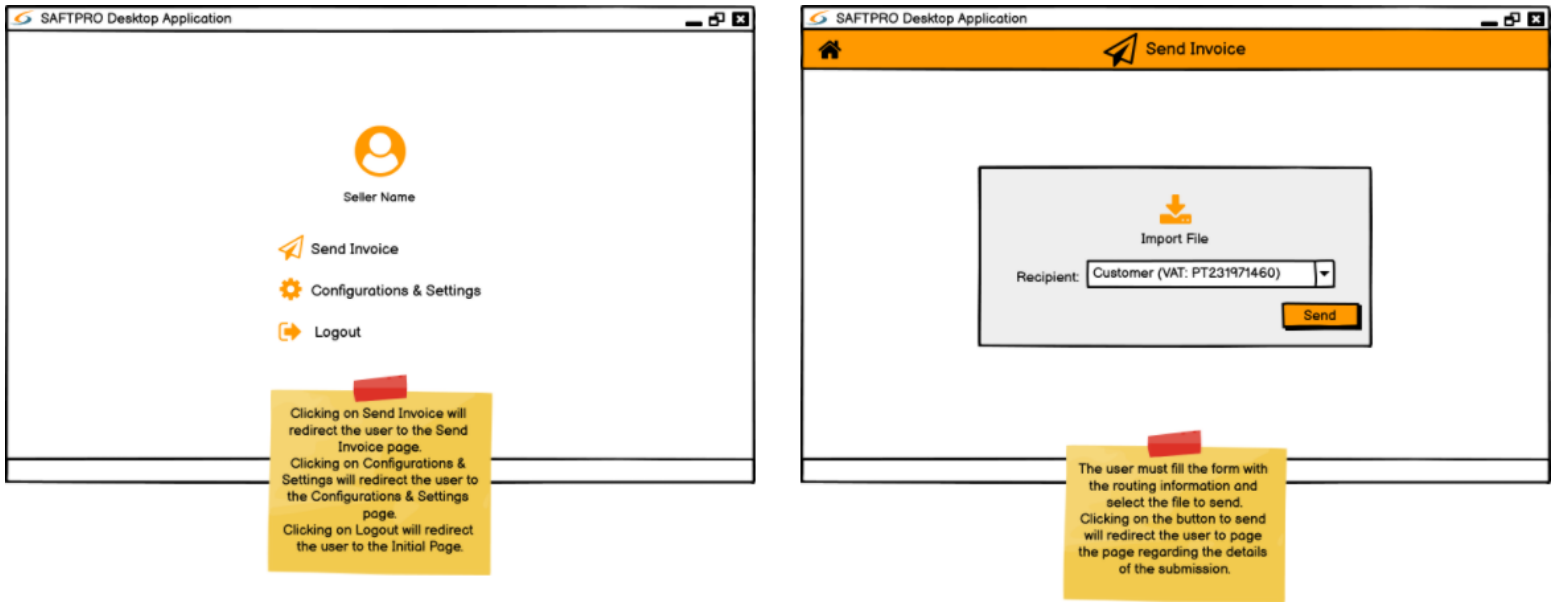


Figura 3.4: *Home Page* do Comerciante e formulário de submissão de faturas

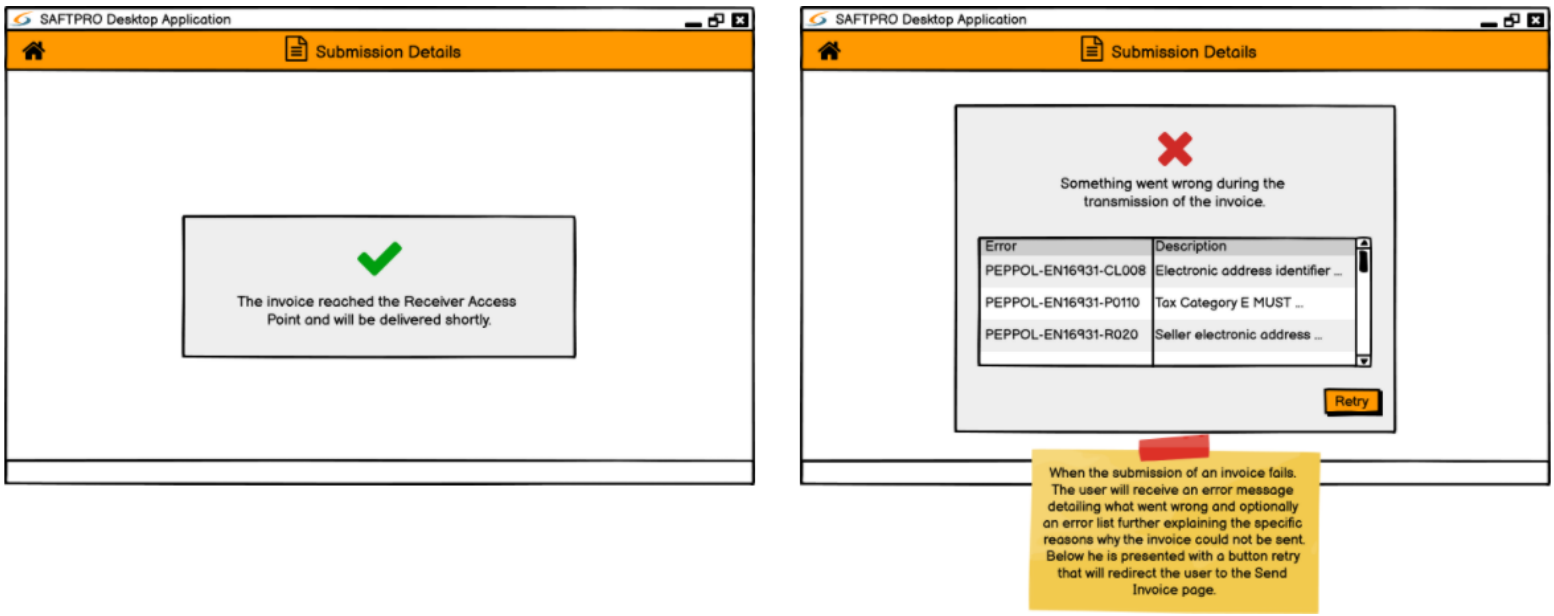


Figura 3.5: Detalhes de submissão

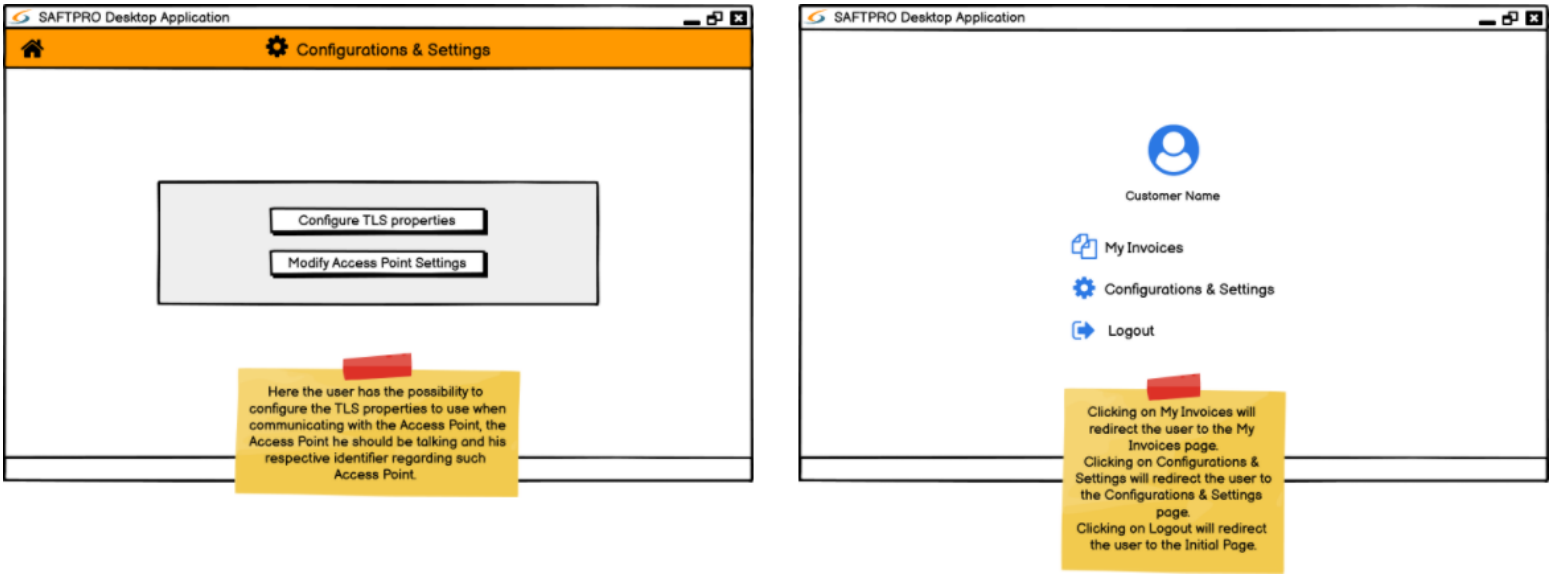


Figura 3.6: Página para configurações do Comerciante e *Home Page* do Consumidor

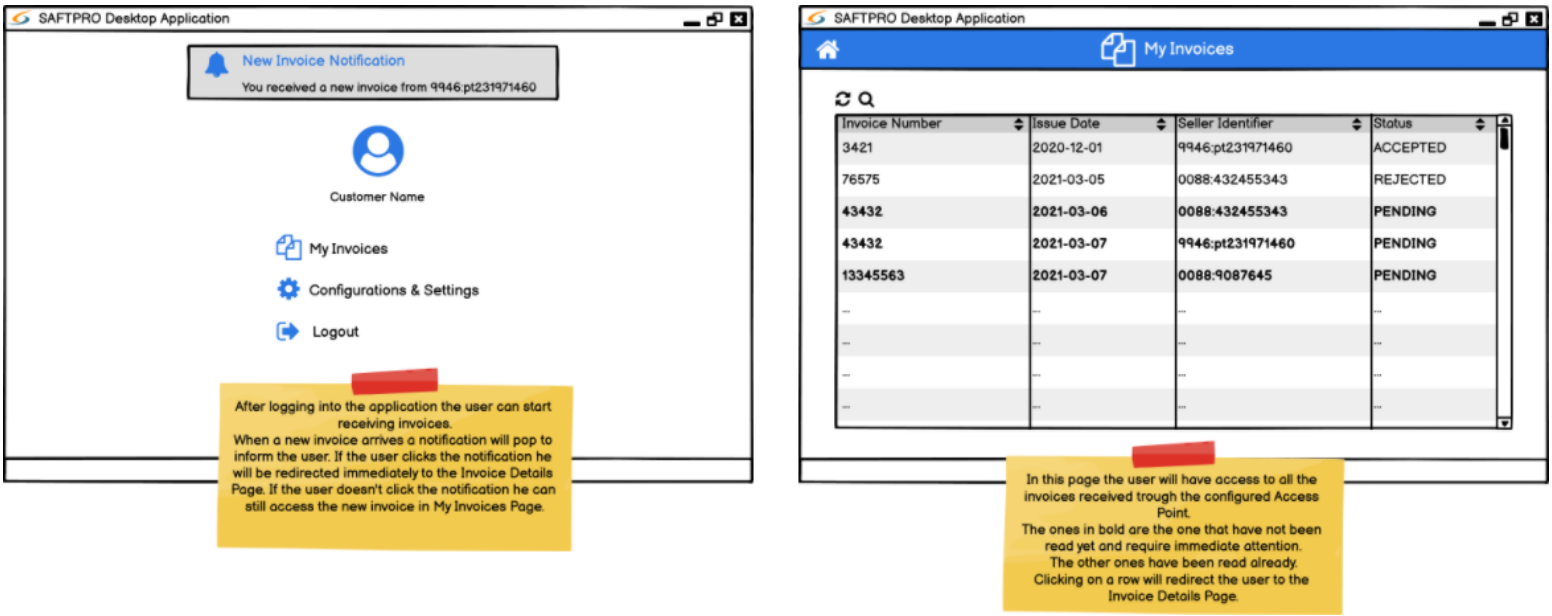


Figura 3.7: Notificação de uma nova fatura e página para consulta das faturas recebidas

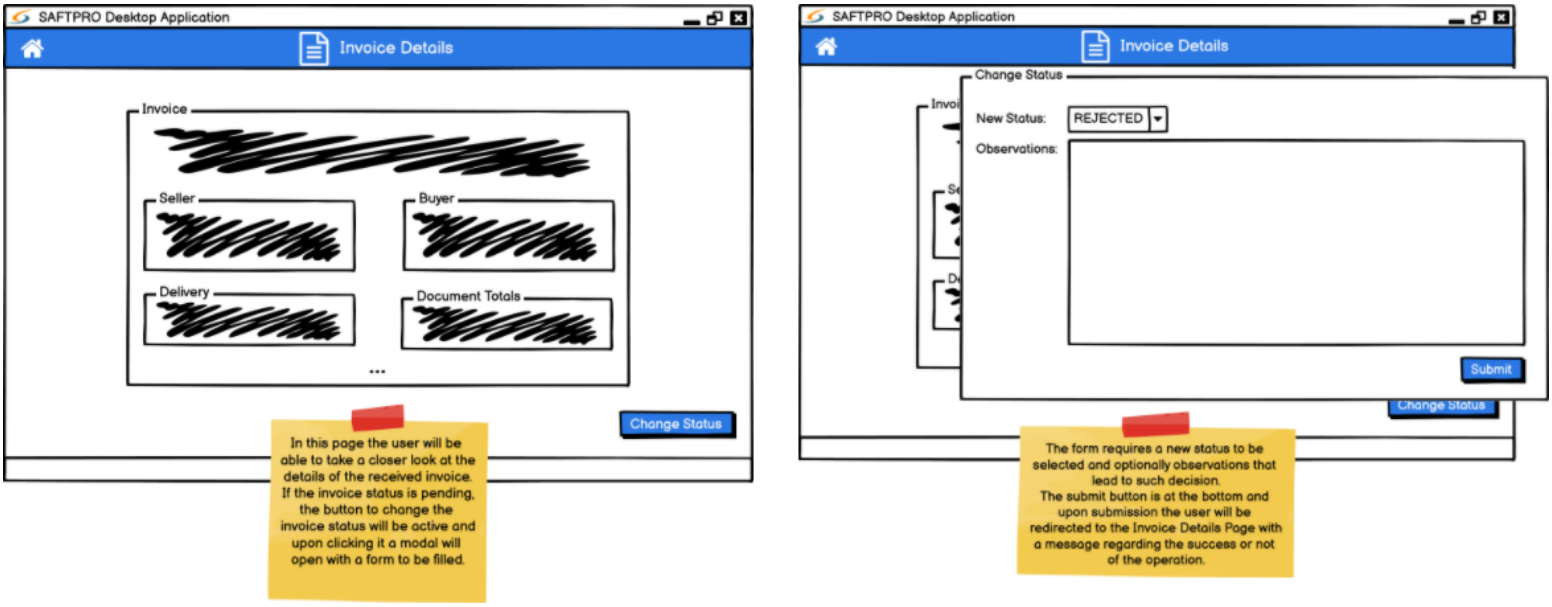


Figura 3.8: Página para consulta dos detalhes de uma fatura e aprovação

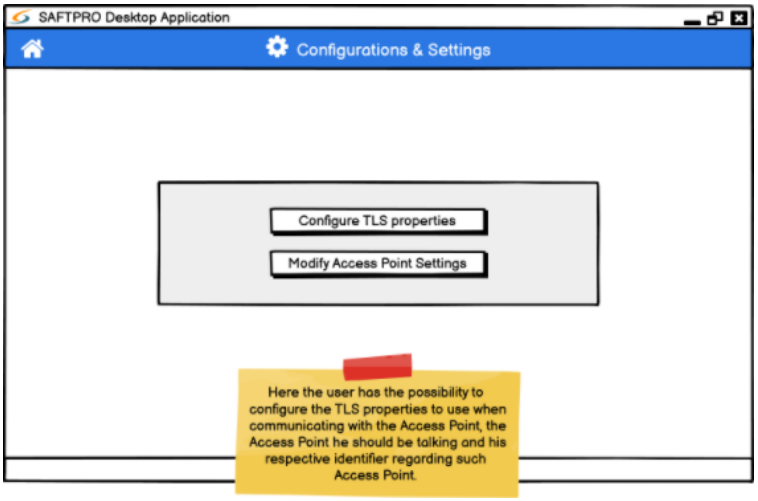


Figura 3.9: Página para configurações do Consumidor

3.2 Modelo de Dados

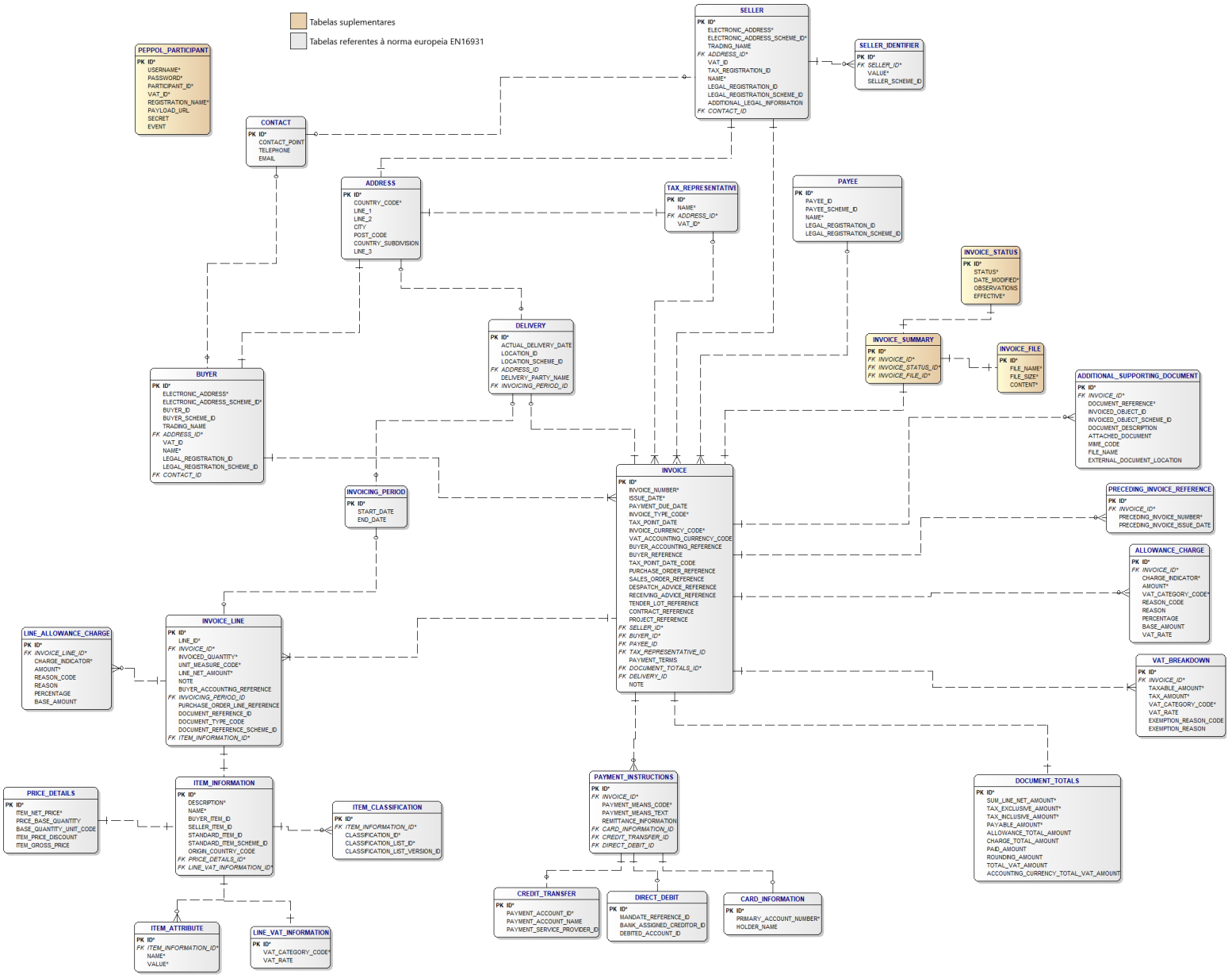


Figura 3.10: Módulo *e-Invoicing* - Modelo de dados

A figura 3.10 representa o modelo de dados, que deriva do mapeamento definido pela Peppol BIS *Billing* 3.0 para a norma EN16931. Neste modelo, existem existem quatro campos que não foram considerados.

O campo *Invoice note subject code* não está coberto explicitamente no Peppol, visto que é opcional e não transporta informação vital. Os campos `PROCESS CONTROL`, *Business process type* e *Specification identifier* estão mapeados, mas o seu valor é fixo e portanto basta garantir que estão corretos na validação.

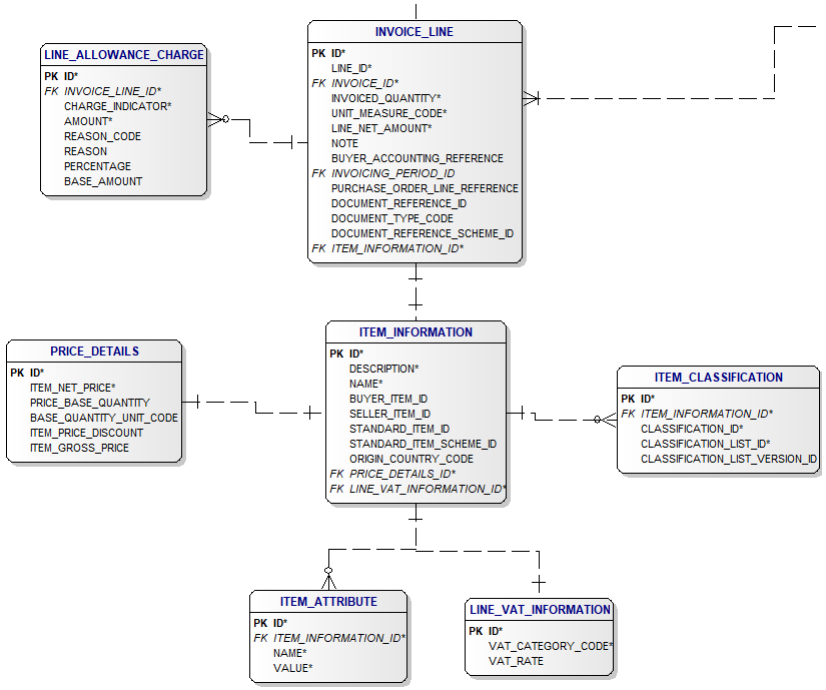


Figura 3.11: Módulo *e-Invoicing* - Modelo de dados ampliado (1/4)

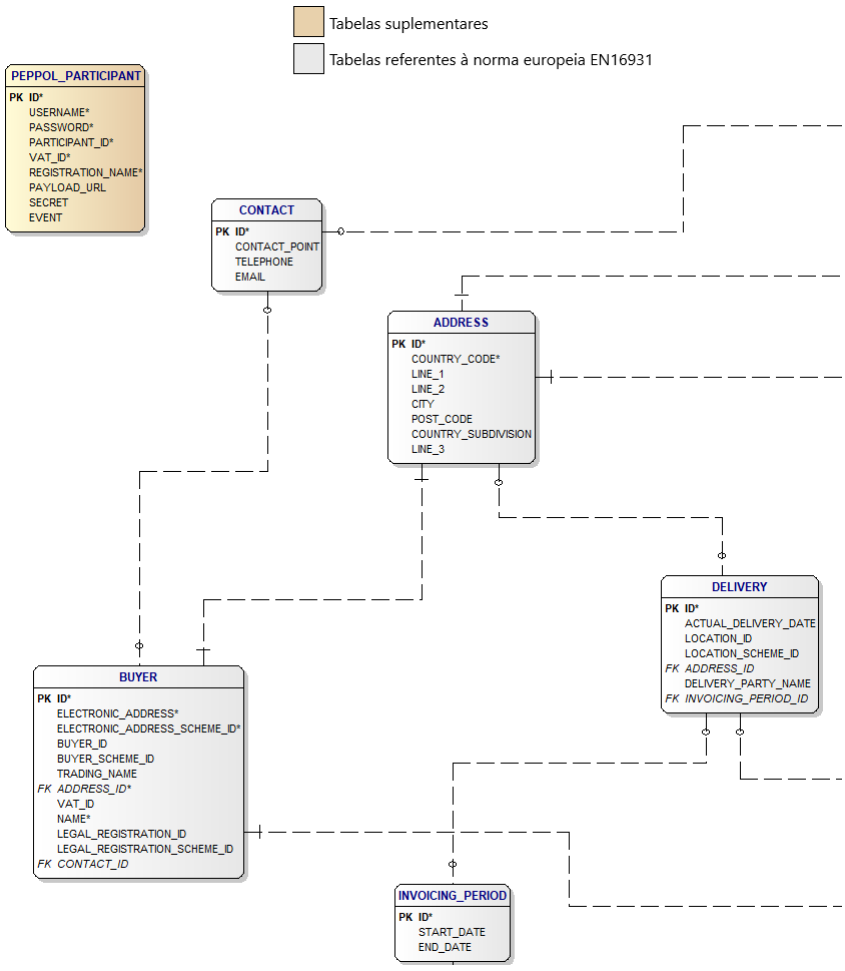


Figura 3.12: Módulo *e-Invoicing* - Modelo de dados ampliado (2/4)

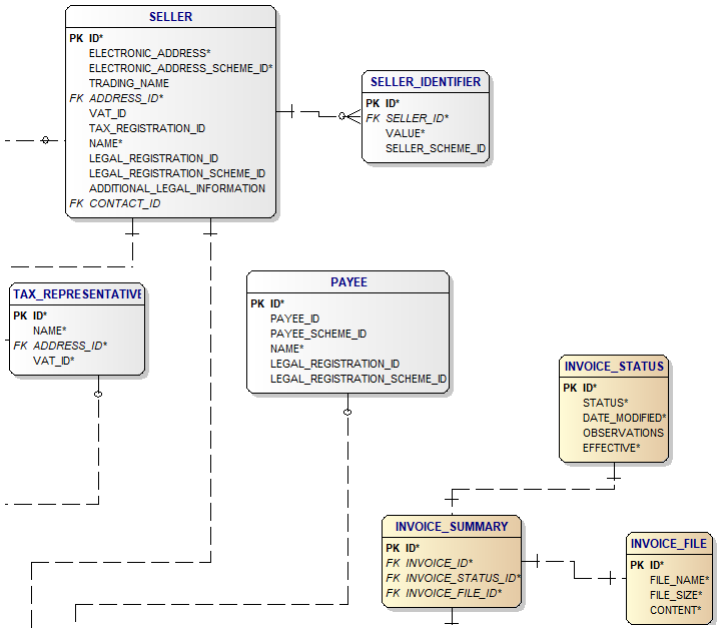


Figura 3.13: Módulo *e-Invoicing* - Modelo de dados ampliado(3/4)

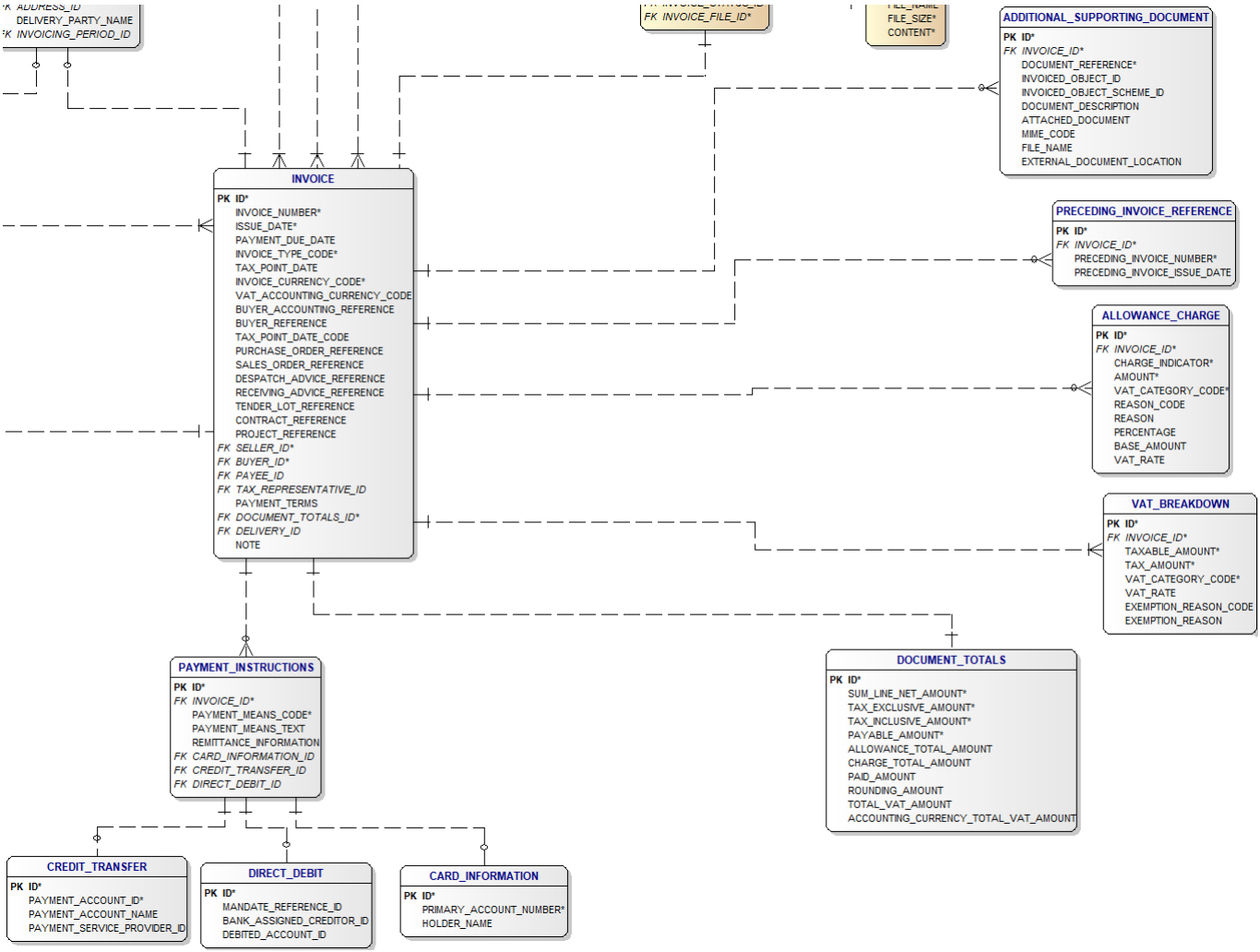


Figura 3.14: Módulo *e-Invoicing* - Modelo de dados ampliado(4/4)

3.3 Requisitos

As necessidades que não são satisfeitas na solução atual e constituem os objetivos a alcançar na fase de desenvolvimento estão descritos na secção 1.3 e foram agrupados num conjunto de requisitos funcionais e não funcionais.

Os requisitos funcionais descrevem explicitamente as funcionalidades do sistema e estão listados na figura 3.15.

ID Need	ID RQ	Descrição
ND01	**RQ01**	O sistema deverá ser capaz de receber, processar e analisar faturas emitidas sob o formato UBL 2.1. Podemos assumir que as faturas podem ser mapeadas integralmente em memória.
	RQ02	O sistema deverá conseguir receber e processar configurações de validações para o formato UBL 2.1 através de ficheiros XSD. Estes ficheiros de configuração também cabem integralmente em memória.
	RQ03	O sistema deve ser dotado com a capacidade para gerar código dinamicamente com base nas validações descritas no ficheiros de configuração.
	RQ04	O sistema deverá ser capaz de efetuar validações sobre as faturas, no momento em que as transações estão a ocorrer, e comunicar ao emissor quais quer inconsistências encontradas.
ND02	**RQ05**	O sistema deverá estar preparado para comunicar os detalhes da fatura ao consumidor para efeitos de confirmação.
	RQ06	O sistema deverá ser capaz de receber a confirmação do consumidor e despoletar a notificação da aprovação da fatura aos contribuintes.
	RQ07	O sistema deverá ser capaz de persistir a informação difundida e disponibilizá-la no portal do **SAFTPRO.**
ND03	**RQ08**	O sistema deverá disponibilizar canais seguros para a submissão das faturas pelos comerciantes de bens e/ou serviços.
	RQ09	O sistema deverá fornecer APIs que facilitem a adoção do sistema pelos produtores de software de faturação ou POS.
ND04	**RQ010**	O sistema deverá poder certificar a emissão de faturas com base nas assinaturas digitais que as acompanham.
	RQ011	O sistema deve estar dotado para autorizar a emissão de faturas por software desenvolvido por terceiros.

Figura 3.15: SAFTPRO - Requisitos Funcionais

Os Requisitos não funcionais, podem ser categorizados e têm como objetivo descrever as propriedades e restrições do sistema.

A primeira categoria identificada foi a de Desempenho (fig. 3.16), pela sua enorme responsabilidade na solução final, uma vez que todo o ciclo de vida da fatura depende da correta implementação do sistema e é necessário garantir que os processos despoletados

são o mais eficientes possível.

ID Need	ID RQ	Descrição
ND01	**RP1**	O processamento e validação das faturas deverá ser feito em tempo real para permitir detetar qualquer incumprimento de maneira transparente e responsabilizar imediatamente os envolvidos no ato.
	RP2	Qualquer informação que seja requisitada de fontes externas para validações semânticas deve ser o mais eficiente possível, para minimizar o impacto no processamento da fatura.
	RP3	O tamanho da fatura não deverá condicionar a eficiência do seu processamento e validação.
ND02	**RP4**	A comunicação entre o sistema e o consumidor deverá ter baixa latência, para não degradar a performance do processo de aprovação.
ND03	**RP5**	Os canais disponibilizados deverão permitir transmitir faturas e trocar qualquer outra informação de uma forma eficiente, para não congestionar o sistema.
ND04	**RP6**	A verificação das assinaturas digitais e integridade das faturas não devem ser os fatores dominantes no seu fluxo de processamento.

Figura 3.16: SAFTPRO - Requisitos de Desempenho

Não obstante, existem outros requisitos não funcionais como os de Segurança, Documentação e Manutenção, que poderão ser consultados na secção VII.4 do anexo VII.

3.4 Tecnologias a utilizar

Esta secção introduz as principais tecnologias a utilizar para desenvolver a solução:

- **Spring**[3]**:** o SAFTPRO encontra-se dividido intrinsecamente entre uma *Web Application* (SAFTPRO Portais) e um REST-*based Webservice* (SAFTPRO *Webservices*), construídos com o apoio da *framework* Spring, que disponibiliza uma infraestrutura de suporte para o desenvolvimento de aplicações Java. O módulo de *e-Invoicing* incide sobre o projeto SAFTPRO *Webservices* e naturalmente as novas funcionalidades terão suporte desta ferramenta.
  Em relação ao Spring, podemos dizer que melhora o rendimento no desenvolvimento de uma aplicação, porque disponibiliza código reutilizável, introduz abstrações que omitem a complexidade de determinados processos de configuração, suporta *Test-driven development* e é eficiente na utilização de recursos do sistema.
  Em síntese, o Spring é construído em cima da ideia de injeção de dependências e inversão de controlo. Isto é, existe o conceito de *Bean*, que corresponde a um objeto que pode ser instanciado. Cada *Bean* declara as suas dependências (que

[3] https://spring.io/

serviços necessita para funcionar) e o Spring *Container*, fica responsável por resolvê-las automaticamente ligando os *Beans* entre si e coordenando os seus ciclos de vida.

- **phase4**[4]**:** biblioteca concebida para integrar em sistemas já existentes, que permite o envio e receção de mensagens AS4. Suporta o perfil AS4 definido pelo Peppol e tem utilizações comerciais.
- **PHIVE**[5]**:** mecanismo genérico para validação de documentos comerciais. Foi desenvolvido inicialmente para o Peppol, pelo que suporta o mecanismo de validação definido pelo UBL 2.1. Ou seja, envolve diferentes ficheiros XSDs e Schematron, que são ordenados num `ValidationExecutorSet` e aplicados ao documento a validar.
- **ph-schematron**[6]**:** biblioteca para validar documentos XML via Schematron. Para além disso, disponibiliza um *plugin* Maven que permite pré-processar ficheiros Schematron, através de *scripts* XSLT. Deste modo, o ficheiro torna-se ele próprio um *script* XSLT mais eficiente na validação de documentos. Contudo, este processo de pré-compilação é crítico e deve ser feito em *build-time*, visto que um ficheiro Schematron puro, de tamanho médio, demora minutos a processar. A grande vantagem é que o resultado da transformação pode ser armazenado e reutilizado para futuras validações.

  Neste sentido, todas as especificidades introduzias por um país serão traduzidas num ficheiro Schematron puro, que será processado por este *plugin* antes de ser incluído no `ValidationExecutorSet`, disponibilizado pelo PHIVE.
- **Xtend**[7] **e Xtext**[8]**:** o Eclipse Xtext é uma *framework* que permite criar rapidamente DSLs e inclui um editor integrado que oferece *code-completion*, *syntax-highlighting*, formatação e deteção de erros. Para além disso, oferece uma *workbench* que permite visualizar os modelos gerados pela DSL, facilita a sua navegação e disponibiliza mecanismos de *refactoring*.

  Os ficheiros Xtext após definidos são analisados por um *parser*, do qual resulta uma *Abstract Syntax Tree*, utilizada como base para a geração de código. É aqui que entra o Xtend, que é uma linguagem de programação *statically-typed* construída em Xtext e compilada para Java, que oferece oferece *extension methods*, *overloading operators*, *switches*, *polymorphic method invocations*, *template strings*, entre outros.

  Por exemplo, o Xtend permite definir *multi-line* strings com texto estático ou com partes dinâmicas, que podem ser interpretadas em *run-time* e substituídas pelos valores da entidade do modelo considerada naquele determinado instante.

  O Xtend também disponibiliza um editor extremamente útil, porque dá *highlight* dos espaços brancos resultantes no ficheiro final e em contraste com os outros *template engines* permite embutir diretamente nos templates as funções utilizadas

[4] https://github.com/phax/phase4
[5] https://github.com/phax/phive
[6] https://github.com/phax/ph-schematron
[7] https://www.eclipse.org/xtend/index.html
[8] https://www.eclipse.org/Xtext/index.html

para avaliar os valores dinamicamente.
Concluindo, a conjunção destas duas tecnologias formam uma base sólida para definir uma DSL, que captura o essencial do modelo de dados e permite enriquecê-lo, o que permite gerar código para suportar a infraestrutura que fará a extração e persistência dos dados de uma fatura submetida através do *módulo de e-Invoicing*.

- **Liferay**[9]**:** o Liferay fornece uma plataforma robusta para construir *Web Sites* e servi-los em qualquer dispositivo móvel ou *desktop*. Para além disso, oferece uma *framework* que permite customizar aplicações, ou criá-las de raíz através da noção de `Portlets`, que são componentes visuais independentes e servem informação dentro de uma página *Web*.
  A solução é *Open-source* e foi utilizada para implementar o projeto SAFTPRO Portais. Na implementação do módulo de *e-Invoicing*, é necessário complementar o *Backoffice* do SAFTPRO para possibilitar a consulta das faturas submetidas, pelo que será necessário desenvolver novos `Portlets`.
- **phoss SMP**[10]**:** servidor *Service Metdata Publisher* que suporta a especificação SMP do Peppol. É disponibilizado com uma interface gráfica e pode ser integrado com diversas bases de dados.
  Este projeto, será configurado para operar com os requisitos do Peppol e servirá de apoio à prova de conceito.
- **Flutter**[11]**:** selecionado para desenvolver a aplicação *desktop* direcionada para os utilizadores finais. É uma *toolkit* que permite desenvolver UIs modernas e portáveis. Atualmente, a sua versão *desktop* ainda está em *Beta*, o que pode condicionar a distribuição de versões para produção. Não obstante, surgiu a oportunidade de explorar a ferramenta para desenvolver uma pequena aplicação e o facto de possuir *stateful hot reload*[12] e utilizar a linguagem Dart[13], que é muito semelhante ao Java, foram pontos a favor na sua eleição.

[9] https://www.liferay.com/pt/home
[10] https://github.com/phax/phoss-smp
[11] https://flutter.dev/?gclsrc=aw.ds&gclid=CjOKCQjw4eaJBhDMARIsANhrQAAlOCgOtt2kV5qKmqlHzS9-aedGCvOoiyjjDjvAwcB
[12] https://flutter.dev/docs/development/tools/hot-reload
[13] https://dart.dev/

4

Implementação

Este capítulo descreve detalhes mais concretos de implementação e discute potenciais alternativas consideradas na fase de desenvolvimento.

4.1 SAFTPRO: Introdução

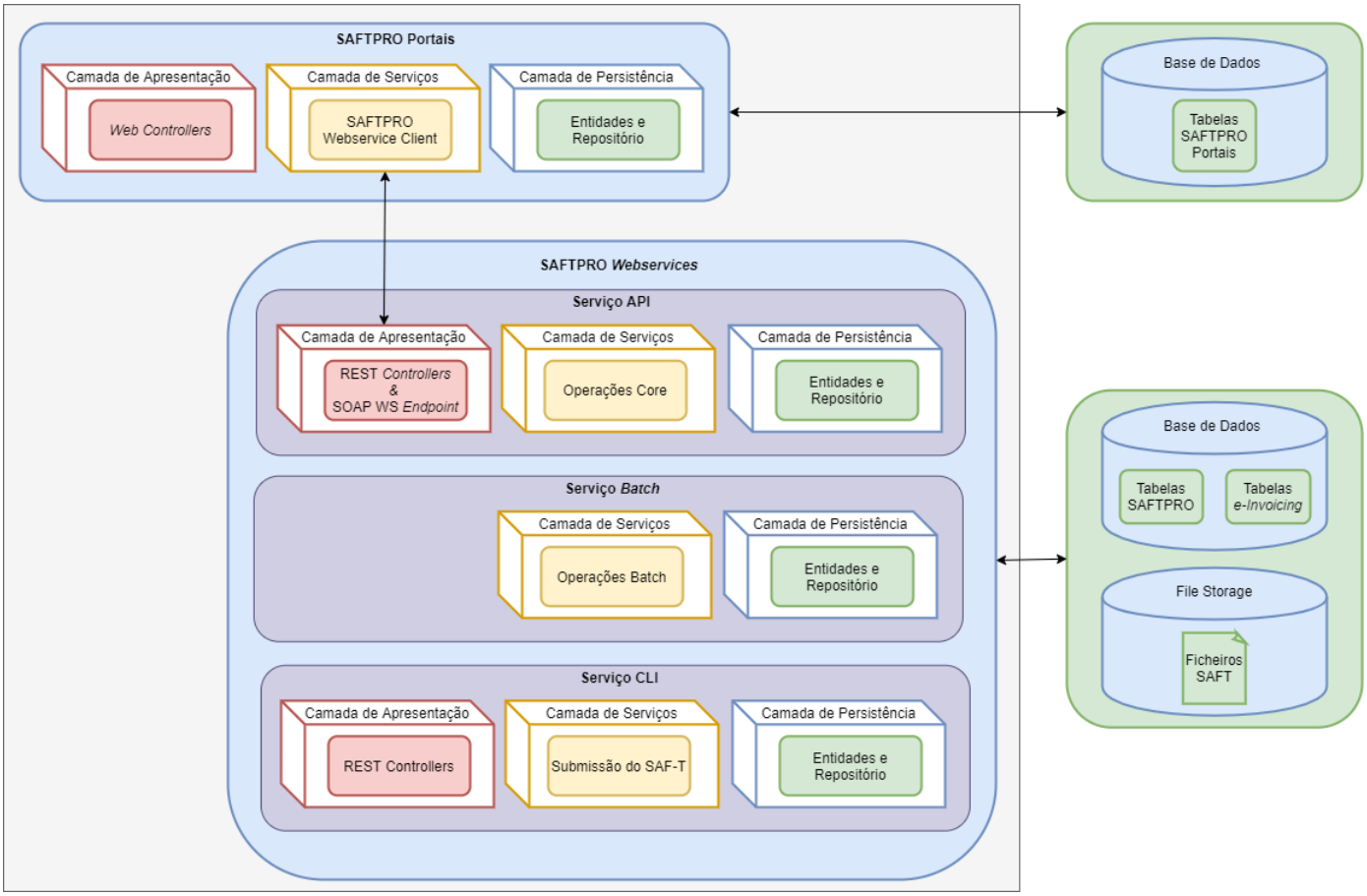


Figura 4.1: Diagrama de implementação

Grande parte da implementação reside no componente SAFTPRO *Webservices*, onde se situa o módulo de *e-Invoicing*. No entanto, o SAFTPRO Portais também sofreu alterações para disponibilizar uma página de consulta para os funcionários das Autoridades Tributárias.

O SAFTPRO Portais e o SAFTPRO *Webservices* implementam variantes de um modelo em camadas[1], com pequenas nuances entre si (o diagrama presente na fig. 4.1 reflete essa organização e a sua estrutura).

O SAFTPRO Portais segue o *workflow* tradicional da *framework* Spring MVC[2] (fig. 4.2), construída sobre a Servlet API.

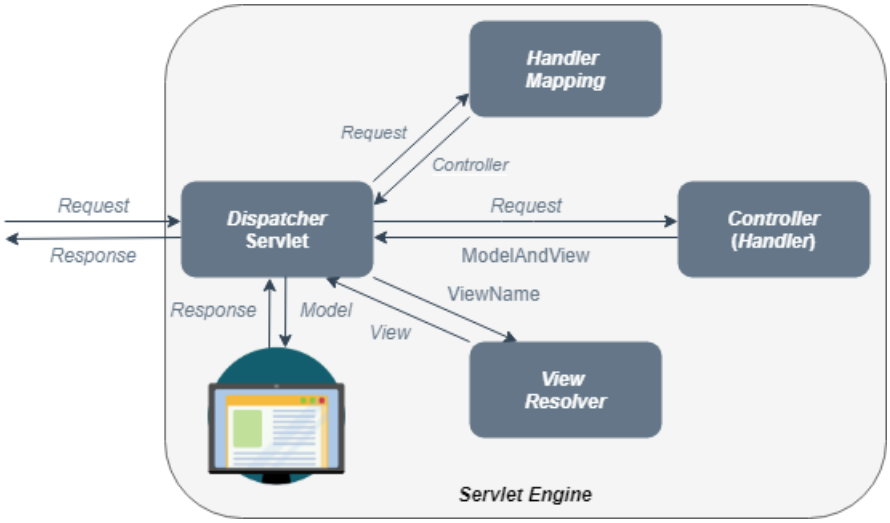


Figura 4.2: Spring MVC *workflow* tradicional

Segundo este, o fluxo da aplicação é controlado pelo `DispatcherServlet`, que atua como *Front Controller*[3], e resume-se no seguinte:

1. O `DispatcherServlet` interceta os pedidos HTTP;
2. Por intermédio do `HandlerMapping`, obtém-se o *Controller* responsável por tratar cada pedido;
3. Ao receber um pedido, o *Controller* executa a lógica de negócio e devolve ao `DispatcherServlet` o objeto `ModelAndView`, composto pela resposta contida num *Model* mais o nome da *View*, para a qual o utilizador deverá ser redirecionado;
4. Por último, o `DispatcherServlet` obtém a `ViewPage` junto do `ViewResolver` e preenche-a com os dados do *Model*, antes de a retornar ao utilizador.

Muito sucintamente, o Spring permite configurar o contexto aplicacional através de anotações. Para processar pedidos HTTP, existe a anotação `@Controller`, que permite

[1] Consultar fig. VII.4 em anexo para mais informação.

[2] https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html

[3] https://www.linkedin.com/pulse/what-front-controller-design-pattern-how-create-ioc-spring-jibhakate/

identificar as classes que exercem a função de *Controller*. Adicionalmente, existe a anotação `@RequestMapping` que designa os pedidos que pertencem a um determinado *Controller* e delimita que tipo de métodos HTTP (p.e., POST, GET e DELETE) são suportados para um determinado URL.

No SAFTPRO *Webservices* o *workflow* simplifica-se (fig. 4.3), porque a resposta HTTP não envolve a noção de *View*. Isto é, os dados são escritos diretamente para o corpo da resposta, tipicamente em JSON ou XML.

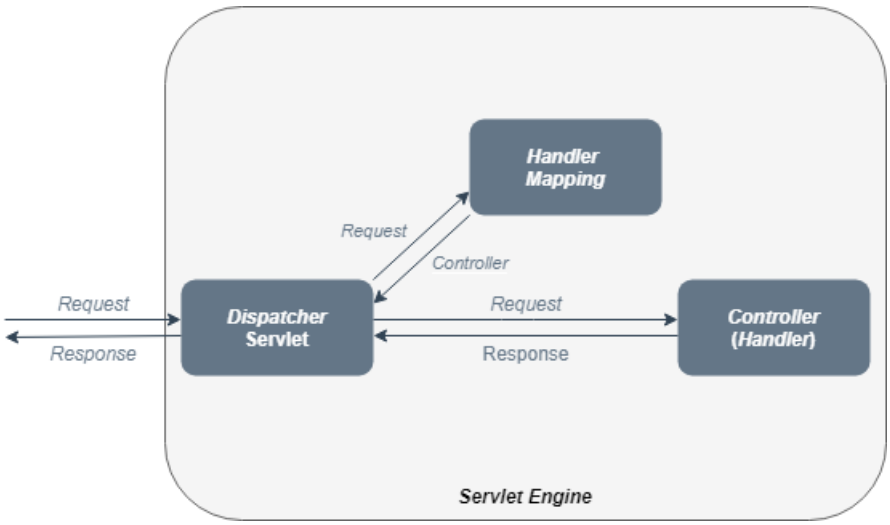


Figura 4.3: Spring MVC workflow de RESTful Webservices

O Spring é capaz de converter o valor retornado por um método e escreve-lo diretamente na *outputstream* da resposta HTTP, através da anotação `@ResponseBody` ao nível do método ou embutida na anotação `@RestController`. Isto é possível, porque o Spring possui no *background*, uma lista de `HttpMessageConverters`, cuja responsabilidade é converter o corpo do pedido HTTP para um objeto e converter um objeto para o corpo da resposta HTTP, com base no MIME definido para um `@RequestMapping`.

Camada de Persistência

Para a camada de acesso a dados, o Spring lançou o projeto Spring *Data*[4], dividido em vários componentes, que permite trabalhar com as mais recentes tecnologias como por exemplo, bases de dados não relacionais, *map-reduce*, etc. Mas, no SAFTPRO predominam as bases de dados relacionais suportadas pelo componente Spring *Data* JPA.

Muito sucintamente, o JPA[5] é a norma utilizada para efetuar o mapeamento de objetos Java para bases de dados relacionais e é importante referir que corresponde apenas a um

[4] https://spring.io/projects/spring-data

[5] https://download.oracle.com/otn-pub/jcp/persistence-2_2-mrel-spec/JavaPersistence.pdf

conjunto de especificações, das quais podem existir inúmeras implementações como o Hibernate[6], no caso do Spring.

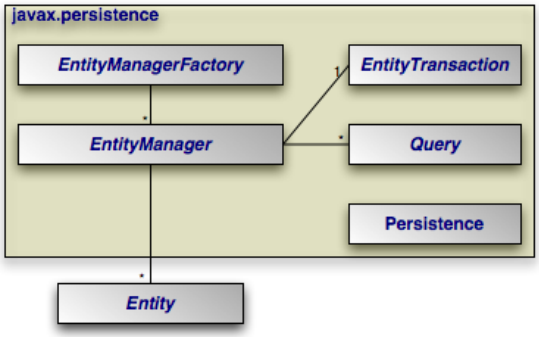


Figura 4.4: Java *Persistence* API - Arquitetura[7]

A figura 4.4 ilustra a arquitetura do JPA, que permite simplificar a complexidade de algumas linguagens de consulta e facilita o armazenamento de informação, sem ter de lidar diretamente com conexões à base de dados:

- **`EntityManagerFactory`:** classe que permite criar e gerir múltiplas instâncias de *EntityManager*;
- **`EntityManager`:** interface que gere um conjunto de *Entities* (objetos persistentes) e dispõe APIs para inserir novos elementos ou apagar os existentes;
- **`Entity`:** objetos que representam os *records* de uma base de dados;
- **`EntityTransaction`:** agrupa as operações em unidades de trabalho atómicas, para assegurar a integridade da base de dados subjacente.
- **`Query`:** interface que difere para cada implementação do JPA. Na sua essência, permite encontrar *entities* que satisfaçam um determinado critério de pesquisa e o JPA oferece especificações para consulta com JPQL[8], ou com SQL.

Apesar de simplificar bastante a camada de acesso a dados, os componentes acima têm de ser configurados corretamente para tirar proveito das funcionalidades do JPA. É neste cenário que o Spring *Data* JPA atua, simplificando a implementação dos repositórios JPA, ao ponto de apenas ser necessário declarar as interfaces para as *entities* que se pretende. Posteriormente, o Spring disponibiliza as suas implementações de modo automático.

Tomemos como exemplo a *Entity* `User` e o seu repositório, tal como exemplificado na listagem 10 e 11, respetivamente:

[6] https://hibernate.org/
[8] https://eclipse-ee4j.github.io/jakartaee-tutorial/#full-query-language-syntax

```java
package pt.opensoft.saftpro.persistence.einvoicing.entity;

import ...

@Entity
@Table(name = "user")
public class User {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID", nullable = false)
    private Long id;
    ...
    @Column(name = "EMAIL_ADDRESS")
    private Long emailAddress;
    ...
    @Column(name = "LAST_NAME")
    private String lastName;
    ...
    // GETTERS & SETTERS
    ...
}
```

Listagem 10: JPA - exemplo *User Entity*

No código acima, destacam-se as anotações:

- **`@Entity`:** especifica que a classe anotada representa objetos com contexto de persistência;
- **`@Id`:** identifica a chave-primária do modelo;
- **`@Table`:** especifica a tabela primária da *Entity* anotada;
- **`@Column`:** mapeia o atributo da classe com a coluna da tabela da base de dados.

```java
1  package pt.opensoft.saftpro.persistence.einvoicing.entity;
2
3  import ...
4
5  public interface UserRepository extends Repository<User, Long> {
6   ...
7   List<User> findByEmailAddressAndLastname(String emailAddress, String lastname);
8   ...
9   @Query("select u from User u where u.emailAddress = :emailAddress")
10  Optional<User> findByEmailAddress(String emailAddress);
11  ...
12 }
```

Listagem 11: JPA - exemplo *User Repository*

Relativamente aos repositórios e tendo como exemplo a listagem 11, o Spring Data JPA pode seguir a convenção, exemplificada pelo método `findByEmailAddressAndLastname()` (linha 7), onde a *query* utilizada para interrogar a base de dados é traduzida automaticamente e assemelha-se a "select u from User u where u.emailAddress = ?1 and u.lastname = ?2". Ou, a *query* pode ser refinada manualmente através da anotação `@Query` (linhas 9 e 10). Para além destas anotações e funcionalidades, existem outras que serão abordadas ao longo deste capítulo.

Mecanismos de segurança

Pelo facto do SAFTPRO *Webservices* ser um componente totalmente independente do SAFTPRO Portais, existe a necessidade de autenticar todos os pedidos efetuados à sua API. Ou seja, a API deverá validar o acesso de aplicações cliente e neste cenário o SAFTPRO Portais age como cliente.

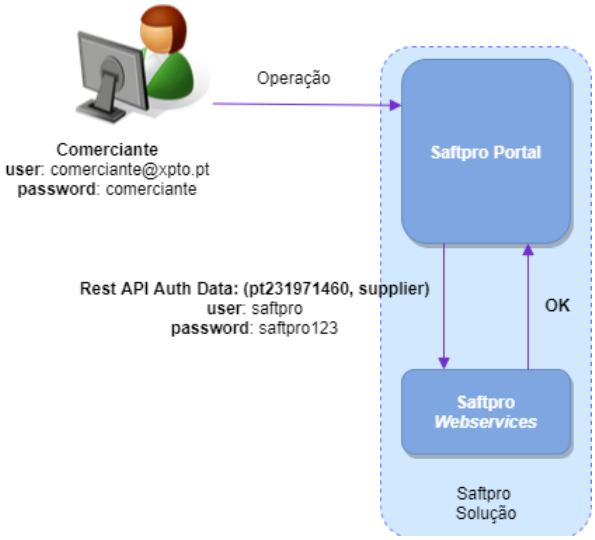


Figura 4.5: SAFTPRO - Autenticação *default*

Na solução atual, a API disponibilizada ao SAFTPRO Portais permite que este se autentique através de *Basic Authentication*, pois presume-se que ambos estão na mesma rede (fig. 4.5). Em simultâneo, as operações definidas na API, têm um conjunto de permissões de acordo com o nível de autorização, nomeadamente:

- ANONYMOUS;
- ROLE_BACKOFFICE;
- ROLE_CONSUMER;
- ROLE_SUPPLIER.

Nos *Controllers* Spring, estas autorizações podem ser expressas através da anotação `@RolesAllowed`, presente na definição de cada método.

Para aceder a uma determinada operação é obrigatório o envio da `API Auth Data`, composta pelo NIF (ou email) do utilizador autenticado no portal e o seu *Role*. Portanto, na realidade a autenticação e autorização de um utilizador é definida ao nível do SAFTPRO Portais, que depois se encarrega de injetar os dados de sessão nos pedidos que efetua ao SAFTPRO *Webservices*, o que permite fazer a sua validação ao nível da anotação `@RolesAllowed`. No módulo de *e-Invoicing*, existe uma parte da API onde não há necessariamente a noção de sessão e portanto a abordagem utilizada será detalhada em 4.2.1.1.

De seguida, exemplifica-se a interação dos dois componentes e realça-se as diferenças assinaladas acima:

```java
package pt.opensoft.saftproportais.backoffice.invoice.details.controller;

import ...

@Controller
@RequestMapping("VIEW")
public class InvoiceDetailsController {

    private InvoiceService invoiceService;
    ...
    @RenderMapping
    public String view(Model model, RenderRequest renderRequest, RenderResponse renderResponse) throws Exception {
        String id = HttpUtil.getOriginalServletRequest(renderRequest).getParameter("id");

        // Calling Saftpro Rest API
        final SaftproModel<InvoiceModel> invoice = invoiceService.getInvoice(id);

        // Setting Model View
        model.addAttribute("invoice", invoice.getData());

        // View
        return "backoffice-invoice-details";
    }
}
```

Listagem 12: SAFTPRO Portais - InvoiceDetailsController.java

```java
package pt.opensoft.saftpro.restcontroller.einvoicing;

import ...

@RestController
@RequestMapping("/rest")
public class InvoiceController {

    private InvoiceService invoiceService;
    ...
    @GetMapping(value = "/e-invoicing/invoices/{id}")
    @RolesAllowed(RoleUtil.BACKOFFICE)
    public ResponseEntity<SaftproModel<InvoiceModel>> getInvoices(@PathVariable Long id) {
        SaftproModel<InvoiceModel> result = new SaftproModel<>(invoiceService.getInvoice(id));

        return ResponseEntity.ok(result);
    }
}
```

Listagem 13: SAFTPRO *Webservices* - InvoiceController.java

4.2 SAFTPRO *Webservices*: Módulo de *e-Invoicing*

As funcionalidades desenvolvidas para o módulo de *e-Invoicing* situam-se no Serviço API do SAFTPRO *Webservices*.

A secção é um pouco extensa e divide-se entre os detalhes da implementação do módulo de comunicação, seguida pela descrição da infraestrutura de armazenamento e validação de documentos no módulo de validação e integração.

4.2.1 Módulo de Comunicação

O módulo de comunicação possui diferentes *handlers* (fig. 4.6), consoante o protocolo de comunicação subjacente. O `Phase4PeppolServlet` também é registado como um *Bean* na aplicação, porém a um nível de abstração inferior ao dos *Controllers*, devido à estratégia de integração adotada para a biblioteca phase4. No entanto, os seus propósitos são muito semelhantes.

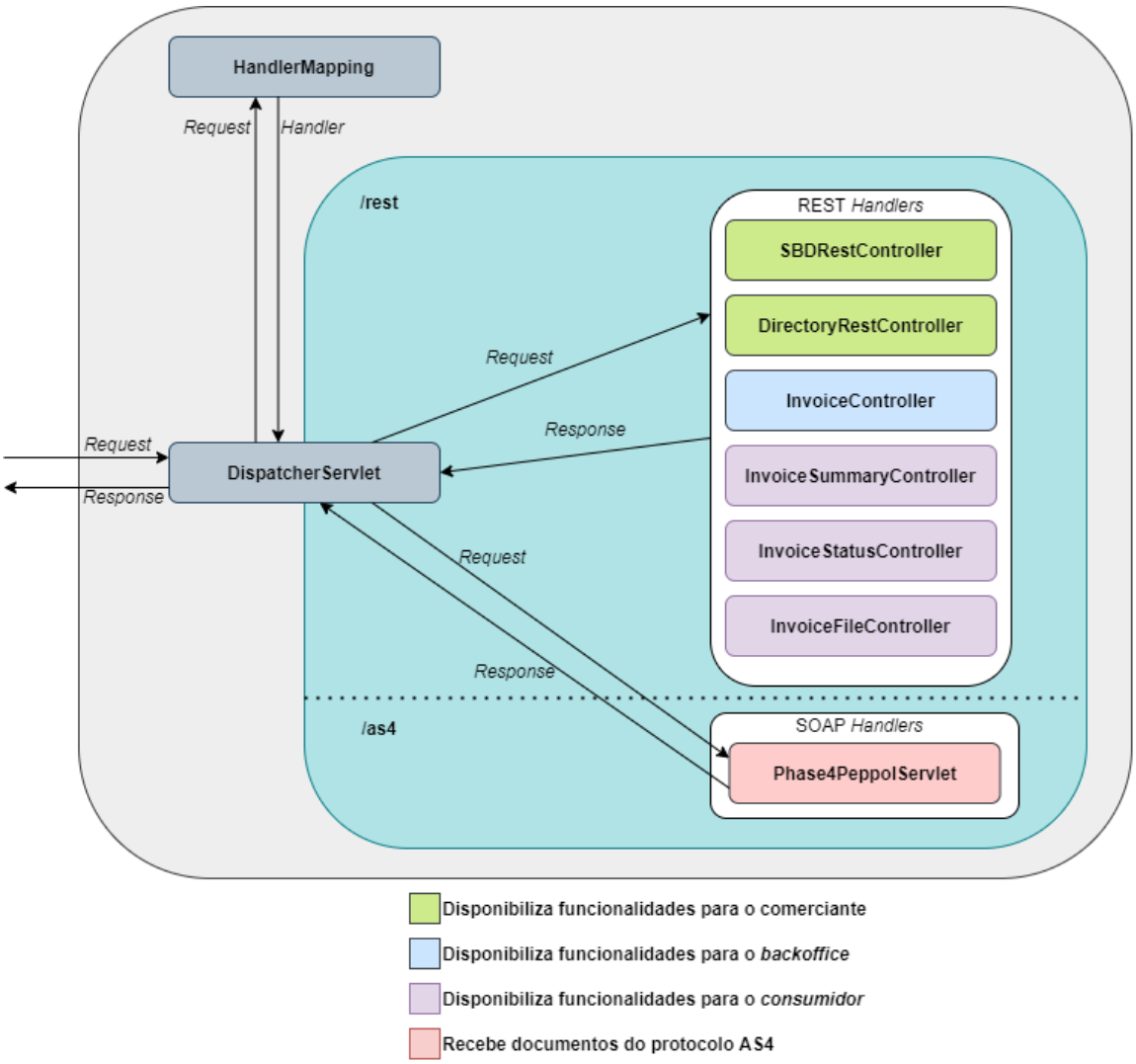


Figura 4.6: Módulo de Comunicação - *Handlers*

De um modo geral:

- `SBDRestController`: disponibiliza a operação que permite a um Comerciante enviar uma fatura para um Consumidor;
- `DirectoryRestController`: disponibiliza uma operação de consulta sobre os participantes servidos pelo *Access Point* e registados no SMP da prova de conceito;

- **`InvoiceController`:** disponibiliza operações de consulta sobre os detalhes de uma fatura persistida;
- **`InvoiceSummaryController`:** semelhante ao `InvoiceController`, mas com informação mais reduzida. Esta distinção é muito importante, porque a fatura possui imensos detalhes e existem cenários, onde queremos apenas um subconjunto deles. Por exemplo, no SAFTPRO Portais as listas são construídas com informação breve, mas suficiente para identificar univocamente as faturas e os seus detalhes só são disponibilizados integralmente, após outro pedido à API;
- **`InvoiceStatusController`:** oferece operações que permitem alterar o estado da fatura;
- **`InvoiceFileController`:** dispõe uma operação que retorna o conteúdo de uma fatura em PDF;
- **`Phase4PeppolServlet`:** permite a receção de documentos através do protocolo AS4 (descrito na secção 2.5.1.2).

Alguns dos *Controllers* disponibilizam exclusivamente operações CRUD, assim como os serviços e repositórios que comunicam. Por este motivo, na descrição dos *worfklows* (secção 4.2.1.6) não será utilizado o mesmo nível de detalhe para descrever todos os componentes da solução.

4.2.1.1 Autenticação e Autorização nas invocações à REST API

As operações definidas na REST API[9] estão divididas consoante o método de autenticação e autorização que utilizam. Ou seja, a porção do módulo que integra com o SAFTPRO Portais mantém a *Basic Authentication*, com injeção dos dados de sessão nos pedidos. Em contrapartida, a parte da API que complementa os *workflows* do Peppol está protegida por *Token-Based Authentication*, através de JWT[10].

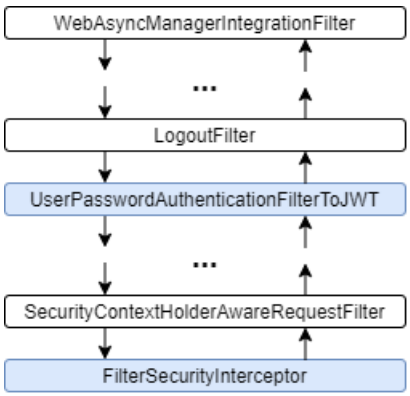


Figura 4.7: Spring Security: Cadeia de filtros

[9] Consultar anexo IX para visualizar a REST API.
[10] https://jwt.io/

Em Spring, a *framework* utilizada para a segurança de *software* intitula-se Spring *Security*[11]. Na camada *Web*, os *Controllers* são protegidos com base numa cadeia de filtros (fig. 4.7) e cada filtro tem responsabilidades diferentes. Mediante os serviços necessários para uma determinada configuração, é possível adicionar ou remover filtros da cadeia e sempre que exista dependências entre eles, a sua ordem deve ser considerada.

Por exemplo, a extração dos dados de sessão é feita no `FilterSecurityInterceptor` e assemelha-se a:

```
1  package pt.opensoft.saftpro.interceptor;
2
3  import ...
4
5  public class FilterSecurityInterceptor extends HandlerInterceptorAdapter {
6  ...
7      @Override
8      public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler)
9      throws Exception {
10         ...
11         // endpoints with /bearer use JWT
12         if (request.getRequestURI().contains("/rest/bearer/e-invoicing")) {
13             return true;
14         }
15         ...
16         // Getting user name & grants from request header
17         String username = request.getHeader(HeaderNameEnum.USER.getName()); // Vat Number or Email
18         if (StringUtil.isEmpty(username)) {
19             throw new UsernameNotFoundException();
20         }
21
22         String grants = request.getHeader(HeaderNameEnum.GRANT.getName()); // Grants
23         if (StringUtil.isEmpty(grants)) {
24             throw new GrantNotFoundException();
25         }
26         ...
27     }
28   ...
29 }
```

Listagem 14: SAFTPRO *Webservices* - `FilterSecurityInterceptor.java`

Em relação à REST API, a sua separação é efetuada através de diferentes prefixos (listagem 15). Isto é, quando na anotação `@GetMapping` existe o prefixo "/bearer" a API está protegida através de *tokens* JWT e o controlo de acessos é mais fino, porque utiliza informação do modelo de dados para autorizar uma determinada operação. Por exemplo, a anotação `@UserIsCustomer` (linha 19) implementa uma política de segurança, que verifica se o utilizador a quem foi atribuído o *token* JWT é igual ao utilizador para o qual se está a tentar obter os resumos das faturas.

Em contraste, quando o prefixo não está presente o controlo de acessos é feito através do *Role* do utilizador, por intermédio da anotação `@RolesAllowed`.

No anexo X seguem mais detalhes sobre a implementação da *Token-Based Authentication* e políticas de segurança.

[11] https://spring.io/projects/spring-security#overview

```java
package pt.opensoft.saftpro.restcontroller.einvoicing;

import ...

@RestController
@RequestMapping("/rest")
@Validated
public class InvoiceSummaryController {

    private InvoiceSummaryService invoiceSummaryService;
    ...
    @GetMapping(value = "/bearer/e-invoicing/invoices/summary", produces = MediaType.APPLICATION_JSON_VALUE)
    @UserIsCustomer
    public ResponseEntity<SaftproModel<List<InvoiceSummaryModel>>> getInvoicesSummariesByCustomerElectronicAddress(
        @RequestParam("customerElectronicAddressSchemeId") String customerElectronicAddressSchemeId,
        @RequestParam("customerElectronicAddress") String customerElectronicAddress) { ... }

    @GetMapping(value = "/e-invoicing/invoices/summary", produces = MediaType.APPLICATION_JSON_VALUE)
    @RolesAllowed(RoleUtil.BACKOFFICE)
    public ResponseEntity<SaftproModel<List<InvoiceSummaryModel>>> getInvoicesSummaries(
        @RequestBody @Valid SaftproModel<InvoiceSummarySearchCriteriaModel> model) { ... }
}
```

Listagem 15: SAFTPRO *Webservices* - *Controller* com diferentes prefixos

4.2.1.2 Autenticação e Autorização nas invocações à SOAP API (AS4)

Para o envio e receção de documentos são utilizadas as políticas do protocolo AS4. Na biblioteca phase4, isto resume-se à configuração das *keystores* e *truststores* que devem ser utilizadas para autenticar os *Access Points*, assim como assinar, cifrar e validar a integridade das mensagens.

```
# Phase4 Security Config
org.apache.wss4j.crypto.provider=org.apache.wss4j.common.crypto.Merlin
org.apache.wss4j.crypto.merlin.keystore.type=JKS
org.apache.wss4j.crypto.merlin.keystore.file=keystores/invalid-keystore-pw-peppol.jks
org.apache.wss4j.crypto.merlin.keystore.password=peppol
org.apache.wss4j.crypto.merlin.keystore.alias=1
org.apache.wss4j.crypto.merlin.keystore.private.password=peppol
org.apache.wss4j.crypto.merlin.truststore.type=JKS
org.apache.wss4j.crypto.merlin.truststore.file=keystores/complete-truststore.jks
org.apache.wss4j.crypto.merlin.truststore.password=peppol
```

Listagem 16: Application.properties - Configuração de segurança da biblioteca phase4

Como exemplo, podemos considerar o método `_checkReceiverAPCert` presente na classe `Phase4PeppolSender`(listagem 17), que é invocado antes do envio de uma mensagem AS4 e tem como objetivo garantir que o *Receiver Access Point* obtido no DNS *lookup* é fidedigno.

Naturalmente, as classes que a biblioteca disponibiliza para receber os documentos também implementam este tipo de validações. Mas, não se entra em detalhe, uma vez que a biblioteca é *Open-source* e passou no teste de conformidade do Peppol[12].

[12] https://github.com/phax/phase4/blob/master/docs/PEPPOL/TestBedReport-POP000306-20190906T103327.pdf

```
...
   private static void _checkReceiverAPCert(@Nullable final X509Certificate aReceiverCert,
        @Nullable final IPhase4PeppolCertificateCheckResultHandler aCertificateConsumer) throws Phase4PeppolException {
        ...
        final OffsetDateTime aNow = MetaAS4Manager.getTimestampMgr().getCurrentDateTime();
        final EPeppolCertificateCheckResult eCertCheckResult = PeppolCertificateChecker
                                                                .checkPeppolAPCertificate(aReceiverCert, aNow,
                                                                                          ETriState.UNDEFINED, null);

        // Interested in the certificate?
        if (aCertificateConsumer != null)
            aCertificateConsumer.onCertificateCheckResult(aReceiverCert, aNow, eCertCheckResult);

        if (eCertCheckResult.isInvalid()) {
            throw new Phase4PeppolException("The configured receiver AP certificate is not valid (at " +
                    aNow + ") and cannot be used for sending. Aborting. Reason: " + eCertCheckResult.getReason());
        }
        ...
    }
...
```

Listagem 17: Phase4PeppolSender.java - Método `_checkReceiverApCert()`

4.2.1.3 Tratamento de erros na REST API

De um modo geral, a estratégia global para tratamento de erros na REST API gira em torno das anotações `@ControllerAdvice` e `@ExceptionHandler`.

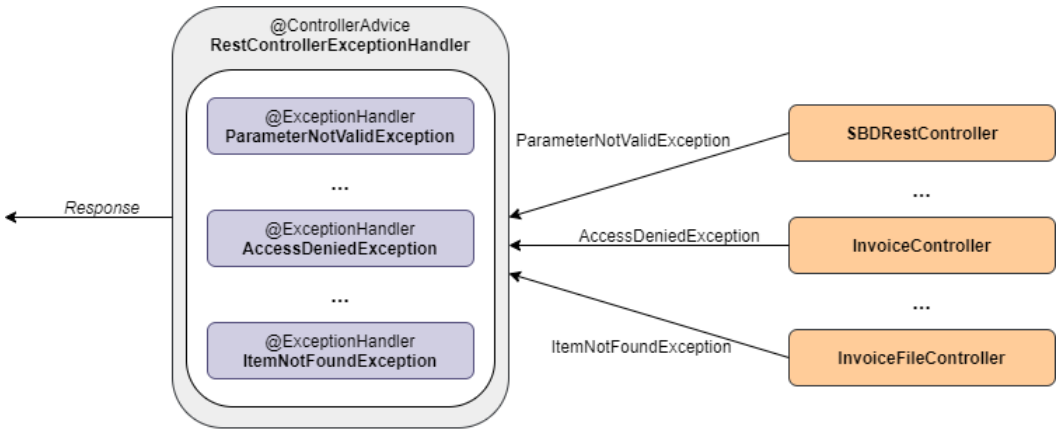


Figura 4.8: SAFTPRO *Webservices* - Estratégia de tratamento erros REST

O `@ControllerAdvice` permite consolidar, num único componente, diferentes `@ExceptionHandler` responsáveis por definir métodos que uniformizam o tratamento de erros relacionados com uma determinada exceção. Deste modo, podemos encarar o `@ControllerAdvice` como um intercetor de exceções dos métodos presentes nos *Controllers* e é boa prática definir um *handler* por exceção, para que o utilizador da API receba uma resposta adequada.

A título de exemplo, encontra-se na listagem 18 o método designado para tratar exceções do tipo `ItemNotFoundException`.

```java
package pt.opensoft.saftpro.handler;

import ...

@Slf4j
@ControllerAdvice
public class RestControllerExceptionHandler {
    ...
    @ExceptionHandler(ItemNotFoundException.class)
    public ResponseEntity<SaftproModel<?>> handleItemNotFoundException(ItemNotFoundException exception) {

        SaftproErrorModel error = new SaftproErrorModel();

        error.setStatus(HttpStatus.NOT_FOUND.value());
        error.setMessage(HttpStatus.NOT_FOUND.getReasonPhrase() + " " + exception.getMessage());

        SaftproModel<?> result = new SaftproModel<>();
        result.setErrors(error);

        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(result);
    }
    ...
}
```

Listagem 18: SAFTPRO *Webservices* - RestControllerExceptionHandler.java

4.2.1.4 Tratamento de erros na SOAP API (AS4)

Mais uma vez, no *endpoint* SOAP o tratamento de erros é assegurado ao nível da biblioteca phase4. Contudo, é importante saber distinguir os erros específicos relacionados com o protocolo de comunicação[13], dos erros que ocorrem no processamento de uma mensagem no contexto aplicacional.

Relativamente ao erros específicos do protocolo de comunicação, estes são tratados de uma maneira transparente pela biblioteca e não é necessário implementar qualquer tipo de lógica de negócio. Por exemplo, isto inclui a transmissão do erro "EBMS:0101" quando o módulo de segurança da biblioteca não consegue validar a assinatura presente no *header* de uma mensagem AS4, ou o erro "EBMS:0001" quando algum dos elementos presentes numa mensagem AS4 é irreconhecível, etc. Não obstante, o conteúdo do documento, presente no *payload* de uma mensagem AS4, é escrutinado apenas ao nível aplicacional e poderão existir situações em que um erro nesta camada impossibilita a transmissão do documento ao seu destinatário. Por exemplo, o *Receiver Access Point*, ao receber uma mensagem, verifica que o consumidor presente nos metadados da mensagem AS4 difere do consumidor indicado na fatura UBL. Nestas situações, o *Access Point* poderá recusar-se a terminar o *workflow* e é necessário configurar o *Servlet* disponibilizado pela biblioteca, para que possa gerar e transmitir o erro AS4 apropriado.

No contexto aplicacional, os erros originados no processamento de um documento podem ser comunicados ao *Sender Access Point* através do método booleano `exceptionTranslatesToAS4Error()` presente nos *handlers* registados para tratar um pedido. Quando este método retorna `true`, qualquer exceção que ocorra dentro do contexto de

[13] *Error Handling* (ver secção 6 em[36]).

execução do *handler* leva à criação do erro "EBMS:0004", que é transmitido sob a forma de uma `SignalMessage` ao *Sender Access Point*. Na biblioteca phase4, a expressividade da descrição do erro é um pouco limitada, pelo que se optou utilizar como descrição do erro a causa da exceção. Em simultâneo, é crucial ter a noção de que o erro "ebMS:0004" corresponde a um erro não contemplado pela especificação ebMS 3.0 e portanto nem todos os *Access Points* terão capacidade de os interpretar e agir em conformidade com a sua descrição. À luz desta informação, na solução atual quando um *Access Point* responde com este tipo de erro, a transmissão da mensagem AS4 é considerada um insucesso e o erro é transmitido ao utilizador, ficando ao seu critério interpretá-lo e entrar em contacto com o *Access Point*, fora de banda, para o poder resolver.

4.2.1.5 Integração da biblioteca phase4 no SAFTPRO *Webservices*

A biblioteca phase4 desempenha um papel fulcral na concretização dos *workflows* contemplados pelo Peppol, uma vez que implementa a camada de comunicação entre dois *Access Points*. A solução é genérica, mas tem suporte específico para o perfil AS4 do Peppol e tecnicamente pode ser integrada em qualquer aplicação *Servlet-based*.

Para o envio e receção de mensagens AS4, destacam-se os seguintes componentes que ajudam a compor a biblioteca:

- **phase4-peppol-client:**
  - Disponibiliza a classe `Phase4PeppolSender`, que possui um *Builder* capaz de personalizar os detalhes de envio de mensagem AS4 para a rede Peppol;
  - Contem exemplos do envio de mensagens AS4 para diversos *endpoints*;
  - Opcionalmente, oferece a validação sintática e semântica dos documentos, antes de os enviar;
  - Requer apenas a configuração das propriedades específicas da biblioteca (como exemplificado na listagem 16) para ser utilizado no contexto aplicacional do Spring.
- **phase4-peppol-servlet:**
  - Funciona como ponto de entrada para receber mensagens da rede Peppol;
  - Assume que está a correr sobre um *Servlet engine*, como o Apache Tomcat[14] (*default* do Spring) ou o Eclipse Jetty[15], o que torna a sua integração mais complicada;
  - Endereça os pedidos através de *handlers*, que assentam na metodologia *Service Providers Interface*[16] e implementam a interface `IPhase4PeppolIncomingSBDHandlerSPI`;
  - Por omissão, verifica se é o destino das mensagens através de um SMP *lookup* composto pelo identificador do *Receiver* (consumidor), o tipo de processo e o tipo do documento. No entanto, este comportamento pode ser alterado;

[14] http://tomcat.apache.org/
[15] https://www.eclipse.org/jetty/
[16] https://docs.oracle.com/javase/tutorial/sound/SPI-intro.html

– À semelhança do phase4-peppol-client, as chaves e certificados utilizados para as assinaturas digitais e restantes mecanismos criptográficos são carregados através das propriedades presentes no ficheiro `Application.properties` e acessíveis através da classe `AS4CryptoFactoryProperties`.

Ainda em relação ao phase4-peppol-servlet, os *handlers* deverão ser declarados na *package* `META-INF/services`, para que o *Servlet* os consiga reconhecer e carregar no arranque do servidor.

```
1 pt.opensoft.saftpro.demo.spi.StoringPeppolIncomingSBDHandlerSPI
```

Listagem 19: com.helger.phase4.peppol.servlet.IPhase4PeppolIncomingSBDHandlerSPI

```
1  ...
2  <configuration>
3          <instructions>
4                  <Automatic-Module-Name>com.helger.phase4.peppol.servlet</Automatic-Module-Name>
5                  <Export-Package>com.helger.phase4.peppol.servlet.*</Export-Package>
6                  <Import-Package>!javax.annotation.*,*</Import-Package>
7                  <Require-Capability>osgi.extender; filter:="(osgi.extender=osgi.serviceloader.registrar)",
8                          osgi.extender; filter:="(osgi.extender=osgi.serviceloader.processor)",
9                          osgi.serviceloader;
10                         filter:=
11                         "(osgi.serviceloader=com.helger.phase4.peppol.servlet.IPhase4PeppolIncomingSBDHandlerSPI)";
12                         cardinality:=multiple; resolution:=optional
13                 </Require-Capability>
14                 <Provide-Capability>osgi.serviceloader;
15                         osgi.serviceloader=com.helger.phase4.servlet.spi.IAS4ServletMessageProcessorSPI
16                 </Provide-Capability>
17         </instructions>
18 </configuration>
19 ...
```

Listagem 20: Ficheiro pom.xml do subprojeto phase4-peppol-servlet

No phase4, a complexidade do protocolo AS4 é atenuada através da divisão de responsabilidades no processamento dos pedidos. Ou seja, estes são processados através de um *Servlet Handler* que por sua vez possui o *Request Handler*, onde se regista o `IAS4ServletMessageProcessorSPI` responsável por carregar todos os `IPhase4PeppolIncomingSBDHandlerSPI`. Consequentemente, a lógica para entregar e persistir as faturas deverá centrar-se no `IPhase4PeppolIncomingSBDHandlerSPI`, visto que é o componente da biblioteca onde existe acesso aos documentos presentes nas mensagens AS4.

4.2.1.5.1 Estratégia de integração entre o phase4-peppol-servlet e o Spring

A estratégia para integrar o *Servlet* na aplicação consiste em registá-lo como *Bean* (listagem 21), tirando partido do servidor aplicacional embebido.

Deste modo, é possível aceder às abstrações e funcionalidades do Spring, como por exemplos os repositórios JPA, que permitem a extração e persistência dos detalhes das faturas.

```java
package pt.opensoft.saftpro.configuration.einvoicing.servlet;

import ...

@Configuration
@Slf4j
public class ServletConfig implements ServletContextAware {

    @Value("${phase4.url-mapping}")
    private String urlMapping;
    ...
    private ServletContext servletContext;

    @Override
    public void setServletContext(@Nonnull ServletContext servletContext) {
        this.servletContext = servletContext;
    }

    @Bean
    public ServletRegistrationBean<Phase4PeppolServlet> peppolServletRegistrationBean() {
        init(servletContext);

        ServletRegistrationBean<Phase4PeppolServlet> bean =
            new ServletRegistrationBean<>(new Phase4PeppolServlet(), true, urlMapping);
        bean.setLoadOnStartup(1);

        return bean;
    }

    private void init(ServletContext servletContext) {
        if (!WebScopeManager.isGlobalScopePresent()) {
            WebScopeManager.onGlobalBegin(servletContext);
            initGlobalSettings();
            initAS4();
            initPeppolAS4();
        }
    }

    private void initGlobalSettings() {
        ...
        AS4ProfileSelector.setCustomAS4ProfileID(AS4PeppolProfileRegistarSPI.AS4_PROFILE_ID); // Enforce Peppol profile
        ...
    }

    private void initAS4() {
        AS4ServerInitializer.initAS4Server();
    }

    private void initPeppolAS4() {
        // Check if crypto properties are okay
        final KeyStore keyStore = AS4CryptoFactoryProperties.getDefaultInstance().getKeyStore();
        if (keyStore == null)
            throw new InitializationException("Failed to load configured Keystore");

        final KeyStore.PrivateKeyEntry privateKeyEntry =
            AS4CryptoFactoryProperties.getDefaultInstance().getPrivateKeyEntry();
        if (privateKeyEntry == null)
            throw new InitializationException("Failed to load configured private key");
        ...
    }
    ...
}
```

Listagem 21: SAFTPRO *Webservices* - ServletConfig.java

Contudo, é preciso ter algumas considerações porque os *handlers* utilizados pelo *Servlet* são carregados através de um *loader*, o que os impossibilita de ser geridos pelo Spring e aceder diretamente ao `ApplicationContext`, onde se encontram registados os serviços que implementam a lógica de negócio.

Para contornar este contratempo foi disponibilizada uma classe que implementa a interface `ApplicationContextAware` e possui o método `setApplicationContext()`, invocado na inicialização do objeto para injetar o `ApplicationContext` do Spring *Container*. Nesta perspetiva, a classe funciona como um *placeholder* do `ApplicationContext` e torna-o acessível aos objetos que não são geridos explicitamente pelo Spring.

```
1  package pt.opensoft.saftpro.component.einvoicing;
2
3  import ...
4
5  @Component
6  public class SDBHandlerServiceLocator implements ApplicationContextAware {
7
8      private static ApplicationContext context;
9
10      /**
11      * Returns the Spring managed bean instance of the given service if it exists.
12      * Returns null otherwise.
13      */
14     public static SBDHandlerService getService(SBDHandlerServiceSelectorEnum selector) {
15         try {
16             return context.getBean(selector.getLabel(), SBDHandlerService.class);
17         } catch (BeansException e) {
18             return null;  // Nothing to be done here
19         }
20     }
21
22     @Override
23     public void setApplicationContext(ApplicationContext context) throws BeansException {
24         // store ApplicationContext reference to access required beans later on
25         SBDHandlerServiceLocator.context = context;
26     }
27 }
```

Listagem 22: SAFTPRO *Webservices* - `SDBHandlerServiceLocator.java`

A abordagem descrita funciona, mas deve-se ter presente que vai um pouco contra o princípio de inversão de dependências promovido pelo Spring. Ainda assim, é a alternativa que minimiza as alterações ao *core* da biblioteca e permite fazer uma ponte entre o código dos *handlers* e o contexto aplicacional do Spring. Para minimizar este aspeto, foi utilizado *Service-Locator-Pattern*[17], com o intuito de oferecer algum nível de abstração e uniformizar o padrão de interação dos serviços que interagem diretamente com os *handlers*.

Na listagem 22, encontra-se a classe *Singleton*[18] `SBDHandlerServiceLocator`, cujo o objetivo é fornecer aos *handlers* o serviço correspondente. Assim, garante-se que todos os serviços disponibilizados implementam a interface `SBDHandlerService` (listagem 23) e que este objeto corresponde à única ponte entre o código Spring e os *handlers* da biblioteca.

[17] https://martinfowler.com/articles/injection.html#UsingAServiceLocator
[18] https://refactoring.guru/design-patterns/singleton

```java
package pt.opensoft.saftpro.service.einvoicing.sbd.api;

import ...

public interface SBDHandlerService {

    void handle() throws Phase4Exception;

    void setMessageMetadata(IAS4IncomingMessageMetadata messageMetadata);

    void setHttpHeaders(HttpHeaderMap httpHeaders);

    void setUserMessage(Ebms3UserMessage userMessage);

    void setStandardBusinessDocument(StandardBusinessDocument standardBusinessDocument);

    void setStandardBusinessDocumentBytes(byte[] standardBusinessDocumentBytes);

    void setPeppolStandardBusinessDocumentHeader(PeppolSBDHDocument peppolStandardBusinessDocumentHeader);

    void setMessageState(IAS4MessageState messageState);

}
```

Listagem 23: SAFTPRO *Webservices* - `SBDHandlerService.java`

De seguida temos o exemplo de um serviço que implementa esta interface:

```java
package pt.opensoft.saftpro.service.einvoicing.sbd.impl;

import ...

@Service(SBDHandlerServiceSelector.Constants.CUSTOM_PEPPOL_INCOMING_VALUE)
public class CustomPeppolIncomingSBDHandlerServiceImpl implements SBDHandlerService {

    // Non-managed Spring fields
    private IAS4IncomingMessageMetadata messageMetadata;
    ...
    private IAS4MessageState messageState;

    // Managed Spring fields
    private InvoiceRepository repository;
    ...

    @Override
    public void handle() throws Exception {
        // Very complex logic that leads to persisting the document in a database.
        Invoice invoice = new Invoice();
        invoice.setId("Very unique id");

        this.repository.save(invoice);
    }
    ...
}
```

Listagem 24: `SBDHandlerService` - Exemplo de implementação

E o exemplo de invocação num *handler* na listagem 25.

```java
package pt.opensoft.saftpro.spi;

import ...

@IsSPIImplementation
public class CustomPeppolIncomingSBDHandlerSPI implements IPhase4PeppolIncomingSBDHandlerSPI {

    public void handleIncomingSBD(@Nonnull final IAS4IncomingMessageMetadata aMessageMetadata,
                                  @Nonnull final HttpHeaderMap aHeaders,
                                  @Nonnull final Ebms3UserMessage aUserMessage,
                                  @Nonnull final byte[] aSBDBytes,
                                  @Nonnull final StandardBusinessDocument aSBD,
                                  @Nonnull final PeppolSBDHDocument aPeppolSBD,
                                  @Nonnull final IAS4MessageState aState) throws Exception {
        SBDHandlerService service = SDBHandlerServiceLocator
                .getService(SBDHandlerServiceSelector.CUSTOM_PEPPOL_INCOMING);

        // Inject the parameters into the service
        service.setMessageMetadata(aMessageMetadata);
        ...

        // Handle the request, might raise an exception
        service.handle();
    }
 ...
}
```

Listagem 25: `IPhase4PeppolIncomingSBDHandlerSPI` - Exemplo de implementação

Alternativas

A outra estratégia idealizada para integrar o phase4 no SAFTPRO *Webservices*, consiste em separar o *deploy* da aplicação Spring, onde é desenvolvida a lógica de negócio, do *Servlet*, onde o pedido é efetivamente recebido. Neste contexto, para a aplicação Spring aceder a um pedido deverá ser efetuado um *redirect* nos *handlers* do *Servlet* para a aplicação. O problema desta abordagem reflete-se na duplicação dos pedidos, com a agravante de estes terem um *payload* associado, assim como o desperdício de recursos em ambos os componentes, para replicar a lógica que trata das respostas. Para além disso, a biblioteca não foi concebida para ser disponibilizada como uma aplicação *standalone*, pelo que teria de ser investido esforço extra para concretizar esta abordagem[19].

No outro extremo do espectro, poder-se-ia recorrer à implementação do protocolo AS4 de raíz, uma vez que a sua especificação é pública. Mas, a hipótese foi descartada porque iria dominar a fase de desenvolvimento da prova de conceito, de modo que optou-se por uma solução *Open-source* com potencialidade comercial.

4.2.1.6 *Worfklows*

Dos *workflows* considerados pelo Peppol (ver secção 2.5.1.1), o módulo de *e-Invoicing* suporta o registo de um participante no SMP e a troca de documentos entre parceiros comerciais (fig. 4.9).

[19] Também existem implementações *Open-source* do protocolo em Java disponibilizadas como aplicações *standalone*, das quais se destacam o Oxalis (`https://github.com/OxalisCommunity/oxalis`) e o Holodeck B2B (`http://holodeck-b2b.org/`).

De momento, o registo do SMP e dos participantes no *Service Metadata Locator* é irrealizável porque não existe internamente um certificado Peppol, que nos permita ligar oficialmente à rede.

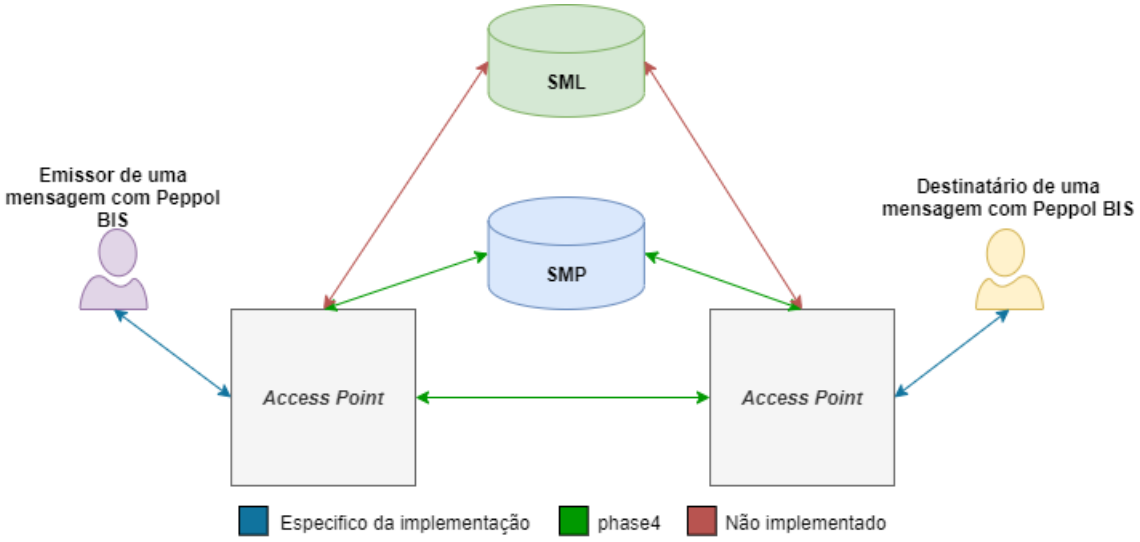


Figura 4.9: Peppol - Implementação do *4-Corner Model*

Em relação ao Consumidor, os *workflows* do Peppol são insuficientes para atingir os objetivos da tese, tornando necessário complementar a solução com outras funcionalidades. Nomeadamente, a disponibilização de uma API para que o Consumidor possa intervir ativamente no processo de aprovação/rejeição de uma fatura e um mecanismo de resiliência que lhe permita fazer o *download* passivo da fatura, caso exista algum problema na entrega em tempo real.

Um *Access Point* em conformidade com o Peppol deve ser capaz de enviar e receber mensagens da rede. Naturalmente, o módulo de *e-Invoicing* suporta ambas as funcionalidades, mas a receção de um documento é mais elaborada que o seu envio, pelo que serão abordados em separado.

4.2.1.6.1 Concretização do registo de um participante

No Peppol distinguem-se dois tipos de participantes, o *Sender* e o *Receiver*. A troca de documentos presume que o *Sender* tem acordo com um *Access Point* credenciado para poder utilizar a rede.

Como foi descrito em 2.5.1.1.2, se o *Sender* utilizar a rede apenas para enviar documentos, não necessita de se registar explicitamente num SMP. Assim sendo, o modelo de *Sender* contemplado na prova de conceito, baseia-se nesta premissa e o seu registo ocorre apenas localmente no sistema, para possa aceder à operação de envio de faturas. De momento, este processo de registo é simulado e consiste em persistir o identificador Peppol do *Sender*, assim como o seu *username* e *password* na base de dados (listagem 26).

```
...
    // Sender registration
    if (!peppolParticipantJpaRepository.findByUsername("issuer").isPresent()) {
            PeppolParticipantEntity peppolIssuer = new PeppolParticipantEntity();

            peppolIssuer.setParticipantId("9915:phase4-test-issuer");
            peppolIssuer.setUsername("issuer");
            peppolIssuer.setPassword(passwordEncoder.encode("password"));
            peppolIssuer.setRegistrationName("Issuer Company");
            peppolIssuer.setVatId("SE4234235555");

            peppolParticipantJpaRepository.save(peppolIssuer);
    }
...
}
```

Listagem 26: SAFTPRO *Webservices* - Exemplo de registo de um *Sender*

O registo dos *Receivers* é semelhante (listagem 27), mas são necessários campos adicionais. Pois, para notificá-los sobre uma nova fatura estão a ser utilizados *webhooks*[20], que facilitam a transmissão de informação em tempo real e evitam que o Consumidor esteja constantemente a interrogar o sistema, para verificar se recebeu novos documentos.

```
...
    // Recipient registration
    if (!peppolParticipantJpaRepository.findByUsername("jleal").isPresent()) {
            PeppolParticipantEntity peppolRecipient = new PeppolParticipantEntity();

            peppolRecipient.setParticipantId("9946:pt231971460");
            peppolRecipient.setUsername("jleal");
            peppolRecipient.setPassword(passwordEncoder.encode("password"));
            peppolRecipient.setRegistrationName("José Leal");
            peppolRecipient.setVatId("pt231971460");

            // Recipient needs 3 additional fields to allow invoice notifications
            peppolRecipient.setEvent(WebhookEventEnum.INVOICE_NOTIFICATION);
            peppolRecipient.setPayloadUrl("https://localhost:8082/callback");
            peppolRecipient.setSecret("#verysecret");

            peppolParticipantJpaRepository.save(peppolRecipient);

            // Lastly register at SMP the participant
            registerParticipant(peppolRecipient.getParticipantId());
    }
...
}
```

Listagem 27: SAFTPRO *Webservices* - Exemplo de registo de um *Receiver*

Para além disso, é necessário registar o participante no SMP cujo modelo de dados centra-se no `ServiceGroup`, que representa o conjunto de serviços associados a um participante e é composto por uma lista de referências para a estrutura `SignedServiceMetadata`. O `ServiceMetadata`, associa um participante com a sua capacidade para receber um tipo de documento sobre um protocolo específico. Ou por outras palavras, disponibiliza toda a informação necessária para que um *Sender Access Point* possa enviar uma mensagem para

[20] https://www.techtarget.com/searchapparchitecture/definition/webhook

o *Receiver Access Point* desse participante. Por último, o `SignedServiceMetadata` não é mais do que o `ServiceMetadata` assinado pelo SMP.

No phoss SMP, o registo de um participante pode ser efetuado de duas maneiras distintas:

- Manualmente através de uma interface gráfica;
- Ou, através da sua REST API[21] (listagem 28).

```
...
    private void registerParticipant(String participantId) throws Exception {
        // Create new client for the SMP
        SMPClient smpClient = new SMPClient(new URI(smpUrl));
        BasicAuthClientCredentials credentials = new BasicAuthClientCredentials(smpUsername, smpPassword);

        // Append BusDox prefix to identifiers
        IParticipantIdentifier serviceGroupId =
            PeppolIdentifierFactory.INSTANCE.createParticipantIdentifierWithDefaultScheme(participantId);
        IDocumentTypeIdentifier documentId =
            PeppolIdentifierFactory.INSTANCE.createDocumentTypeIdentifierWithDefaultScheme(documentTypeId);
        IProcessIdentifier processId =
            PeppolIdentifierFactory.INSTANCE.createProcessIdentifierWithDefaultScheme(processTypeId);

        // Register Service Group (Participant) at SMP
        smpClient.saveServiceGroup(serviceGroupId, credentials);

        // Create information regarding the AP that will serve the participant being registered
        ServiceInformationType serviceInformation = new ServiceInformationType();
        serviceInformation.setParticipantIdentifier(new SimpleParticipantIdentifier(serviceGroupId));
        serviceInformation.setDocumentIdentifier(new SimpleDocumentTypeIdentifier(documentId));

        EndpointType endpoint = new EndpointType();
        endpoint.setEndpointReference(new W3CEndpointReferenceBuilder().address(apUrl).build());
        endpoint.setTransportProfile(ESMPTransportProfile.TRANSPORT_PROFILE_PEPPOL_AS4_V2.getID());
        endpoint.setCertificate(getAPCertificateAsPEMString());
        endpoint.setServiceActivationDate(LocalDateTime.now());
        endpoint.setServiceDescription("Invoice Demo");
        endpoint.setServiceExpirationDate(LocalDateTime.now().plusYears(1));
        endpoint.setTechnicalContactUrl("jleal@opensoft.pt");
        endpoint.setRequireBusinessLevelSignature(false);

        ServiceEndpointList serviceEndpointList = new ServiceEndpointList();
        serviceEndpointList.getEndpoint().add(endpoint);

        ProcessType process = new ProcessType();
        process.setProcessIdentifier(new SimpleProcessIdentifier(processId));
        process.setServiceEndpointList(serviceEndpointList);

        // A participant can be enrolled in more than 1 process, so we need a list
        ProcessListType processList = new ProcessListType();
        processList.getProcess().add(process);

        serviceInformation.setProcessList(processList);

        // Register the endpoint responsible for the Service Group, Document Type and Process Type
        smpClient.saveServiceInformation(serviceInformation, credentials);
    }
...
```

Listagem 28: Phoss SMP - registo de um participante através da API

[21] https://github.com/phax/phoss-smp/wiki/REST-API

4.2.1.6.2 Concretização do envio de documentos

```
1  package pt.opensoft.saftpro.restcontroller.einvoicing;
2
3  import ...
4
5  @RestController
6  @RequestMapping("/rest")
7  public class SBDRestController {
8  ...
9      @PostMapping(value = "/bearer/e-invoicing/sbd", consumes = {MediaType.MULTIPART_FORM_DATA_VALUE},
10                  produces = MediaType.APPLICATION_JSON_VALUE)
11     public ResponseEntity<SaftproModel<?>> send(HttpServletRequest httpServletRequest, Authentication user)
12                                                 throws Exception {
13         ServletFileUpload upload = new ServletFileUpload();
14
15         FileItemIterator iter = upload.getItemIterator(httpServletRequest);
16
17         byte[] businessDocumentBytes = null;
18         DNSLookUpModel dnsLookUp = null;
19
20         while (iter.hasNext()) {
21             FileItemStream item = iter.next();
22
23             if (item.getFieldName().equalsIgnoreCase("businessDocument")) {
24                 businessDocumentBytes = IOUtils.toByteArray(item.openStream());
25             } else if (item.getFieldName().equalsIgnoreCase("dnsLookUp")) {
26                 dnsLookUp = JsonUtil.fromJson(item.openStream(), DNSLookUpModel.class);
27             } else {
28                 throw new ParameterNotValidException("Unknown parameter with name '" + item.getFieldName() + "'");
29             }
30         }
31
32         if (!securityService.matchesSenderParticipantId(dnsLookUp.getSenderParticipantId(), user)) {
33             throw new AccessDeniedException("The authenticated user does not match the"
34                 + "participant identifier on the routing information");
35         }
36
37         sbdService.send(dnsLookUp, businessDocumentBytes);
38
39         return ResponseEntity.ok(new SaftproModel<>());
40     }
41 }
```

Listagem 29: SAFTPRO *Webservices* - SBDRestController.java

O *endpoint* definido para o envio de faturas (listagem 29) recebe um pedido HTTP *Multipart*, composto pelo documento em XML mais a informação de *routing* (SenderParticipantId, ReceiverParticipantId, DocumentTypeId, ProcessId) contida num modelo JSON. O cabeçalho do método `send()` é um pouco invulgar, porque recebe diretamente o `HttpServletRequest` e o utilizador autenticado. Isto acontece, porque o *upload* de ficheiros SAF-T não tira proveito do `MultipartResolver` pré-configurado pelo Spring, uma vez que os ficheiros submetidos podem ter grandes dimensões (+4Gb).

Consequentemente, a abordagem para *upload* de ficheiros UBL 2.1 adotou uma abordagem semelhante, que consiste em extrair os componentes do corpo do pedido HTTP manualmente através das funcionalidades de *upload* do Apache.

```java
package pt.opensoft.saftpro.service.einvoicing;

import ...

@Service
public class SBDService {
...
    public void send(DNSLookUpModel dnsLookUp, byte[] businessDocumentBytes) {
        Set<ConstraintViolation<DNSLookUpModel>> violations = constraintValidator.validate(dnsLookUp);

        if (!violations.isEmpty()) {
            throw new ConstraintViolationException("Invalid routing information", violations);
        }

        // Validate business document, handler throws SBDValidationException if the result contains atleast 1 error
        sbdValidator.validate(domParser.parseXml(new ByteArrayInputStream(businessDocumentBytes)),
              new BusinessDocumentValidationResultHandler());

        // Append BusDox prefix to identifiers
        IParticipantIdentifier senderId = PeppolIdentifierFactory.INSTANCE
              .createParticipantIdentifierWithDefaultScheme(dnsLookUp.getSenderParticipantId());
        IDocumentTypeIdentifier documentTypeId = PeppolIdentifierFactory.INSTANCE
              .createDocumentTypeIdentifierWithDefaultScheme(dnsLookUp.getDocumentTypeId());
        IProcessIdentifier processId = PeppolIdentifierFactory.INSTANCE
              .createProcessIdentifierWithDefaultScheme(dnsLookUp.getProcessId());
        IParticipantIdentifier receiverId = PeppolIdentifierFactory.INSTANCE
              .createParticipantIdentifierWithDefaultScheme(dnsLookUp.getReceiverParticipantId());

        try {

            IAS4EndpointDetailProvider endpointDetailProvider =
                new AS4EndpointDetailProviderPeppol(new SMPClientReadOnly(new URI(smpUrl)));
            endpointDetailProvider.init(documentTypeId, processId, receiverId);

            String receiverAPUrl = endpointDetailProvider.getReceiverAPEndpointURL();

            SenderSignalMessageConsumer senderSignalMessageConsumer = new SenderSignalMessageConsumer();

            // Try to send message to Receiver AP
            ESimpleUserMessageSendResult result = Phase4PeppolSender.builder()
                    .documentTypeID(documentTypeId)
                    .processID(processId)
                    .senderParticipantID(senderId)
                    .receiverParticipantID(receiverId)
                    .senderPartyID(partyId)
                    .payload(businessDocumentBytes)
                    .endpointDetailProvider(endpointDetailProvider)
                    .signalMsgConsumer(senderSignalMessageConsumer)
                    .sendMessageAndCheckForReceipt(e -> {
                        // Unwrap exception and translate it into an internal server error
                        Throwable unwrappedException = e.getCause();
                        throw new SBDSendException(unwrappedException.getMessage());
                    });

            if (result.isFailure() && senderSignalMessageConsumer.hasErrorEntries()) {
                // We were able to transmit the document but the receiver AP refused to process it due to an error
                throw new SBDSendException(senderSignalMessageConsumer.getError().toString());
            }

        } catch (Phase4Exception | URISyntaxException | GeneralSecurityException e) {
            // Failure locating the SML/SMP or there was a problem with the URI
            throw new SBDSendException(e.getMessage());
        }
    }
...
}
```

Listagem 30: SAFTPRO *Webservices* - SBDService.java

Em seguida, o documento é entregue ao SBDService (listagem 30) que:

- **Linhas 9 a 13:** valida a informação de *routing*;
- **Linhas 16 a 17:** valida a fatura de acordo com as regras do Peppol BIS *Billing* 3.0 (mais detalhes em 4.2.2.1);
- **Linhas 31 a 35:** interroga o SMP para obter o *Receiver* AP;
- **Linhas 40 a 53**: transmite o documento e aguarda a resposta do *Receiver* AP.

Para poder enviar uma fatura, o emissor deverá autenticar-se perante o módulo de *e-Invoicing* (listagem 31). Em caso de sucesso, o utilizador recebe um *token* que lhe permite efetuar o pedido que despoleta a transmissão de um documento pela rede (listagem 32).

```
1  ...
2      private String getToken(String username, String password) {
3          HttpHeaders requestHeaders = new HttpHeaders();
4          requestHeaders.setContentType(MediaType.APPLICATION_JSON);
5
6          LoginModel login = new LoginModel(username, password);
7
8          HttpEntity<LoginModel> requestEntity = new HttpEntity<>(login, requestHeaders);
9
10         ResponseEntity<SaftproModel> result =
11                 restTemplate.exchange(peppolAs4AuthenticationServiceEndpoint, HttpMethod.POST,
12                                       requestEntity, SaftproModel.class);
13         if (result.getStatusCode().is2xxSuccessful()) {
14                 return result.getHeaders().get(HttpHeaders.AUTHORIZATION).get(0);
15         }
16
17         return null;
18     }
19 ...
```

Listagem 31: Exemplo de obtenção do *token* JWT

```
1  ...
2      private <T> ResponseEntity<T> sendSBD(String path, DNSLookUpModel dnsLookUp, String token, Class<T> responseType)
3                                            throws RestClientException {
4         HttpHeaders requestHeaders = new HttpHeaders();
5         requestHeaders.setContentType(MediaType.MULTIPART_FORM_DATA);
6         requestHeaders.add(HttpHeaders.AUTHORIZATION, token);
7
8         MultiValueMap<String, Object> body = new LinkedMultiValueMap<>();
9
10        HttpHeaders attachmentHeaders = new HttpHeaders();
11        attachmentHeaders.setContentType(MediaType.APPLICATION_XML);
12        HttpEntity<FileSystemResource> attachmentPart = new HttpEntity<>(new FileSystemResource(path), attachmentHeaders);
13        body.set("businessDocument", attachmentPart); // Invoice as an attachment
14
15        HttpHeaders jsonHeaders = new HttpHeaders();
16        jsonHeaders.setContentType(MediaType.APPLICATION_JSON);
17        HttpEntity<DNSLookUpModel> jsonPart = new HttpEntity<>(dnsLookUp, jsonHeaders);
18        body.set("dnsLookUp", jsonPart);  // Additional endpoint information
19
20        HttpEntity<MultiValueMap<String, Object>> requestEntity = new HttpEntity<>(body, requestHeaders);
21        return restTemplate.exchange(peppolAs4Endpoint, HttpMethod.POST, requestEntity, responseType);
22    }
23 ...
```

Listagem 32: Exemplo de invocação do *endpoint* para envio de um documento

4.2.1.6.3 Concretização da receção de documentos

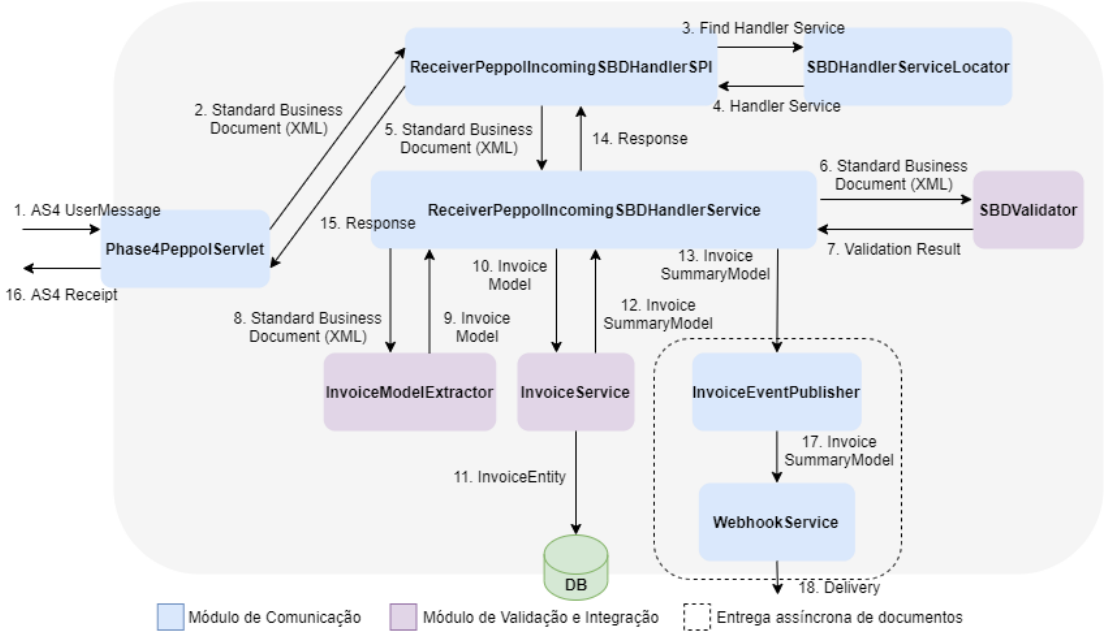


Figura 4.10: SAFTPRO *Webservices* - Receção de uma mensagem AS4

A figura 4.10 descreve a receção de um documento e expressa as interações entre os dois módulos da solução, que culmina na persistência da fatura e a sua entrega ao Consumidor.

Do ponto de vista do módulo de comunicação, a sua função consiste em receber o documento e transmiti-lo ao `IPhase4PeppolIncomingSBDHandlerSPI`, para que depois, de acordo com a estratégia de integração, o `SBDHandlerService` (listagem 33) se encarregue do restante processamento que consiste em:

- **Linhas 10 a 13:** validar as mensagens AS4 recebidas e garantir que estas são direcionadas para um consumidor registado no sistema;
- **Linhas 20 a 27:** validar o documento de acordo com as regras do Peppol e quaisquer outras definidas pelo país em que for instalada a solução;
- **Linhas 34 a 37:** garantir que a informação de *routing* que se encontra nos *headers* reflete o que está na fatura;
- **Linhas 31 a 32 e 39:** extrair a informação relevante do XML e fazer a sua persistência na base de dados;
- **Linhas 41 a 46:** invocar o *webhook* do destinatário para entregar o documento.

No Peppol, um *Access Point* age em prol dos seus participantes e portanto o envio de um documento, com sucesso, não implica que este já tenha chegado efetivamente ao destinatário final. Aliás, esta abordagem até pode ser prejudicial porque a confirmação

da receção pelo Consumidor pode ser demorada, o que aumenta o tempo de espera no lado do emissor.

```
...
    @Transactional(transactionManager = "eInvoicingTransactionManager", propagation = Propagation.REQUIRED,
                   rollbackFor = Exception.class)
    public void handle() throws Phase4Exception {
        if (!peppolStandardBusinessDocumentHeader.hasBusinessMessage()) {
            throw new Phase4Exception("AS4 message with ID '" + messageState.getMessageID() +
                    "' does not have a business document. Please try again");
        }

        if (!peppolParticipantService.exists(peppolStandardBusinessDocumentHeader.getReceiverValue())) {
            throw new Phase4Exception("Recipient of AS4 message with ID '" + messageState.getMessageID() +
                    "' is not registered. Please try again");
        }

        Element businessMessage = peppolStandardBusinessDocumentHeader.getBusinessMessage();

        BusinessDocumentValidationResultHandler businessDocumentValidationResultHandler =
            new BusinessDocumentValidationResultHandler();

        try {
            sbdValidator.validate(businessMessage, businessDocumentValidationResultHandler);
        } catch (SBDValidationException e) {
            throw new Phase4Exception("Business document of AS4 message with message ID '" +
                    messageState.getMessageID() + "' is invalid. Please try again");
        } catch (IllegalStateException | IllegalArgumentException e) {
            ...
        }

        try {
            // Extract all relevant information from the business document
            InvoiceModel invoiceModel =
                        this.invoiceModelExtractor.extractInvoiceModel(businessMessage.getOwnerDocument();

            if (!participantsMatch(invoiceModel)) {
                throw new Phase4Exception("Participants within the AS4 message with message ID '" +
                    messageState.getMessageID() + "' do not match the ones in the business document. Please try again");
            }

            invoiceModel = this.invoiceService.create(invoiceModel);

            byte[] invoiceXML = humanReadableService.convertDocumentToByteArray(businessMessage.getOwnerDocument());

            InvoiceSummaryModel invoiceSummary = this.invoiceSummaryService.create(invoiceModel, invoiceXML);

            // Generate Async event to trigger the delivery
            this.newInvoiceEventPublisher.publishNewInvoiceEvent(invoiceSummary);

            log.info("Successfully validated business document of AS4 message with message ID '" +
                    messageState.getMessageID() + "'. The document was persisted with the ID '" +
                    invoiceModel.getId() + "' and will be delivered");
        } catch (Exception e) {
            // This happens when we can't persist the business document
            throw new Phase4Exception("Business document of AS4 message with message ID '" + messageState.getMessageID()
                    "' could not be persisted - cause: " + e.getMessage() + ". Please try again");
        }
    }
...
```

Listagem 33: SAFTPRO *Webservices* - Método *handle()* da classe ReceiverPeppolIncomingSBDHandlerServiceImpl.java

Para a prova de conceito, o objetivo foi atenuar o *Round Trip Time*[22] no pedido do emissor, pelo que a entrega de documentos é assíncrona. Ou seja, o módulo de *e-Invoicing* persiste a fatura e extrai toda a sua informação relevante e responde imediatamente ao *Sender Access Point*. Em simultâneo, é gerado um evento que despoleta a entrega do documento ao Consumidor. A partir deste instante, mesmo que o Consumidor esteja sobrecarregado ou incapaz de receber o documento, é improvável que a informação seja perdida, uma vez que já se encontra armazenada na base de dados.

Quando o sistema do Consumidor está ativo, o seu *endpoint* é invocado e o documento é entregue, através do `payloadURL` trocado no processo de registo. À semelhança do envio de faturas, o pedido é *Multipart* e composto pelo `businessDocument`, que corresponde à fatura em UBL 2.1, assim como o `invoiceSummary`, que disponibiliza de imediato alguma informação sobre a fatura em JSON. Adicionalmente, definiram-se alguns *headers* relevantes para determinar o sucesso de uma invocação pelo destinatário, que seguem as boas práticas utilizadas pelos *Webhooks* Github[23]:

- **X-Callback-Event:** evento que desencadeou a entrega do documento;
- **X-Callback-Delivery** UUID que permite identificar a entrega;
- **X-Callback-Signature:** HMacSHA256 dos *headers* descritos previamente, seguidos pelo corpo do pedido:
  - `signatureInput` = (X-Callback-Event + X-Callback-Delivery + `businessDocument`);
  - *key* = *secret* (trocado no momento de registo);
  - X-CallBack-Signature = HMacSHA256(*key*, `signatureInput`).

Em termos de fiabilidade, é utilizado um mecanismo de *Retry* com uma política de *Exponential backoff*[24], para lidar com falhas relacionadas com a entrega de documentos. Esta política opõe-se à linear e é mais eficiente, porque introduz pequenos *delays* variados entre as tentativas, o que permite ao sistema do Consumidor ter mais tempo para recuperar. Isto é essencial, porque o tempo para reenviar uma mensagem pode ser inferior ao tempo que o sistema do Consumidor leva a recuperar, pelo que aumentar simplesmente o número de tentativas, num intervalo fixo de tempo, significa gastar recursos da infraestrutura sem garantias de retorno. Para além disso, pode ter efeitos adversos no sistema do Consumidor, sobrecarregando-o e impedindo que estabilize para aceitar voltar a aceitar novos pedidos.

No Spring, o mecanismo de *Retry* pode ser estabelecido de uma maneira transparente através da anotação `@Retryable` (listagem 34), que apanha uma determinada exceção e despoleta uma nova tentativa. A configuração é efetuada através do número de tentativas, do *delay* inicial (expresso em milissegundos), do *multiplier* utilizado para diversificar os

[22] https://www.cloudflare.com/learning/cdn/glossary/round-trip-time-rtt/

[23] https://docs.github.com/en/developers/webhooks-and-events/webhooks/webhook-events-and-payloads

[24] https://docs.aws.amazon.com/general/latest/gr/api-retries.html

```java
package pt.opensoft.saftpro.service.einvoicing;

import ...

@Service
@Slf4j
public class WebhookService {

    private PeppolParticipantJpaRepository peppolParticipantJpaRepository;
    private RestTemplate restTemplate;
    ...
    @Retryable(value = {RestClientException.class}, maxAttempts = 5,
        backoff = @Backoff(delay = 1000, maxDelay = 1000000, multiplier = 1.2))
    public void invokeWebhook(InvoiceSummaryModel invoiceSummaryModel) {
        //Retrieve webhook from participantId
        PeppolParticipantEntity participant = peppolParticipantJpaRepository
                .findByParticipantId(invoiceSummaryModel.getCustomerIdentifier()).get();

        MultiValueMap<String, Object> body = new LinkedMultiValueMap<>();

        //Attachment
        ...
        body.set("businessDocument", attachmentPart);
        //Json
        ...
        body.set("invoiceSummary", jsonPart);

        HttpHeaders requestHeaders = new HttpHeaders();
        requestHeaders.setContentType(MediaType.MULTIPART_FORM_DATA);

        String webhookEventAsString = WebhookEventEnum.INVOICE_NOTIFICATION.getLabel();
        requestHeaders.add("X-Callback-Event", webhookEventAsString);

        String deliveryUUIDasString = UUID.randomUUID().toString();
        requestHeaders.add("X-Callback-Delivery", deliveryUUIDasString);

        try {
            InvoiceFileModel invoiceFile = invoiceSummaryModel.getInvoiceFile();
            byte[] webhookEventBytes = webhookEventAsString.getBytes();
            byte[] deliveryUUIDBytes = deliveryUUIDasString.getBytes();

            ByteBuffer signatureInput = ByteBuffer.allocate(webhookEventBytes.length + deliveryUUIDBytes.length +
                    invoiceFile.getFileSize().intValue());

            signatureInput.put(webhookEventBytes);
            signatureInput.put(deliveryUUIDBytes);
            signatureInput.put(invoiceFile.getContent());

            byte[] hmacSignature = HmacUtil.computeMac(signatureInput.array(), participant.getSecret().getBytes());

            requestHeaders.add("X-Callback-Signature", Base64.getEncoder().encodeToString(hmacSignature));
        } catch (InvalidKeyException e) {
            ...
        }

        HttpEntity<MultiValueMap<String, Object>> requestEntity = new HttpEntity<>(body, requestHeaders);

        ResponseEntity<Void> result = restTemplate.exchange(participant.getPayloadUrl(), HttpMethod.POST, requestEntity,

        log.info("Webhook for participant ID '" + invoiceSummaryModel.getCustomerIdentifier() +
            "' invoked. Received status code {}", result.getStatusCode());
    }

    @Recover
    public void invokeWebhookFallback(RestClientException e, InvoiceSummaryModel invoiceSummaryModel) {
       ...
    }
}
```

Listagem 34: Exemplo de `@Retryable` com *webhooks*

intervalos e do *delay* máximo que deverá ser considerado. Em complemento, define-se um método anotado com `@Recover`, que executa após o insucesso de todas as tentativas.

4.2.1.6.4 Concretização da aprovação/rejeição de uma fatura

Para que uma fatura, emitida por um prestador de bens ou serviços, seja utilizada para efeitos Fiscais deve ser aprovada pelo Consumidor. O módulo de comunicação disponibiliza um *endpoint* que despoleta o processo de atualização do estado de uma fatura, através de um pedido REST.

Quando a fatura é armazenada na base de dados, é-lhe atribuída o estado `PENDING`, que caracteriza todas as faturas para as quais ainda não se sabe o resultado de aprovação. Posteriormente, a fatura pode transitar para o estado `ACCEPTED` ou `REJECTED`, consoante o escrutínio do Consumidor. Se a fatura for rejeitada, o Consumidor deverá descrever o motivo que o levou a essa decisão e poderá abster-se do seu pagamento.

De acordo com a norma europeia EN16931, o número da fatura é utilizado para "*uniquely identify the Invoice within the business context, time-frame, operating systems and records of the Seller*", o que quer dizer que diferentes *Sellers* poderão emitir uma fatura completamente distinta com o mesmo número. Portanto, o *Controller* (listagem 35) responsável pela atualização do estado de uma fatura recebe o modelo JSON, que representa o novo estado para o qual a fatura deverá transitar, mais o identificador do *Seller* e o ano de emissão da fatura.

```
1  package pt.opensoft.saftpro.restcontroller.einvoicing;
2
3  import ...
4
5  @RestController
6  @RequestMapping("/rest")
7  public class InvoiceStatusController {
8
9      private InvoiceStatusService invoiceStatusService;
10     ...
11     @PutMapping(value = "/bearer/e-invoicing/invoices/{invoiceNumber}/status",
12         consumes = MediaType.APPLICATION_JSON_VALUE, produces = MediaType.APPLICATION_JSON_VALUE)
13     @MatchesReceiverParticipantId
14     public ResponseEntity<SaftproModel<InvoiceStatusModel>> updateStatus(
15         @PathVariable String invoiceNumber,
16         @RequestParam @DateTimeFormat(pattern = "yyyy-MM-dd") Date issueDate,
17         @RequestParam String sellerElectronicAddressSchemeId,
18         @RequestParam String sellerElectronicAddress,
19         @Parameter @RequestBody @Valid InvoiceStatusModel invoiceStatus) {
20
21         InvoiceStatusModel newInvoiceStatus = this.invoiceStatusService.updateStatus(invoiceNumber, issueDate,
22         sellerElectronicAddressSchemeId, sellerElectronicAddress, invoiceStatus);
23
24         SaftproModel<InvoiceStatusModel> result = new SaftproModel<>(newInvoiceStatus);
25
26         return ResponseEntity.ok(result);
27     }
28     ...
29 }
```

Listagem 35: SAFTPRO *Webservices* - InvoiceStatusController.java

4.2.2 Módulo de Validação e Integração

Este módulo tem a responsabilidade de assegurar a validação das faturas difundidas pelo módulo de comunicação, de acordo com as regras estabelecidas por um país e posteriormente armazená-las na base de dados. Deste modo, as Autoridades Tributárias poderão consultá-las através do SAFTPRO Portais e utilizar os seus detalhes para efeitos fiscais.

Ao nível da camada de acesso a dados e de acordo com as abstrações do Spring, o processo acima pode ser resumido na extração dos dados da fatura para objetos, que possam ser manipulados na camada de serviços e persistidos na base de dados, através de repositórios JPA. Para além disso, pressupõe-se que a solução é suficientemente flexível para minimizar as alterações ao produto, quando for necessário fazer a sua instalação noutro país. Como consequência desta premissa, o modelo de dados definido poderá sofrer alterações, mas a utilização da CIUS Peppol BIS Billing 3.0 dá-nos a garantia de que este nunca será mais abrangente na prova de conceito, o que permite simplificar um pouco a solução.

O modelo de validação proposto pelo UBL também desempenha um papel extremamente importante no desacoplamento do modelo de dados e do processo de validação, uma vez que no seu ponto de vista as regras expressas em ficheiros Schematron são aplicadas diretamente aos documentos em XML e este procedimento antecede a extração de informação para interpretação ao nível aplicacional.

4.2.2.1 Validação dos documentos

A ferramenta escolhida para validar as faturas é o PHIVE, que através de conjuntos de ficheiros XSD e/ou Schematron compõe um motor genérico de validação. Na ferramenta os conjuntos de diferentes artefactos de validação designam-se `ValidationExecutorSet` e são registados na classe `ValidationExecutorSetRegistry`, para serem utilizados em tempo de execução.

O mecanismo de validação em UBL separa a validação sintática e semântica dos documentos. Estas distinção pode ser implementada de forma transparente na biblioteca, ao registar todos os artefactos de validação sintática antes dos de validação semântica, uma vez que estes são aplicados ao documento pela ordem de registo. Em relação à identificação dos diferentes `ValidationExecutorSet`, o PHIVE utiliza um mecanismo muito semelhante às coordenadas Maven[25] e a ferramenta disponibiliza alguns exemplos, inclusive da CIUS Portuguesa definida pela eSPap[26].

4.2.2.1.1 Concretização da geração dos artefactos de validação

[25] https://maven.apache.org/pom.html#Maven_Coordinates
[26] https://www.espap.gov.pt/Paginas/home.aspx

Para gerar os artefactos de validação é utilizado o *plugin* Maven `ph-schematron-maven-plugin`, que executa durante a fase `generate-resources` e transforma os ficheiros Schematron em ficheiros XSLT.

O processo de geração foi priorizado em *build-time*, porque as transformações podem levar minutos, mesmo para ficheiros com tamanho médio. Contudo, este processo necessita de ocorrer uma única vez, pelo que os ficheiros compilados podem ser *cached* e utilizados em todas as validações subsequentes.

```
...
    <plugin>
        <groupId>com.helger.maven</groupId>
        <artifactId>ph-schematron-maven-plugin</artifactId>
        <version>x.y.z</version>
        <executions>
            <execution>
                <id>peppol-bis-billing-3.0</id>
                <goals>
                    <goal>convert</goal>
                </goals>
                <configuration>
                    <schematronDirectory>src/test/resources/rule-source/openpeppol</schematronDirectory>
                    <schematronPattern>*.sch</schematronPattern>
                    <xsltDirectory>src/main/resources/validation-artifacts/openpeppol</xsltDirectory>
                </configuration>
            </execution>
        </executions>
    </plugin>
...
```

Listagem 36: Configuração para as regras de validação do Peppol BIS *Billing* 3.0

O troço de código acima exemplifica a configuração do processo de geração dos ficheiros XSLT para as regras de validação do PEPPOL BIS *Billing* 3.0. A fase de execução pode ser customizada[27] e os parâmetros mais importantes são:

- **`schematronDirectory`:** diretoria onde residem os ficheiros Schematron;
- **`schematronPattern`:** padrão que identifica os ficheiros Schematron a ser transformados;
- **`xsltDirectory`:** diretoria onde serão armazenados os ficheiros XSLT resultantes.

Como boa prática, os ficheiros a transformar residem na diretoria `src/test/resources/rule-source` e os ficheiros gerados na diretoria `src/main/resources/validation-artifacts`.

4.2.2.1.2 Concretização do registo dos artefactos de validação

O identificador utilizado para os conjuntos de validação é o *Validation Executor Set* ID (VESID). Este identificador é composto pela concatenação dos elementos `groupID`, `artifactID` e `version`.

[27] https://github.com/phax/ph-schematron/wiki/Maven-plugin

```java
package pt.opensoft.saftpro.validator.einvoicing.sbd;

import ...

@Component
public class SBDValidator {

    ValidationExecutorSetRegistry<IValidationSourceXML> registry;
    ...
    public void validate(Node xml, ValidationResultHandler validationResultHandler)
                                        throws IllegalArgumentException, IllegalStateException {
        if (xml == null) {
            throw new IllegalArgumentException("The XML element can not be null!");
        }

        IValidationExecutorSet<IValidationSourceXML> validationExecutorSet = this.getDefaultValidationExecutorSet();

        if (validationExecutorSet == null) {
            throw new IllegalStateException("The default validation executor set is unknown!");
        }

        ValidationResultList validationResult = ValidationExecutionManager.executeValidation(validationExecutorSet,
                ValidationSourceXML.create(null, xml));

        if (validationResult.containsAtLeastOneError()) {
            validationResultHandler.onValidationErrors(validationResult);
        } else {
            validationResultHandler.onValidationSuccess(validationResult);
        }
    }

    public void validate(Node xml, ValidationResultHandler validationResultHandler, VESID id)
                                        throws IllegalArgumentException, IllegalStateException {
        if (xml == null) {
            throw new IllegalArgumentException("The XML element can not be null!");
        }

        if (id == null) {
            throw new IllegalArgumentException("The validation executor set ID can not be null!");
        }
        ...
    }

    public IValidationExecutorSet<IValidationSourceXML> getDefaultValidationExecutorSet() {
        return registry.getOfID(PeppolValidation3_11_1.VID_OPENPEPPOL_INVOICE_V3);
    }

    public IValidationExecutorSet<IValidationSourceXML> getValidationExecutorSet(VESID id) {
        return this.registry.getOfID(id);
    }

    public void addValidationExecutorSet(IValidationExecutorSet<IValidationSourceXML> validationExecutorSet) {
        IValidationExecutorSet<IValidationSourceXML> oldValidationExecutorSet =
                this.getValidationExecutorSet(validationExecutorSet.getID());
        if (oldValidationExecutorSet == null) {
            this.registry.registerValidationExecutorSet(validationExecutorSet);
        }
    }

    public void removeValidationExecutorSet(VESID id) {
        this.registry.unregisterValidationExecutorSet(id);
    }
}
```

Listagem 37: SAFTPRO *Webservices* - SBDValidator.java

Para o registo dos artefactos foi desenvolvida a classe `SBDValidator` (listagem 37), que funciona como um *wrapper* da `ValidationExecutorSetRegistry` e disponibiliza as operações de validação.

O PHIVE disponibiliza dois tipos de *validation executors*, são eles:

- **`ValidationExecutorXSD`:** utilizado preferencialmente para artefactos de validação sintática;
- **`ValidationExecutorSchematron`:** utilizado para artefactos de validação semântica.

Após a criação dos *validation executors*, o processo de registo pode ser concluído através da invocação do método `addValidationExecutorSet()`, presente na classe `SBDValidator`.

```
...
    @PostConstruct
    private void loadDefaultValidationExecutorSet() {
        registry.registerValidationExecutorSet(ValidationExecutorSet.create(
                PeppolValidation3_11_1.VID_OPENPEPPOL_INVOICE_V3,
                "OpenPEPPOL Invoice",
                false,
                ValidationExecutorXSD.create(EUBL21DocumentType.INVOICE),
                ValidationExecutorSchematron.createXSLT(
                    PeppolValidation3_11_1.INVOICE_CEN,
                    UBL21NamespaceContext.getInstance()),
                ValidationExecutorSchematron.createXSLT(
                    PeppolValidation3_11_1.INVOICE_PEPPOL,
                    UBL21NamespaceContext.getInstance())
                // ValidationExecutorSchematron.createXSLT(COUNTRY_SPECIFIC_XSLT, UBL21NamespaceContext.getInstance())
            )
        );
    }
...
```

Listagem 38: Registo de um ValidationExecutorSet no SBDValidator

A listagem 38 exemplifica o carregamento dos artefactos do Peppol, utilizados por omissão no processo de validação. Para o modelo do UBL ser respeitado, regista-se primeiro as restrições XSD correspondentes ao formato UBL 2.1. De seguida, as regras do modelo semântico da norma europeia EN16931 e por último as do Peppol. Relativamente aos ficheiros Schematron, devem registar-se primeiro as regras menos restritas e só depois as mais específicas.

4.2.2.1.3 Concretização da fase de validação

O `SBDValidator` oferece duas variantes do método `validate()`, que podem ser utilizadas para validar as faturas. Uma delas recebe o VESID do `ValidationExecutorSet` que deve ser utilizado e quando este é omitido o `ValidationExecutorManager` da ferramenta utiliza os artefactos base do Peppol (listagem 39).

```
...
    @Test
    public void testValidateWithDefaultValidationExecutorSet() throws Exception {
        IValidationExecutorSet<IValidationSourceXML> defaultValidationExecutorSet =
                this.sbdValidator.getDefaultValidationExecutorSet();

        Assertions.assertNotNull(defaultValidationExecutorSet);

        FileSystemResource validXmlFile = new FileSystemResource("src/test/resources/examples/einvoicing/example1.xml");
        Document validDocument = domParser.parseXml(new FileInputStream(validXmlFile.getFile()));

        assertThrows(SBDValidationException.class,
                     () -> sbdValidator.validate(invalidDocument1, new BusinessDocumentValidationResultHandler()));
    }
...
```

Listagem 39: Exemplo de validação com os artefactos do Peppol

4.2.2.2 Infraestrutura de armazenamento

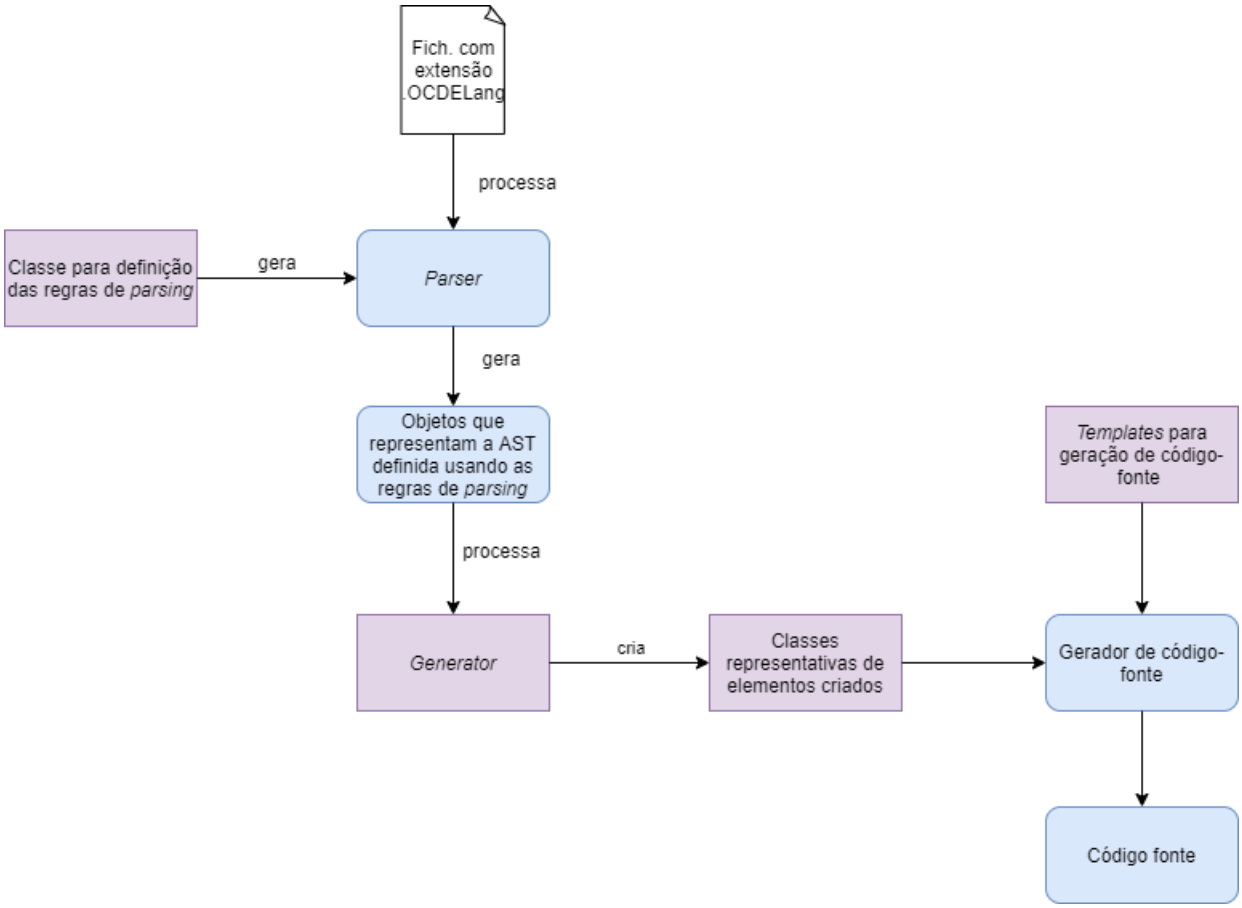


Figura 4.11: OCDE *Lang* - Arquitetura

A solução considerada para a infraestrutura que suporta a consulta e persistência de faturas assenta numa versão inicial da ferramenta de mapeamento desenvolvida pelo colega Pedro Silva, no âmbito da sua tese de mestrado "*In-flight Big Data Validation and*

Integration". A ferramenta tem o nome OCDE *Lang* e foi concebida para, através de uma DSL, mapear ficheiros SAF-T em relações e entidades que possibilitam a geração de código através de templates Xtend. Não obstante, a sobreposição dos objetivos em ambas as teses abriu a possibilidade de explorar o potencial de ferramenta e reutilizá-la para dar a origem à extensão que suporta ficheiros UBL 2.1.

A figura 4.11 ilustra os vários componentes da ferramenta e estabelece, sobre a forma de relação, as interações realizadas entre eles. A roxo estão marcados os elementos definidos manualmente e a azul todos os componentes gerados internamente pela ferramenta Xtext, sobre os quais não existe controlo direto. Internamente a OCDE *Lang* divide-se em quatro componentes principais, sendo estes:

- O ficheiro que define a gramática da linguagem, isto é, onde são definidas as instruções disponibilizadas ao utilizador final da ferramenta;
- O *parser* responsável por receber os objetos representativos das expressões definidas pelo utilizador e construir os objetos utilizados internamente durante a fase de geração de código;
- Os diversos objetos que representam o modelo definido pelo utilizador, tais como elementos, campos, relações, entre outros;
- Os *templates* que utilizam as estruturas criadas internamente para gerar o código Java final.

```
1  ...
2      create element VatBreakdown {
3          field create TaxableAmount tag "cbc:TaxableAmount" float NotNull;
4          field create TaxAmount tag "cbc:TaxAmount" float NotNull;
5          field create VatCategoryCode tag "cac:TaxCategory/cac:TaxScheme[cbc:ID='VAT']/../cbc:ID" string NotNull;
6          field create VatRate tag "cac:TaxCategory/cac:TaxScheme[cbc:ID='VAT']/../cbc:Percent" float NotNull;
7          field create ExemptionReasonCode
8                  tag "cac:TaxCategory/cac:TaxScheme[cbc:ID='VAT']/../cbc:TaxExemptionReasonCode" string;
9          field create ExemptionReason tag "cac:TaxCategory/cac:TaxScheme[cbc:ID='VAT']/../cbc:TaxExemptionReason" string;
10     };
11
12     create element CreditTransfer {
13         field create PaymentAccountId tag "cbc:ID" string NotNull;
14         field create PaymentAccountName tag "cbc:Name" string;
15         field create PaymentServiceProviderId tag "cac:FinancialInstitutionBranch/cbc:ID" string;
16     };
17
18     create element PaymentInstruction {
19         field create PaymentMeansCode tag "cbc:PaymentMeansCode" string NotNull;
20         field create PaymentMeansText tag "cbc:PaymentMeansCode/@name" string NotNull;
21         field create RemittanceInformation tag "cbc:PaymentID" string NotNull;
22         link create child CardInformation tag "cac:CardAccount" OneToOne;
23         link create child CreditTransfer tag "cac:PayeeFinancialAccount" OneToOne;
24         link create child DirectDebit tag "cac:PaymentMandate" OneToOne;
25     };
26 ...
```

Listagem 40: OCDE *Lang* - Exemplo de sintaxe da extensão UBL 2.1

Muito brevemente, a extensão UBL 2.1 acrescenta a *keyword* tag que contêm a expressão XPath relativa do elemento, através da qual é possível criar *templates* que interrogam

os nós do documento na fase de processamento para extrair os valores dos campos considerados relevantes. Para além disso, adiciona os tipos básicos `boolean` e `file` que são traduzidos, respetivamente, para os tipos `Boolean` e `byte[]` do Java. A listagem 40 destaca os aspetos mais importantes da DSL:

- **Linha 2:** permite criar uma entidade com o nome `VatbreakDown`. Na OCDE *Lang*, o nome da entidade pode corresponder ao *path* do elemento no ficheiro XML, mas na extensão UBL não deve ser utilizado porque os elementos podem pertencer a *namespaces* diferentes;
- **Linha 21:** permite criar um campo obrigatório do tipo `String`. Na versão base da OCDE *Lang*, a *keyword* `tag` não é obrigatória, mas na extensão UBL é essencial porque guarda a expressão XPath referente ao valor do campo. Por último, a *keyword* `NotNull` é opcional e indica se o campo é obrigatório;
- **Linha 22:** permite criar uma relação de um para um entre as entidades `PaymentInstruction` e `CardInformation`. A *keyword* `child` pode ser substituída por `father`, indicando quem controla a relação. Semelhante à criação de novos campos, existe a possibilidade de indicar se a relação é obrigatória através da *keyword* `NotNull`. Na criação de relações também é necessário indicar a `tag` referente ao nó do documento que representa o elemento da relação, tornando possível a utilização das expressões XPath de modo composicional.

Na OCDE *Lang* existem outro tipo de operações, como por exemplo a renomeação de elementos, mas as descritas acima são as essenciais para o propósito do módulo de *e-Invoicing*.

Para terminar, os tipos de dados que existem são:

- `string`;
- `int`;
- `date`;
- `float`;
- `boolean`;
- `file`.

E as relações podem ter cardinalidade:

- `OneToOne`;
- `OneToMany`;
- `ManyToOne`;
- `ManyToMany`;

4.2.2.2.1 Concretização da geração da infraestrutura de armazenamento

A base começa com a definição do modelo de dados de acordo com a norma europeia EN16931, tendo em conta a estrutura do Peppol. Para tal, foi necessário efetuar o mapeamento do modelo semântico para a CIUS Peppol BIS *Billing* 3.0 e definir as tabelas da base de dados (fig. 4.12).

Norma EN16931				PEPPOL BILLING BIS 3.0		BD e-Invoicing
ID	Nível	Cardinalidade	Business Term	Campo	Cardinalidade	
BT-1	+	1..1	Invoice number	Invoice/cbc:ID	1..1	INVOICE.INVOICE_NUMBER
BT-2	+	1..1	Invoice issue date	Invoice/cbc:IssueDate	1..1	INVOICE.ISSUE_DATE
BT-9	+	0..1	Payment due date	Invoice/cbc:DueDate	0..1	INVOICE.PAYMENT_DUE_DATE
BT-3	+	1..1	Invoice type code	Invoice/cbc:InvoiceTypeCode	1..1	INVOICE.INVOICE_TYPE_CODE
BG-1	+	0..n	INVOICE NOTE			INVOICE_NOTE
BT-22	++	1..1	Invoice note	Invoice/cbc:Note	0..1	INVOICE_NOTE.NOTE
BT-7	+	0..1	Value added tax point date	Invoice/cbc:TaxPointDate	0..1	INVOICE.TAX_POINT_DATE
BT-5	+	1..1	Invoice currency code	Invoice/cbc:DocumentCurrencyCode	1..1	INVOICE.INVOICE_CURRENCY_CODE
BT-6	+	0..1	VAT accounting currency code	Invoice/cbc:TaxCurrencyCode	0..1	INVOICE.VAT_ACCOUNTING_CURRENCY_CODE
BT-19	+	0..1	Buyer accounting reference	Invoice/cbc:AccountingCost	0..1	INVOICE.BUYER_ACCOUNTING_REFERENCE

Figura 4.12: Modelo de Dados - Mapeamento do modelo semântico da norma EN16931

Depois, o diagrama ER é construído na ferramenta DeZign para tirar proveito da funcionalidade que permite gerar o *script* DDL para a criação da base de dados (fig. 4.13). Em simultâneo, é necessário traduzir o diagrama para a extensão da DSL, como demonstrado nos elementos `VatBreakdown`, `CreditTransfer` e `PaymentInstruction`.

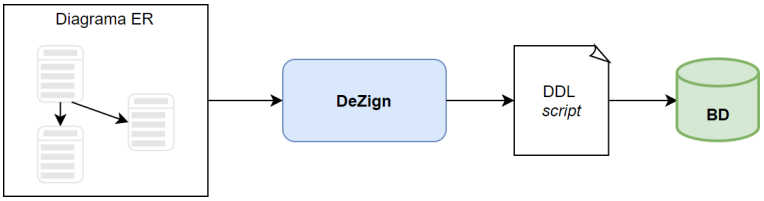


Figura 4.13: DeZign - Geração da base de dados

O passo seguinte consiste em definir os *templates* que trabalham as classes representativas dos elementos criados:

- `DOMParser`;
- `UBLNamespaceContextConfig`;
- `XPathExtractor`;
- `ElementModelExtractorTemplate`;
- `ModelClassTemplate`;
- `MapperClassTemplate`;
- `EntityClassTemplate`;
- `JpaRepositoryClassTemplate`;

Os primeiros quatro *templates* geram classes as que permitem fazer a extração da informação. O `DOMParser` processa o conteúdo em *bytes* de um ficheiro XML e constrói a

sua representação em memória. O `XPathExtractor` é configurado com a classe `UBLNamespaceContextConfig` e interroga a representação das faturas em memória para extrair a sua informação, consoante o tipo do campo declarado na DSL. Por último, o `ElementModelExtractorTemplate` gera as classes que utilizam o `XPathExtractor` para construir o modelos dos elementos da fatura definidos. Os três primeiros *templates* são estáticos, uma vez que não dependem diretamente do modelo de dados. No entanto, foram incluídos neste módulo porque são a base para transformar a informação na representação adequada para ser armazenada.

Os últimos prendem-se mais com a persistência da informação. O `ModelClassTemplate` gera a representação dos elementos utilizada na camada de serviços e o `EntityClassTemplate` gera as *entities*, para as quais devem ser criados os repositórios JPA. Para concluir, o `MapperClassTemplate` origina as classes que permitem converter os modelos nas *entities* utilizadas na camada de persistência e vice-versa.

A título de exemplo, segue em anexo pequenos excertos do `ElementModelExtractorTemplate` (listagem 48 e 49), `EntityClassTemplate` (listagem 50 e 51) e `UBLNamespaceContextConfig` (listagem 52 e 53). Após a criação dos *templates* e do modelo base, para efetuar alterações basta redefinir os elementos na ferramenta de mapeamento e voltar a gerar as classes que constituem a camada de persistência. Assim, é possível reconfigurar o modelo de dados[28], sem que o módulo de comunicação tenha essa percepção e as validações também não são afetadas, visto que os mecanismos estão desacoplados.

Alternativas

Relativamente à infraestrutura de armazenamento, foi explorada a ferramenta Telosys para gerar o código de suporte para a extração e persistência das faturas, uma vez que existem casos de utilização internos na Opensoft. Para tal, bastava aceder à base de dados no Telosys través de uma conexão JDBC e extrair a informação necessária. No entanto, constatou-se que existe uma grande limitação no modelo de dados. Ou seja, gerar os repositórios e entidades JPA é uma tarefa simples e requer apenas algumas modificações nas cardinalidades das relações, mas a parte da extração da informação é mais complicada, visto que assenta nas expressões XPath definidas na fase de mapeamento da norma EN16931 para o Peppol BIS *Billing* 3.0. Ou seja, é praticamente impossível expressar trivialmente no modelo de dados estas expressões ou inferi-las, o que iria limitar a generalidade da solução no que diz respeito ao código de extração, que teria de ser desenvolvido à medida e sofreria alterações sempre que fosse necessários acrescentar ou remover elementos da fatura.

[28] Consultar anexo XI para visualizar alguns exemplos de manipulação do modelo de dados.

4.3 SAFTPRO Portais

O SAFTPRO foi concebido a pensar nas Autoridades Tributárias proporcionando uma visão integrada das várias transações, que ocorrem numa determinada economia, para lhes permitir definir medidas mais eficazes de combate à fraude e evasão fiscal.

A solução é modular e composta por várias funcionalidades que visam simplificar não só o cumprimento das obrigações por parte dos utilizadores finais da solução, como também agilizar o controlo e gestão deste processo pelas várias Autoridades Tributárias. Consequentemente, o enriquecimento do SAFTPRO ao nível dos seus *webservices* criou a necessidade de desenvolver uma nova página no *backoffice* da aplicação, para que os funcionários das Autoridades Tributárias tenham acesso aos documentos comunicados através do Peppol.

4.3.1 Instâncias Liferay

O Liferay é uma solução integrada que disponibiliza uma série de funcionalidades de suporte ao desenvolvimento de portais *web*, nomeadamente um módulo de autenticação com gestão de utilizadores, um módulo de gestão documental e um gestor de conteúdos *web*. Simultaneamente, o Liferay permite integrar as suas soluções com sistemas de autenticação externos, configurar repositórios de documentos na *Cloud*, fazer o *deploy* das soluções em *containers docker* e pode ser integrado com inúmeras bases de dados, como por exemplo Oracle, MySQL, H2, etc.

O SAFTPRO Portais é representado por duas instâncias virtuais do Liferay:

- **Saftpro (saftpro.com)**: composto pelos *sites*:
  - ***Consumer* (Consumidor):** que integra as páginas privadas definidas para os utilizadores da aplicação registados como consumidores. Estas páginas disponibilizam, entre outras, as funcionalidades de registo e atualização de faturas submetidas em ficheiros SAF-T;
  - ***Merchant* (Comerciante):** integra os comerciantes que poderão aceder às páginas privadas aqui definidas. Estas páginas disponibilizam, por exemplo, as funcionalidades de submissão e consulta de ficheiros SAF-T;
  - ***Public*:** disponibiliza as páginas públicas acessíveis aos *guests* da aplicação, nomeadamente a *homepage* do portal e a típica página de contactos, etc;
- ***Backoffice* (saftpro-backoffice.com):** composto por um único que site que oferece páginas privadas para a configuração de parâmetros da aplicação, como por exemplo a língua, o tamanho máximo permitido para *upload*, largura de banda disponível, etc.

Ambas as instâncias possuem o *site Global*, gerado na criação de qualquer instância virtual do Liferay, com a particularidade de não ser possível definir-lhe quaisquer tipo de páginas, assim como o facto de não poder ser desativado ou removido. No *deploy* da

solução, os *sites Consumer* e *Merchant* podem ser desativados, caso esse seja o objetivo, uma vez que o cliente alvo poderá não estar interessado em ter disponível os módulos relativos ao consumidor e/ou ao comerciante. É isto que acontece na prova de conceito, uma vez que as funcionalidades disponibilizadas aos Agentes Económicos são ilustradas pela mini-aplicação SAFTPRO *Desktop Application*.

O desenvolvimento de novas funcionalidades ao nível do SAFTPRO Portais incide na instância *Backoffice*[29] e assenta numa arquitetura de *Portlets* que permite adicionar, assim como remover conteúdos *web* ou funcionalidades a uma determinada página. Em Java, as *portlets* estão definidas pelas especificações JSR 168 e JSR 286[30]. Segundo estas, as *portlets* são caracterizadas como um componente *web* reutilizável gerado por um *container* específico que processa um pedido e gera dinamicamente o conteúdo que será mostrado ao utilizador.

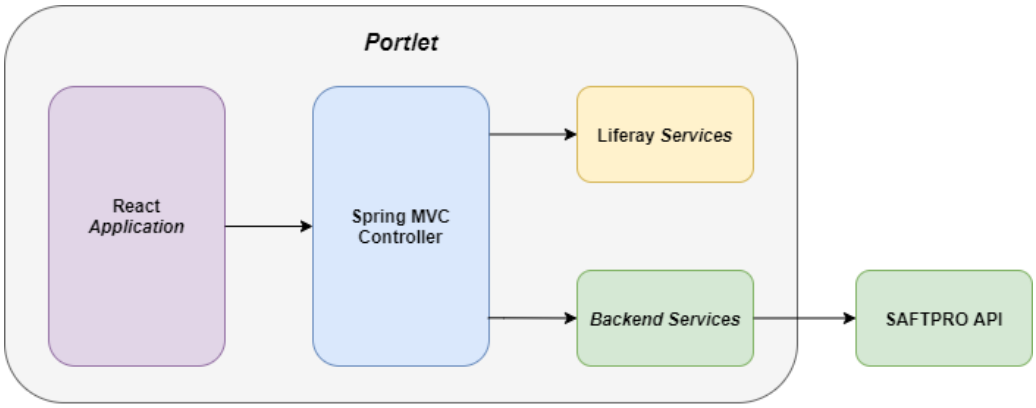


Figura 4.14: SAFTPRO Portais - Visão geral de uma *portlet*

O Liferay pode ser integrado com Spring, pelo que é comum utilizar o componente Spring MVC *Portlet*[31] para conceber novas funcionalidades nas aplicações *web*. Em relação ao desenvolvimento das páginas HTML, pode recorrer-se a qualquer biblioteca ou *framework*, mas o React[32] é predominante.

4.4 SAFTPRO *Desktop Application*

No seguimento da primeira fase de desenvolvimento, surgiu a necessidade de construir uma pequena aplicação capaz de demonstrar um exemplo de integração com as funcionalidades disponibilizadas pelo módulo de *e-Invoicing*.

O utilizador alvo são os agentes económicos e a aplicação foi desenvolvida em Flutter, que de momento está a adicionar suporte para *Native Desktop Applications*.

[29] Consultar anexo XII para mais detalhes sobre as páginas de consulta implementadas.

[30] https://www.oracle.com/java/technologies/intro-java-portlet-specifications.html

[31] https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/portlet.html

[32] https://reactjs.org/

4.4.1 Flutter

O Flutter é uma *framework* que permite desenvolver UIs modernas, compiladas nativamente a partir de uma única base de código. A lógica aplicacional baseia-se em programação reativa, mas o Flutter é multi-paradigma e existem outras técnicas que se tornam relevantes, como por exemplo a programação *event-driven* na codificação das interações entre os utilizadores e o estado dos componentes da aplicação.

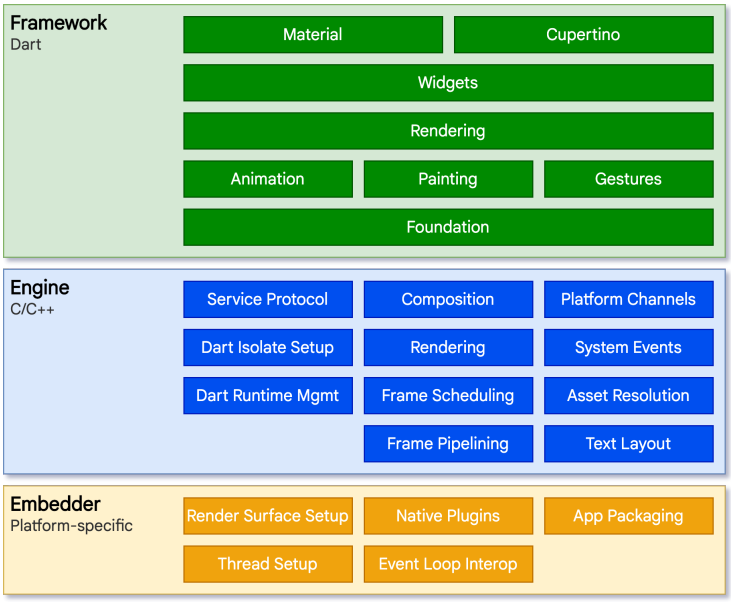


Figura 4.15: Flutter - Arquitetura [20]

Um dos conceitos arquiteturais mais importante na *framework*, é a sua categorização em termos de complexidade e respetiva disposição em camadas de complexidade decrescente.

Tendo em conta a figura 4.15, a camada superior possui os *Widgets* nativos do Flutter. A camada imediatamente a seguir suporta as primitivas necessárias para as aplicações, nomeadamente o componente de renderização de baixo nível e a última camada desce ao código específico das plataformas.

No Flutter, o *building block* básico é o *Widget* e este pode ir desde um simples botão ou campo de texto a um formulário, ecrã *pop-up*, entre outros. Os *Widgets* são reutilizáveis através de uma abordagem composicional, uma vez que os mais complexos são compostos tipicamente por *Widgets* mais pequenos e centralizados, que produzem um efeito mais significativo quando são combinados. Neste sentido, o número de conceitos é reduzido e a hierarquia de classes é deliberadamente superficial e ampla para maximizar o número possível de combinações, reforçar a modularidade dos componentes e permitir que estes se foquem num único propósito. Por exemplo, até as funcionalidades básicas de alinhamento e *padding* são implementadas em *Widgets* separados, em vez de serem incorporadas no *core* da *framework*. Ou seja, para alinhar um *Widget* ao centro utiliza-se o *Widget Center* e não uma propriedade inerente do *Widget* que se quer alinhar.

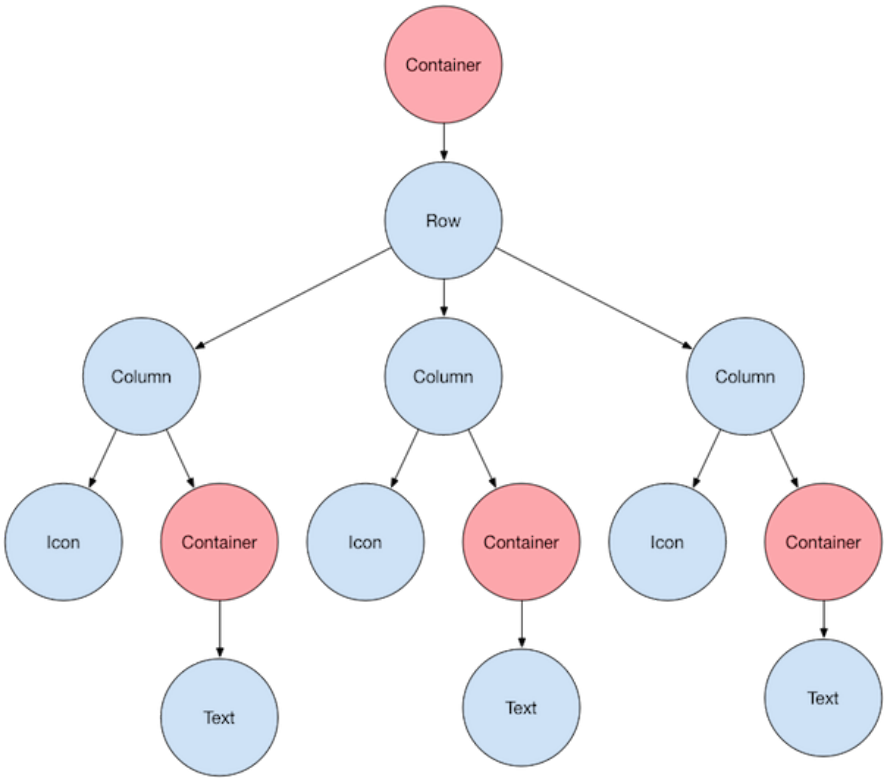


Figura 4.16: Flutter - *Widget Tree*[21]

Para terminar, a noção de composição dá origem ao termo *Widget Tree* utilizado para descrever os componentes que constituem a interface do utilizador e os *Widgets* podem ser *stateless* ou *stateful*, embora a sua diferença resida exclusivamente na noção de estado. Isto é, ambos os tipos de *Widget* agem da mesma maneira, mas o *stateful* possui adicionalmente o objeto estado, constituído por propriedades mutáveis que moldam e refletem a interação dos utilizadores.

Relativamente às alterações no estado dos componentes, os *Widgets* são renderizados justamente com as alterações necessárias, através da comparação com o estado antigo, o que permite afinar o desempenho das aplicações.

4.4.2 Arquitetura

Em Flutter, o *State Management* é um tópico notoriamente descartado porque a *framework* recomenda diferentes abordagens[33], mas não existe consenso sobre qual deve ser adotada, uma vez que esta pode ser influenciada pelo tipo e tamanho da aplicação. Para a SAFTPRO *Desktop Application* está a ser utilizada a noção de *Provider*[34], com as convenções definidas no artigo "Flutter *Architecture - My Provider Implementation Guide*"[32], que se resumem nos seguintes pontos:

[33] https://flutter.dev/docs/development/data-and-backend/state-mgmt/options
[34] https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple

- Cada ecrã possui um modelo próprio que deriva da classe `ChangeNotifier`;
- Os *Listeners* (ou *Consumers*) de um modelo apenas são invocados quando o seu estado é alterado;
- Os ecrãs possuem apenas dois estados, *Idle* ou *Busy*, pelo que qualquer outro componente que o constitua e necessite de estado próprio terá o seu modelo específico, para que o ecrã principal só seja atualizado quando o seu estado global é alterado;
- Os serviços e *Providers* são injetados através de um *Service Locator*, para separar a lógica de negócio dos `Widgets` que constituem a interface gráfica do utilizador;
- Os modelos dos ecrãs obtêm dados exclusivamente através dos serviços que necessitam, para uniformizar a serialização da informação obtida.

4.4.2.1 Concretização da Aplicação

A nível interno, a aplicação encontra-se divida em duas diretorias:

- ***Core*:** composto pelos ficheiros associados à lógica da aplicação. Ou seja, os modelos utilizados para serializar a informação obtida da API, os modelos utilizados pelos ecrãs e os Serviços que tratam efetivamente da lógica de negócio.
- **UI:** composta pelos ficheiros direcionados para a implementação da interface gráfica. Nomeadamente, os ficheiros das *Views* (ou ecrãs) da aplicação, assim como todos os *Widgets* personalizados desenvolvidos.

Naturalmente, o conteúdo de ambas as diretorias não se restringe na totalidade ao que está mencionado acima e existem outros componentes, como por exemplo o `router.dart` (listagem 41) responsável por tratar da navegação entre ecrãs.

```
1  import '...';
2
3  class Router {
4    static const initialRoute = InitialPage.routeName;
5
6    static Route<dynamic> generateRoute(RouteSettings settings) {
7      switch (settings.name) {
8        case InitialPage.routeName: return MaterialPageRoute(builder: (_) => InitialPage());
9        case LoginPage.routeName: return MaterialPageRoute(builder: (_) => LoginPage());
10       case InitialSetupPage.routeName: return MaterialPageRoute(builder: (_) => InitialSetupPage());
11       case HomePage.routeName: return MaterialPageRoute(builder: (_) => HomePage());
12       case SettingsPage.routeName: return MaterialPageRoute(builder: (_) => SettingsPage());
13       case SendInvoicePage.routeName: return MaterialPageRoute(builder: (_) => SendInvoicePage());
14       case MyInvoicesPage.routeName: return MaterialPageRoute(builder: (_) => MyInvoicesPage());
15       case InvoiceDetailsPage.routeName:
16         var invoiceDetailsArguments = settings.arguments as InvoiceDetailsArguments;
17
18         return MaterialPageRoute(
19             builder: (_) => InvoiceDetailsPage(invoiceSummary: invoiceDetailsArguments.invoiceSummary)
20             );
21       default: ...
22     }
23   }
24 }
```

Listagem 41: SAFTPRO *Desktop Application* - router.dart

Comecemos pelo `BaseModel` (listagem 42), cuja responsabilidade é influenciar o *layout* dos ecrãs através de notificações, sempre que existem alterações de estado. Para além disso, este componente uniformiza as mensagens de erro disponibilizadas ao utilizador e é derivado por todos os modelos utilizados pelos ecrãs da aplicação.

```
import '...';

class BaseModel extends ChangeNotifier {
  ViewState _state = ViewState.idle;
  String _errorMessage = '';
  Map<String, String> _fieldErrors = Map();

  ViewState get state => _state;

  void setState(ViewState viewState) {
    _state = viewState;
    notifyListeners();
  }

  bool hasErrors() {return _errorMessage.isNotEmpty || _fieldErrors.isNotEmpty;}

  void handleOnError(Object? exception) {...}

  void handleOnSuccess() {...}
}
```

Listagem 42: SAFTPRO *Desktop Application* - base_model.dart

```
import '...';

class BaseView<T extends BaseModel> extends StatefulWidget {
  final Widget Function(BuildContext context, T model, Widget? child) builder;
  final Function(T)? onModelReady;

  BaseView({required this.builder, this.onModelReady});

  _BaseViewState<T> createState() => _BaseViewState<T>();
}

class _BaseViewState<T extends BaseModel> extends State<BaseView<T>> {
  T model = locator<T>();

  void initState() {
    if (widget.onModelReady != null) {
      widget.onModelReady!(model);
    }
    super.initState();
  }

  Widget build(BuildContext context) {
    return ChangeNotifierProvider<T>(create: (context) => model, child: Consumer<T>(builder: widget.builder));
  }
}
```

Listagem 43: SAFTPRO *Desktop Application* - base_view.dart

Em simultâneo, foi definido o *Widget* `BaseView` (listagem 43) que centraliza o *setup* do `ChangeNotifierProvider` e o respetivo *Consumer*. Deste modo, sempre que for necessário desenvolver ecrãs que possuam um modelo associado, basta criar o respetivo modelo (estendendo o `BaseModel`) e retornar a `BaseView` no método `build()` do ecrã. Assim,

o *Widget* que representa o ecrã será sempre notificado quando o estado do modelo é alterado, através da invocação do método `setState()` definido no `BaseModel`.

Para simplificar os *stateful Widgets*, a `BaseView` é convertida ela própria num *stateful Widget* para armazenar localmente o modelo do ecrã e recebe pelo construtor a função do *Widget* "pai" que deverá ser invocada na sua inicialização para popular o modelo do ecrã.

```
1  import '...';
2
3  class MyInvoicesPageModel extends BaseModel {
4    final InvoiceService _invoiceService = locator<InvoiceService>();
5    late InvoiceSummaryModelList _invoiceSummaryModelList;
6
7    Future<void> fetchInvoices() async {
8      setState(ViewState.busy);
9
10     final User user = locator<User>();
11
12     if (user.userType == UserType.customer) {
13       final CustomerAccessPointSettings accessPointSettings = locator<CustomerAccessPointSettings>();
14
15       try {
16         _invoiceSummaryModelList = await _invoiceService.fetchInvoices(accessPointSettings, user);
17
18         handleOnSuccess();
19       } catch (e) {
20         handleOnError(e);
21       }
22     }
23
24     setState(ViewState.idle);
25   }
26
27   InvoiceSummaryModelList get invoiceSummaryModelList => _invoiceSummaryModelList;
28 }
```

Listagem 44: SAFTPRO *Desktop Application* - my_invoices_page_model.dart

Tomemos como exemplo o ecrã `MyInvoicesPage`, utilizado para aceder às faturas recebidas pelo módulo de *e-Invoicing*, visto que todos os ecrãs desenvolvidos seguem a mesma estrutura.

O primeiro passo consiste em criar o modelo `MyInvoicesPageModel` (listagem 44), relembrando que este é o responsável por solicitar o serviço necessário para ir buscar os dados utilizados pelo ecrã. Para tal, este componente precisa de uma referência do `InvoiceService` (injetada pelo *locator*) e expõe uma lista de *invoices*. A função `fetchInvoices()` é assíncrona e despoleta a recolha da informação, pelo que a invocação do serviço é envolvida no método `setState()` para atualizar o estado da *View*. Ou seja, enquanto está a decorrer a recolha de informação o estado da *View* é *Busy* e quando o resultado está disponível a *View* retorna ao estado *Idle*. Deste modo, o *Widget* poderá agir em conformidade e mostrar ao utilizador dados relativos ao progresso de obtenção das faturas e posteriormente ajustar o *layout* para disponibilizar as faturas obtidas.

Depois, é necessário registá-lo no *locator* como *Factory*, para ficar acessível nos *Widgets* e ser criada uma nova instância sempre que for requisitado. Em contrapartida, os serviços são registados como *Singleton* e são partilhados por todos os *Widgets*.

```
import '...';

GetIt locator = GetIt.instance;

void _registerPageModels() {
  locator.registerFactory(() => MyInvoicesPageModel());
  ...
}

void _registerServices() {
  locator.registerLazySingleton(() => AuthenticationService());
  locator.registerLazySingleton(() => InvoiceService());
  ...
}
...
}
```

Listagem 45: SAFTPRO *Desktop Application* - locator.dart

```
import '...';

class MyInvoicesPage extends StatelessWidget {
  static const routeName = '/invoices';

  Widget build(BuildContext context) {
    return BaseView<MyInvoicesPageModel>(
        onModelReady: (model) { model.fetchInvoices(); },
        builder: (context, model, child) => Theme(
              data: UIHelper.getThemeForUser(),
              child: Scaffold(
                appBar: UIHelper.appBar(
                    IconButton(
                      icon: const Icon( Ionicons.home_outline),
                      onPressed: () { Navigator.of(context).pushNamed(HomePage.routeName); }),
                    Ionicons.documents_outline,
                    'My Invoices'),
                body: model.state == ViewState.busy ? Center(child: UIHelper.progressIndicator()) : _buildBody(model),
              ),
            ));
  }

  Widget _buildBody(MyInvoicesPageModel model) {
    return model.hasErrors() ? _buildErrors(model)
                : InvoicesDataTable(invoiceSummaryList: model.invoiceSummaryModelList);
  }

  Widget _buildErrors(MyInvoicesPageModel model) {
    ...
  }
}
```

Listagem 46: SAFTPRO *Desktop Application* - my_invoices_page.dart

Para concluir, resta construir explicitamente o *layout* do `MyInvoicesPage` (listagem 46). Quando a *View* é inicializada o primeiro passo é invocar o método que requisita as faturas à API. Em seguida, consoante o estado da *View* é renderizado um `ProgressIndicator` circular ou o *Widget* `InvoicesDataTable` (listagem 47), que constrói uma tabela com o conjunto de faturas presente no modelo.

```dart
import '...';

class InvoicesDataTable extends StatefulWidget {
  const InvoicesDataTable(
      {Key? key, required InvoiceSummaryModelList invoiceSummaryList})
      : _invoiceSummaryList = invoiceSummaryList, super(key: key);

  final InvoiceSummaryModelList _invoiceSummaryList;

  _InvoicesDataTableState createState() => _InvoicesDataTableState();
}

class _InvoicesDataTableState extends State<InvoicesDataTable> {
  int? _currentSortColumn;
  bool _isAscending = true;
  InvoiceStatus? _filter;
  _DataSource? _dataSource;

  void initState() {
    super.initState();

    final rows = widget._invoiceSummaryList.values.map((i) => _InvoiceRow(i)).toList();

    _dataSource = _DataSource(context, rows: rows);
  }

  Widget build(BuildContext context) {
    return Container(
      margin: UiSettings.kMargin64,
      child: ListView(
        shrinkWrap: false,
        children: [
          PaginatedDataTable(
            rowsPerPage: 6,
            header: Align(alignment: Alignment.centerLeft, child: _buildDropdownFilter()),
            showCheckboxColumn: false,
            sortColumnIndex: _currentSortColumn,
            sortAscending: _isAscending,
            columns: [
              DataColumn(onSort: (colIdx, _) {_sort(colIdx);}, label: Text('Invoice Number')),
              DataColumn(onSort: (colIdx, _) {_sort(colIdx);}, label: Text('Issue Date')),
              DataColumn(onSort: (colIdx, _) {_sort(colIdx);}, label: Text('Seller Identifier')),
              DataColumn(label: Text('Status')),
              DataColumn(label: Text('Observations'))
            ],
            source: _dataSource!,
          ),
        ],
      ),
    );
  }

  Row _buildDropdownFilter() {...}

  void _sort(int idx) {...}
}

class _InvoiceRow {
  _InvoiceRow(this.invoice);

  InvoiceSummaryModel invoice;
  bool selected = false;
}

class _DataSource extends DataTableSource {
  ...
}
```

Listagem 47: SAFTPRO *Desktop Application* - invoices_data_table.dart

5

Validação

A validação da solução segue o plano de testes proposto pela Opensoft, que descreve a abordagem a adotar para garantir a conformidade das soluções desenvolvidas com os requisitos definidos, bem como a abordagem para validar que o sistema realiza o pretendido.

5.1 *User Stories*

As *User stories* são descrições curtas e simples das funcionalidades do sistema na perspetiva da entidade que as vai executar. Para o efeito, apresentam a entidade responsável pela *User Story*, a funcionalidade pretendida e o objetivo para essa funcionalidade. São também identificados os *inputs*, *outputs* e regras de negócio dessa *User Story*, bem como os critérios de aceitação que determinam se a *User Story* está feita de acordo com o esperado.

5.1.1 US1 - Como fornecedor de bens e serviços pretendo comunicar em tempo real uma fatura através do envio de um ficheiro UBL 2.1

- Um fornecedor de bens ou serviços pretende enviar uma fatura ao adquirente;
- A Autoridade Tributária da Administração Fiscal do adquirente, adotou o formato UBL 2.1;
- Tanto o Fornecedor como o Adquirente estão registados nessa Administração Fiscal;
- O fornecedor utiliza o *webservice* disponibilizado pela Administração Fiscal para enviar a informação da Fatura;
- A Administração Fiscal regista os dados da fatura;
- A fatura é posteriormente, encaminhada para o consumidor para aprovação;
- As faturas aprovadas pelos adquirentes ficam permanentemente registadas na Administração Fiscal para efeitos fiscais (Ex: Apuramento do IVA devido).

Regras de Negócio

RN ID	Regra
RN01	A submissão de uma fatura só é possível após autenticação (Autenticação).
RN02	O utilizador só tem acesso à funcionalidade de submissão de faturas caso pertença ao universo de utilizadores habilitados para aceder a esta funcionalidade na Administração Fiscal (Autorização).
RN03	O ficheiro submetido deve ter o encoding UTF-8.
RN04	A fatura deve respeitar a especificação PEPPOL BIS Billing 3.0.
RN05	A informação de routing deve respeitar a especificação PEPPOL AS4 Profile.
RN06	Caso a fatura não seja válida, são comunicados ao emissor os erros presentes no documento, de uma forma detalhada.
RN07	Quando o emissor submete uma fatura com sucesso, tem a garantia de que o Access Point (PEPPOL) do destinatário confirmou a receção da fatura perante este sistema. Isto significa, que a entrega da fatura ao destinatário ocorrerá num futuro próximo.
RN08	Quando a submissão não tem sucesso, é responsabilidade do emissor realizar um novo envio.

Figura 5.1: US1 - Regras de Negócio

Inputs

Campos		Descrição
Ficheiro no formato UBL 2.1		Fatura a submeter.
Informação de Routing	Identificador PEPPOL do Emissor	Permite identificar quem inicia o ato da submissão na rede PEPPOL.
	Identificador PEPPOL do Destinatário	Permite identificar o destinatário do documento na rede PEPPOL.
	Tipo de Documento	Indica que tipo de documento consta no payload da mensagem AS4.
	Identificador do Processo	Especifica a que processo o documento está associado.

Figura 5.2: US1 - *Inputs*

Outputs

No caso da submissão de uma fatura com sucesso os dados são:

Campos	Descrição
Mensagem	"Successfully sent document"

Figura 5.3: US1 - *Output* sucesso

Na tentativa de submissão de uma fatura com erros os dados são:

Campos	Descrição
Mensagem	"The main business message is invalid"
Erros	Descritivo com os erros encontrados.

Figura 5.4: US1 - *Output* insucesso

Diagrama de Sequência

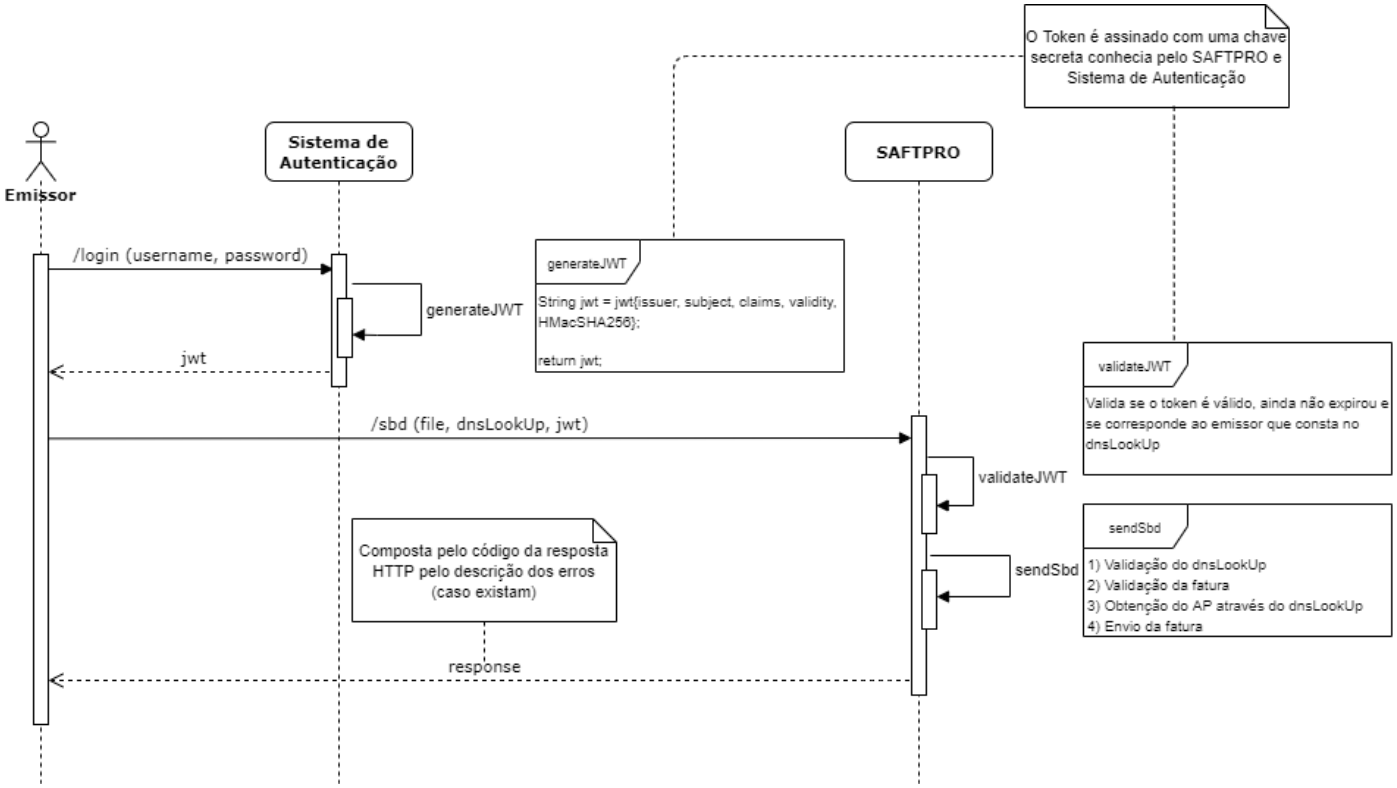


Figura 5.5: US1 - Diagrama de Sequência

Critérios de Aceitação

CA ID	Descrição
CA1	A fatura não pode ser submetida se for considerada inválida segundo as regras de validação do PEPPOL BIS Billing 3.0.
CA2	A fatura não pode ser submetida caso a informação de routing esteja vazia.
CA3	A fatura não pode ser submetida caso a informação de routing esteja inválida.
CA4	A fatura não pode ser submetida caso não seja encontrado o Access Point responsável para o tuplo <identificador Peppol do Destinatário; Tipo de Documento; Identificador do Processo>.

Figura 5.6: US1 - Critérios de Aceitação

5.1.2 US2 - Como adquirente pretendo aprovar uma fatura disponibilizada pela Autoridade Tributária

- O sistema da Autoridade Tributária, recebeu previamente uma fatura emitida por um prestador de bens ou serviços;
- Essa fatura é encaminhada para o adquirente para aprovação;
- Pretende-se que o consumidor aprove ou rejeite essa fatura;
- Caso a fatura seja aprovada, a informação da mesma pode ser utilizada para efeitos Fiscais pela Autoridade Tributária (ex: Apuramento do IVA);
- Caso a fatura seja rejeitada, o Adquirente não efetua o respetivo pagamento e a informação não é utilizada pela Autoridade Tributária para efeitos fiscais.

Regras de Negócio

RN ID	Regra
RN01	A aprovação de uma fatura só é possível após autenticação (Autenticação)
RN02	O utilizador só deverá ter acesso à funcionalidade de receção e aprovação de faturas caso pertença ao universo de utilizadores habilitados para aceder a esta funcionalidade (Autorização)
RN03	Caso a fatura seja aprovada a informação da mesma é utilizada pela Autoridade Tributária para efeitos fiscais (Ex: apuramento do IVA cobrado e devido pelo Fornecedor).
RN04	Caso a fatura não seja aprovada, é responsabilidade do adquirente, resolver as questões derivadas do seu não pagamento fora do sistema, com o fornecedor.
RN05	Caso a fatura não seja aprovada, o consumidor deve indicar o motivo da rejeição.
RN06	Quando o consumidor recebe a notificação de uma fatura, tem a garantia de que esta é válida do ponto de vista semântico e sintático de acordo com o formato UBL 2.1.
RN07	Quando o consumidor recebe a notificação de uma fatura, tem a garantia de que toda a informação relevante para a Autoridade Tributária já foi persistida na Base de Dados do sistema.

Figura 5.7: US2 - Regras de Negócio

Inputs

Campos		Descrição
Identificador da fatura	Número da Fatura	*"A unique identification of the Invoice. The sequential number required in Article 226(2) of the directive 2006/112/EC, to uniquely identify the Invoice within the business context, time-frame, operating systems and records of the Seller. No identification scheme is to be used."*
	Data de Emissão	Data em que a fatura foi emitida.
	Identificador Peppol do Emissor	Permite identificar quem inicia o ato da submissão na rede PEPPOL.
Estado da Fatura	Estado a transitar	A fatura quando é criada encontra-se no estado "Pendente". Depois, transita para o estado "Aceite" ou "Rejeitada".
	Observações	Campo de texto livre que permite acrescentar informação adicional à operação de aprovação/rejeição. No caso da aprovação a sua utilização é opcional, mas é necessário que seja preenchido quando a fatura é rejeitada por parte do consumidor.

Figura 5.8: US2 - *Inputs*

Outputs

No caso de aprovação/rejeição de uma fatura com sucesso os dados são:

Campos	Descrição
Estado da Fatura	É devolvido o estado atualizado da fatura, que deve refletir o input do pedido.

Figura 5.9: US2 - *Output* sucesso

No caso de tentativa de aprovação/rejeição de uma fatura com erros os dados são:

Campos	Descrição
Mensagem	Descritivo do erro que impossibilitou a alteração do estado da fatura.

Figura 5.10: US2 - *Output* insucesso

Diagrama de Sequência

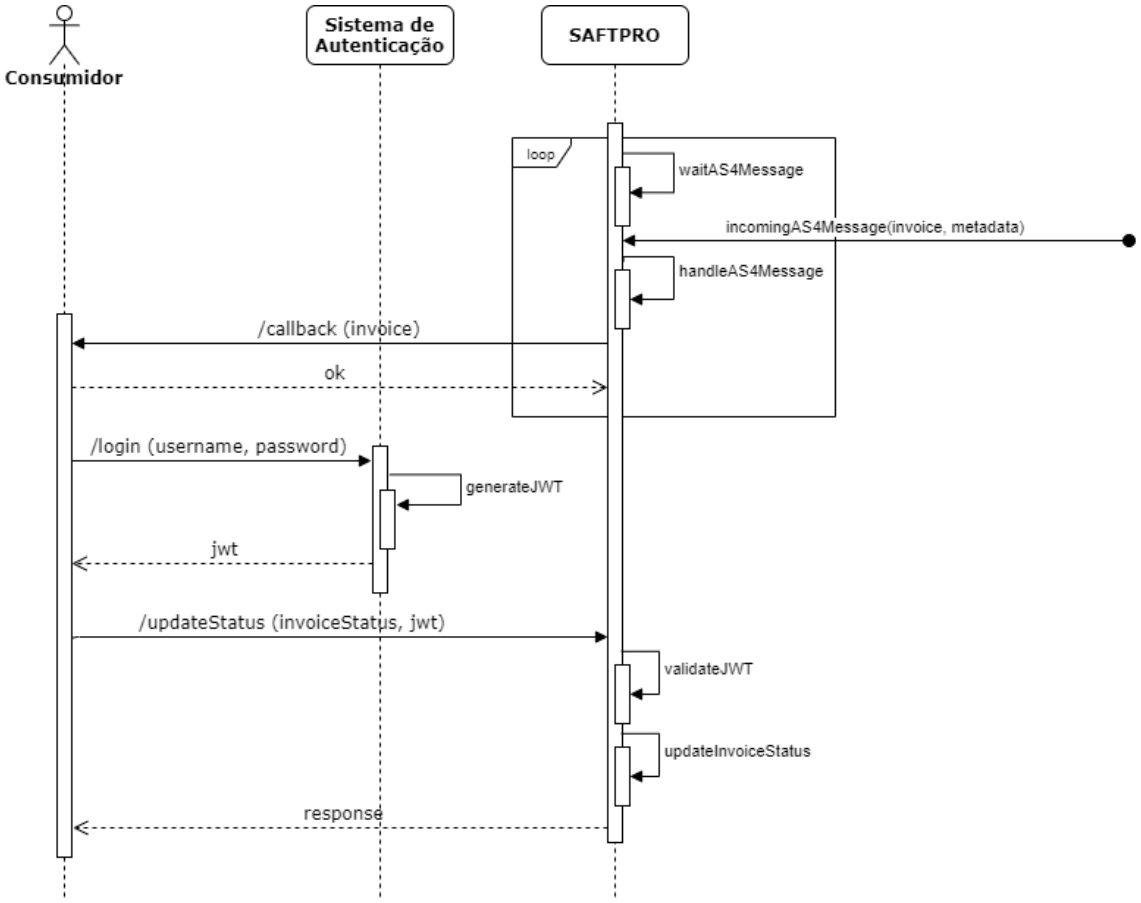


Figura 5.11: US2 - Diagrama de Sequência

Critérios de Aceitação

CA ID	Descrição
CA1	A fatura só pode transitar de estado se ainda não tiver sido aceite ou rejeitada.
CA2	A rejeição de uma fatura só é considerada válida, caso o campo de observações seja preenchido.

Figura 5.12: US2 - Critérios de Aceitação

5.1.3 US3 - Como funcionário da AT, quero consultar as faturas comunicadas através da Peppol *e-Delivery Network*

O funcionário da AT consulta as faturas comunicadas através da rede Peppol, para averiguar o seu estado e verificar alguma inconsistência que possa ocorrer.

Regras de Negócio

RN ID	Regra
RN01	O funcionário poderá consultar qualquer fatura que conste na base de dados do modelo de e-Invoicing.
RN02	Os resultados poderão ser filtrados através dos identificadores PEPPOL dos parceiros comerciais envolvidos, tipo de fatura, estado da fatura e por intervalo de data de submissão.
RN03	Os resultados da pesquisa são originalmente apresentados do mais recente para o mais antigo.
RN04	Ao consultar uma fatura, será aberta uma nova página com informação mais detalhada sobre a mesma.

Figura 5.13: US3 - Regras de Negócio

Inputs

Campos		Descrição
Critérios de pesquisa	Identificador de Participante Emissor	N/A
	Identificador de Participante Destinatário	N/A
	Tipo de Fatura	Lista de tipos considerados no Peppol BIS Billing 3.0.
	Estado da Fatura	O estado da fatura pode ser: PENDENTE, ACEITE, NÃO ACEITE.
	Data início	N/A
	Data fim	N/A

Figura 5.14: US3 - *Inputs*

Outputs

Campos		Descrição
Resultados	Número de fatura	N/A
	Data de emissão	N/A
	Identificador de Participante Emissor	N/A
	Identificador de Participante Destinatário	N/A
	Estado da Fatura	N/A
Detalhes da fatura	Metadados	N/A
	Totais do Documento	N/A
	Comerciante	N/A
	Consumidor	N/A
	Descontos e Encargos	N/A
	Discriminação do IVA	N/A
	Representante Fiscal	N/A
	Instruções de Pagamento	N/A
	Beneficiário	N/A
	Entrega	N/A
	Documentos Adicionais	N/A
	Referências de faturas anteriores	N/A
	Linhas da Fatura	N/A

Figura 5.15: US3 - *Outputs*

Critérios de Aceitação

CA ID	Descrição
CA1	Quando o utilizador submete uma pesquisa com dados válidos, obtém uma lista de faturas.
CA2	Dado um critério de pesquisa válido, quando não existem documentos que lhe correspondem, então é apresentada a menção "Não existem faturas para os critérios de pesquisa indicados".
CA3	Quando o utilizador clicar numa das faturas resultantes da pesquisa é redirecionado para uma página com detalhes da fatura.
CA4	Na página de detalhes da fatura todos os campos obrigatórios estão populados.

Figura 5.16: US3 - Critérios de Aceitação

5.2 Plano de Testes

- Testes Funcionais;
- Testes de Integração;
- Testes de Carga (Opcional);
- Testes de Segurança;
- Testes de Instalação/Sistemas/Usabilidade (Opcional);
- Testes de Interface do Utilizador (Opcional);
- Testes de Desempenho (Opcional);
- Testes de Volume (Opcional);
- Testes de Ciclo de Negócio;
- Testes de Regressão.

5.2.1 Testes Funcionais

Os testes funcionais validam as funcionalidades do sistema, ou seja, asseguram que o sistema funciona em conformidade com os requisitos. Estes testes são efetuados, de forma manual ou automática, como uma "caixa negra", para garantir que a aplicação retorna os resultados esperados face os dados previstos.

O resultado esperado em cada uma das funcionalidades está documentado, sob a forma de critérios de aceitação nas *User Stories*. Estes critérios apresentam as condições que o sistema deve satisfazer para ser aceite pelos *stakeholders*, ou seja, averiguam se a funcionalidade está a operar de acordo com o pretendido. Assim, são usados no âmbito dos testes funcionais para confirmar que, dado o estímulo adequado, as funcionalidades respondem de acordo com o comportamento pretendido.

5.2.2 Testes de Integração

O objetivo dos testes de integração é assegurar que todos os componentes que comunicam com interfaces cumprem os requisitos da interface e podem ser integrados no sistema. Esta comunicação é feita por funcionalidades descritas através de *User Stories*. Assim, a verificação destas interfaces pode ser feita, automaticamente ou manualmente, através de testes funcionais às *User Stories* que comunicam com as interfaces.

5.2.3 Testes de Segurança

Estes testes pretendem assegurar que o sistema não apresenta vulnerabilidades aos ataques de segurança identificados nos requisitos do documento Visão. Estes ataques tentam encontrar diferentes caminhos na solução para causar danos que podem ser mais ou menos graves. Para esta solução, os caminhos que representam potenciais alvos de entrada a estes ataques de segurança são os que a solução expõe, ou seja, são as funcionalidades disponíveis ao público. Assim, estes testes podem ser realizados sobre as *User Stories* documentadas.

A *Open Web Application Security Project*[1] mantém um documento de consciencialização, que vai atualizando, com uma lista dos riscos de segurança mais críticos para aplicações web. De acordo com esta lista, as vulnerabilidades mais típicas das aplicações são:

- **Ataques de SQL *Injection***: envio de dados não confiáveis e não sanitizados para o sistema, que os admite como parte de um comando ou *query* e lhes permite a geração de danos no sistema (fig. 5.17);
- **Quebras de Autenticação:** ataques anónimos para roubo de dados valiosos, especialmente informação de identificação pessoal, comprometendo passwords ou explorando temporária ou permanentemente a identidade do utilizador(fig. 5.18);
- **Ataques *Cross-Site Scripting*:** envio de dados de fontes não confiáveis para o sistema (web app) que não os valida ou filtra, consentindo a criação de HTML ou Javascript no browser da vítima e permitindo o roubo de sessões de utilizadores, deformação dos sites ou redirecionamento do utilizador para sites maliciosos (fig. 5.19).

A realização destes testes é feita seguindo o processo *Test-driven Development*, ou seja, implementando testes automáticos que permitam verificar a segurança da aplicação de acordo com os critérios abaixo indicados.

CA ID	Descrição
CA1	A aplicação não permite a injeção de código SQL através dos campos de preenchimento.
CA2	A aplicação não permite a injeção de código SQL através da alteração do URL da página.

Figura 5.17: Proteção contra Ataques de SQL *Injection* - Critérios de Aceitação

CA ID	Descrição
CA1	Um utilizador com o perfil de Administrador consegue aceder a todas as funcionalidades da aplicação.
CA2	Um utilizador que não tenha o perfil de Administrador apenas consegue aceder às funcionalidades de consulta.
CA3	Um utilizador não autenticado não consegue executar operações que requeiram autenticação.
CA4	Um utilizador autenticado não consegue aceder a informação alheia ou executar operações em nome de terceiros (sem a autorização expressa)

Figura 5.18: Proteção contra Quebras de Autenticação - Critérios de Aceitação

[1] https://owasp.org/

CA ID	Descrição
CA1	A aplicação não permite a injecção de scripts através de informação dinâmica: • Reflected XSS - manipulação de campos refletidos em campos ou URL • Persisted XSS - injeção de scripts em dados persistidos na BD

Figura 5.19: Proteção contra Ataques de *Cross-Site Scripting* - Critérios de Aceitação

As *User Stories* nas quais são incluídos testes automáticos para verificar estes critérios são a US1 e a US2.

5.2.4 Testes de Interface do Utilizador

Estes testes asseguram que a interação do utilizador com a solução se encontra em conformidade com os requisitos de interação do utilizador. Assim, verificam-se sempre nas *User Stories* documentadas e que envolvem interação com o utilizador.

5.2.5 Testes de Regressão

Os testes de regressão garantem que alterações, melhorias ou complementos desenvolvidos na solução (ou componentes da solução) não afetam negativamente a operacionalidade da solução já existente.

Os testes de regressão são executados automaticamente através da bateria de testes unitários, funcionais e de segurança previamente implementados. Desta forma, asseguram a conformidade da solução com especificações previamente estabelecidas verificando que cada uma das funcionalidades do sistema continua a aceitar dados adequados e a retornar o resultado esperado.

O resultado da execução destes testes fica reportado no relatório de testes automáticos.

5.2.6 Itens de Teste

As aplicações abrangidas por este plano de testes são :

- SAFTPRO *Webservices*;
- SAFTPRO Portais;
- SAFTPRO *Desktop Application*.

5.2.7 Pressupostos de Ambiente

Os testes à solução são executados em ambientes previamente estabelecidos, podendo ser diferentes (ex: desenvolvimento, testes ou qualidade) para cada tipo/método de teste a realizar. Para as necessidades dos componentes desenvolvidos os testes serão executados em Desenvolvimento.

5.3 Relatório de Testes Unitários

O relatório de testes unitários foi obtido através do *plugin* Maven Surefire *Report*[2].

5.3.1 Ambiente de Execução

Contexto	Descrição
Sistema Operativo	Windows 10 Pro (10.0.19043 Build 19043)
Processador	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz, 2419 Mhz, 4 Core(s), 8 Logical Processor(s)
Memória RAM	32GB
Plataforma	Spring Boot Starter Test 2.2.5.RELEASE
Java	1.8.0_281-b09
Base de Dados	H2 1.4.199
Ambiente	Desenvolvimento

Figura 5.20: Testes Unitários - Ambiente de Execução

5.3.2 Resultados

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
pt.opensoft.saftpro.util.einvoicing	1	0	0	0	100%	0.182
pt.opensoft.saftpro.validator.component	8	0	0	0	100%	0.056
pt.opensoft.saftpro.validator.einvoicing.sbd	5	0	0	0	100%	0.664
pt.opensoft.saftpro.restcontroller	47	0	0	0	100%	54.735
pt.opensoft.saftpro.restcontroller.einvoicing	20	0	0	0	100%	22.963
pt.opensoft.saftpro.service.einvoicing	16	0	0	0	100%	18.212

Figura 5.21: Surefire *Report* - *Packages*

pt.opensoft.saftpro.util.einvoicing

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	DataLoaderTest	1	0	0	0	100%	0.182

Figura 5.22: Surefire *Report* - pt.opensoft.saftpro.util.einvoicing Classes

DataLoaderTest

	loadData	0.15

Figura 5.23: Surefire *Report* - DataLoaderTest.java *Test Cases*

[2] https://maven.apache.org/surefire/maven-surefire-report-plugin/

pt.opensoft.saftpro.validator.einvoicing.sbd

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	SBDValidatorTest	5	0	0	0	100%	0.664

Figura 5.24: Surefire *Report* - pt.opensoft.saftpro.validator.einvoicing.sbd Classes

SBDValidatorTest

	testValidateWithNonDefaultValidationExecutorSet	0.441
	testValidateWithDefaultValidationExecutorSet	0.195
	testAddValidationExecutorSet	0
	testRemoveValidationExecutorSet	0
	testGetDefaultValidationExecutorSet	0

Figura 5.25: Surefire *Report* - SBDValidatorTest.java *Test Cases*

pt.opensoft.saftpro.restcontroller.einvoicing

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	DirectoryControllerTest	2	0	0	0	100%	4.132
	InvoiceFileControllerTest	3	0	0	0	100%	4.671
	InvoiceStatusControllerTest	7	0	0	0	100%	4.225
	InvoiceSummaryControllerTest	3	0	0	0	100%	3.988
	SBDRestControllerTest	5	0	0	0	100%	5.947

Figura 5.26: Surefire *Report* - pt.opensoft.saftpro.restcontroller.einvoicing Classes

DirectoryControllerTest

	testGetParticipantsUnknownUser	0.143
	testGetParticipantsOk	0.253

Figura 5.27: Surefire *Report* - DirectoryControllerTest.java *Test Cases*

InvoiceFileControllerTest

	testDownloadInvoiceNotFound	0.19
	testDownloadInvoiceForbidden	0.079
	testDownloadInvoicePdfOk	1.045

Figura 5.28: Surefire *Report* - InvoiceFileControllerTest.java *Test Cases*

InvoiceStatusControllerTest

	testPutInvoiceStatusInvalidModel	0.172
	testPutInvoiceStatusInvoiceNotFound	0.111
	testGetInvoiceStatusUserIsNotReceiver	0.114
	testGetInvoiceStatusInvoiceNotFound	0.1
	testPutInvoiceStatusOk	0.095
	testGetInvoiceStatusOk	0.093
	testGetInvoiceStatusUnknownUser	0.08

Figura 5.29: Surefire *Report* - InvoiceStatusControllerTest.java *Test Cases*

InvoiceSummaryControllerTest

	testGetInvoicesSummaryOk	0.183
	testGetInvoicesSummaryUnknownUser	0.095
	testGetInvoicesSummaryUserIsNotCustomer	0.105

Figura 5.30: Surefire *Report* - InvoiceSummaryControllerTest.java *Test Cases*

SBDRestControllerTest

	testSendSenderParticipantDoesNotMatch	0.314
	testSendNoRegisteredParticipant	0.094
	testSendValidationErrors	1.435
	testSendExpiredToken	0.01
	testSendOk	0.117

Figura 5.31: Surefire *Report* - SBDRestControllerTest.java *Test Cases*

pt.opensoft.saftpro.service.einvoicing

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	HumanReadableServiceTest	3	0	0	0	100%	3.439
	InvoiceFileServiceTest	2	0	0	0	100%	3.547
	InvoiceStatusServiceTest	5	0	0	0	100%	4.027
	InvoiceSummaryServiceTest	1	0	0	0	100%	3.108
	SBDHandlerServiceTest	5	0	0	0	100%	4.091

Figura 5.32: Surefire *Report* - pt.opensoft.saftpro.service.einvoicing Classes

HumanReadableServiceTest

	testParseXmlToPdfOk	0.155
	testDocumentToByteArrayInvalidDocument	0.015
	testParseXmlToPdfInvalidArray	0

Figura 5.33: Surefire *Report* - HumanReadableServiceTest.java *Test Cases*

InvoiceFileServiceTest

	testFindInvoiceFileNotFound	0.015
	testFindInvoiceFileOk	0

Figura 5.34: Surefire *Report* - InvoiceFileServiceTest.java *Test Cases*

InvoiceStatusServiceTest

	testUpdateStatusInvoiceNotFound	0
	testUpdateStatusParameterNotValid	0.01
	testUpdateStatusOk	0
	testFindInvoiceStatusOk	0
	testFindInvoiceStatusNotFound	0

Figura 5.35: Surefire *Report* - InvoiceStatusServiceTest.java *Test Cases*

InvoiceSummaryServiceTest

	testFindAllByCustomerElectronicAddressOk	0.016

Figura 5.36: Surefire *Report* - InvoiceSummaryServiceTest.java *Test Cases*

SBDHandlerServiceTest

	testHandleDuplicateInvoice	0.531
	testHandleInvalidBusinessMessage	0.031
	testHandleSuccessFollowedByDeletion	0.156
	testHandleTransactionFailure	0.09
	testHandleSuccess	0.159

Figura 5.37: Surefire *Report* - SBDHandlerServiceTest.java *Test Cases*

5.4 Relatório de Testes Manuais de Aceitação

Este relatório pretende apresentar o resultado dos testes que não se enquadram na bateria de testes automáticos, mas pertencem ao Plano de Testes. Estes promovem incrementalmente a deteção antecipada de problemas e podem resultar na remoção antecipada de defeitos. Os seus resultados poupam um custo considerável de isolamento de falhas e trabalho refeito associado a problemas detetados na solução.

Os testes manuais executados cobrem os seguintes testes do Plano de Testes:

- Testes Funcionais;
- Testes de Integração;
- Testes de Interface do Utilizador.

5.4.1 Ambiente de Execução

Contexto	Descrição
Sistema Operativo	Windows 10 Pro (10.0.19043 Build 19043)
Processador	11th Gen Intel(R) Core(TM) i5-1135G7 @ 2.40GHz, 2419 Mhz, 4 Core(s), 8 Logical Processor(s)
Memória RAM	32GB
Plataforma	Docker (20.10.8, Build 3967b7d)
Browser	Chrome
Java	1.8.0_281-b09
Base de Dados	MySQL 8.0.27
Ambiente	Desenvolvimento

Figura 5.38: Testes Manuais de Aceitação - Ambiente de Execução

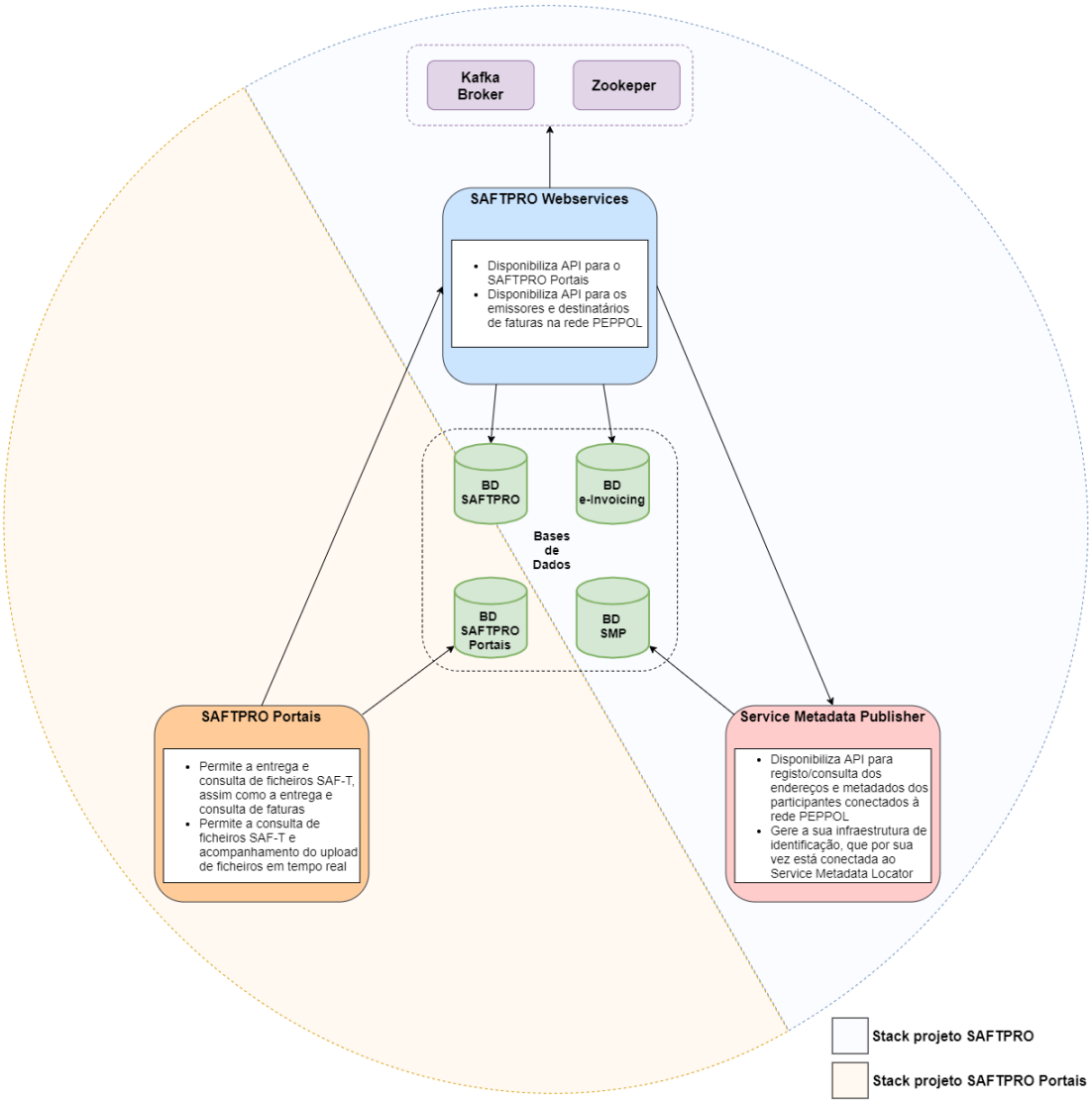


Figura 5.39: Docker - Visão Global

5.4.2 Resultados - *User Story* 1

Os cenários descritos abaixo necessitam de uma fase de configuração do *Access Point* e as credenciais de um utilizador registado, com capacidade para submeter faturas.

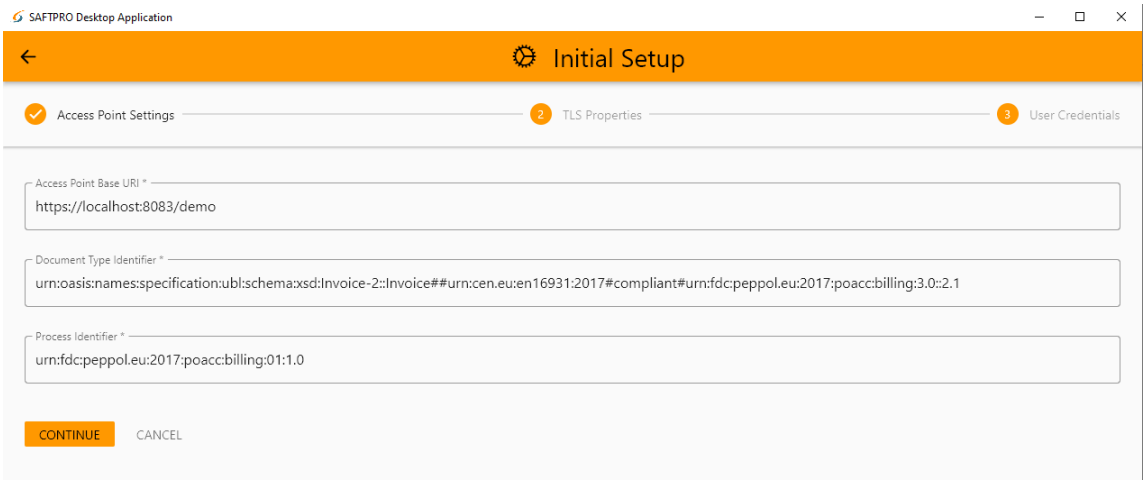


Figura 5.40: US1 - Configuração das propriedades do *Access Point*

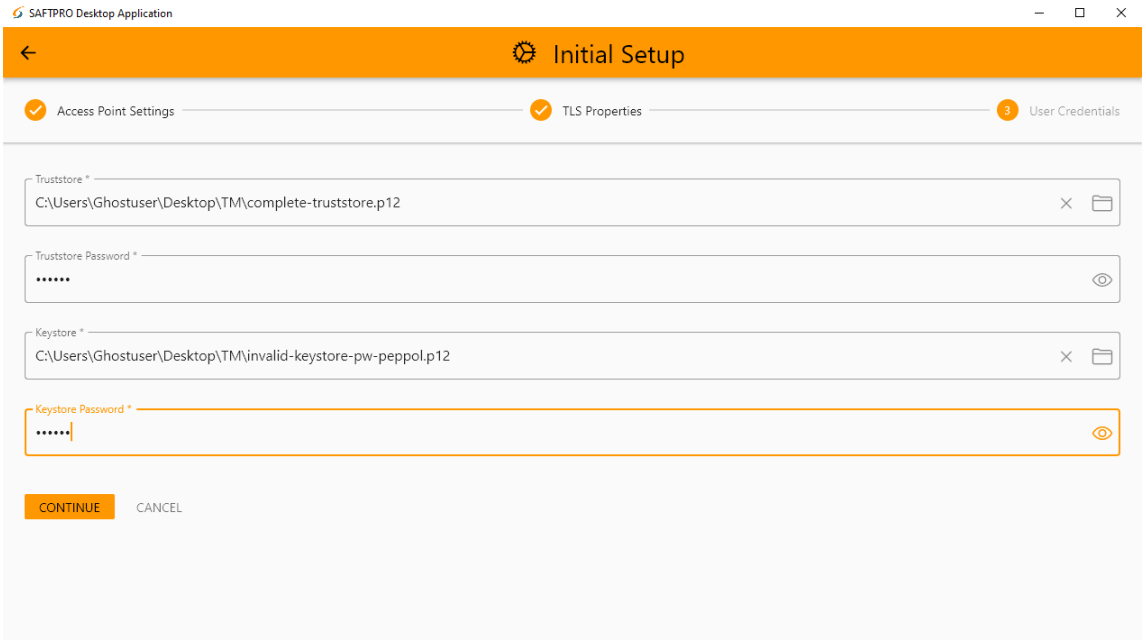


Figura 5.41: US1 - Configuração das propriedades do TLS

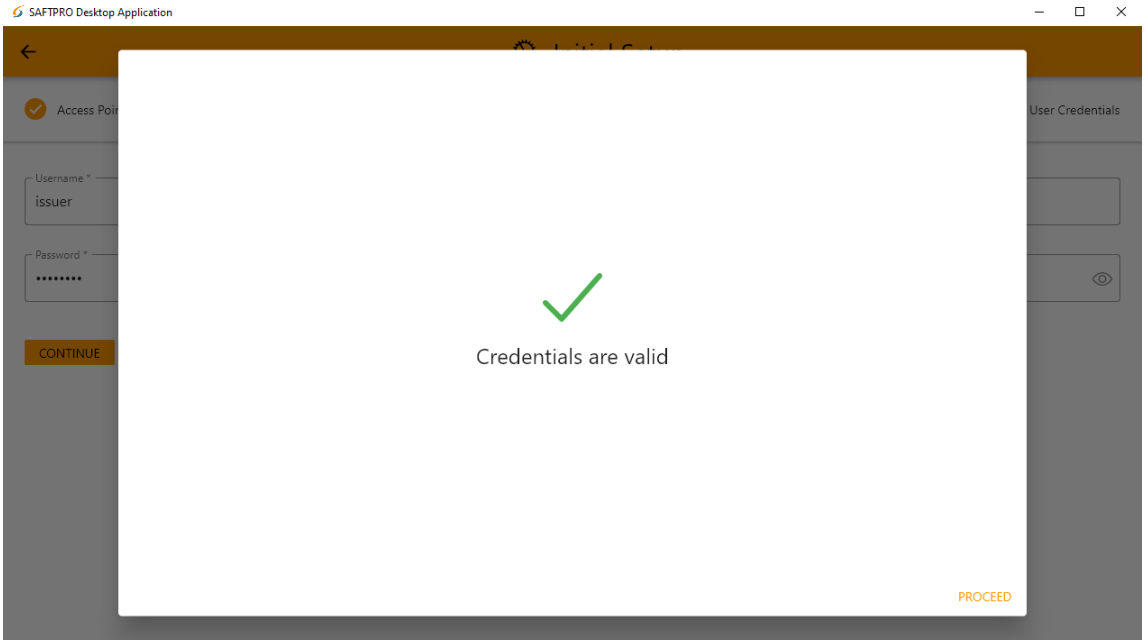


Figura 5.42: US1 - *Login* com um utilizador credenciado

5.4.2.1 Critério de Aceitação - CA1

5.4.2.1.1 Cenário 1

Indicando os campos obrigatórios do *webservice* e um ficheiro que ainda não tenha sido submetido ele é validado e transmitido ao Consumidor.

Dados de Teste / Observações

Identificador Peppol do Comerciante: 9915:phase4-test-issuer

Identificador Peppol do Consumidor: 9946:pt231971460

Número da Fatura: 1234567

Data de Emissão: 2021-10-28

Ficheiro: base-example.xml

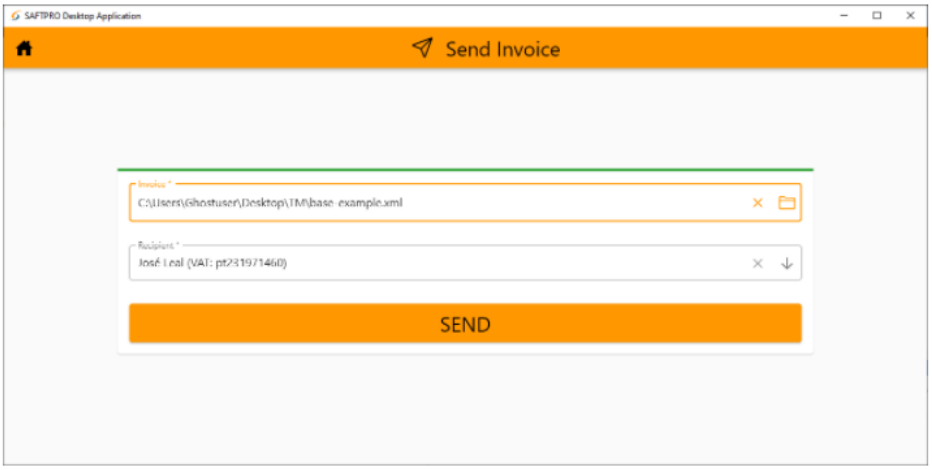


Figura 5.43: US1-CA1-C1 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

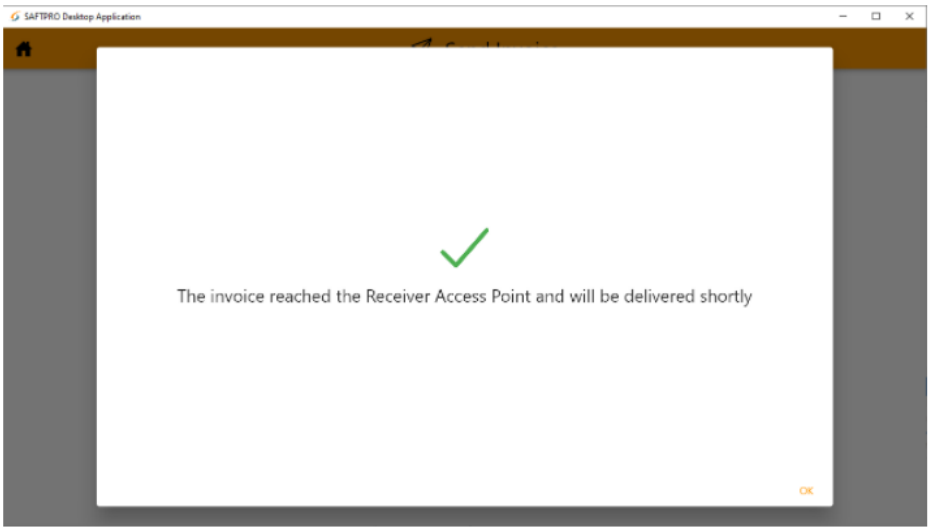


Figura 5.44: US1-CA1-C1 - Dados Obtidos

5.4.2.1.2 Cenário 2

Indicando os campos obrigatórios do *webservice* e um ficheiro que já tenho sido submetido ele é detetado como duplicado.

Dados de Teste / Observações

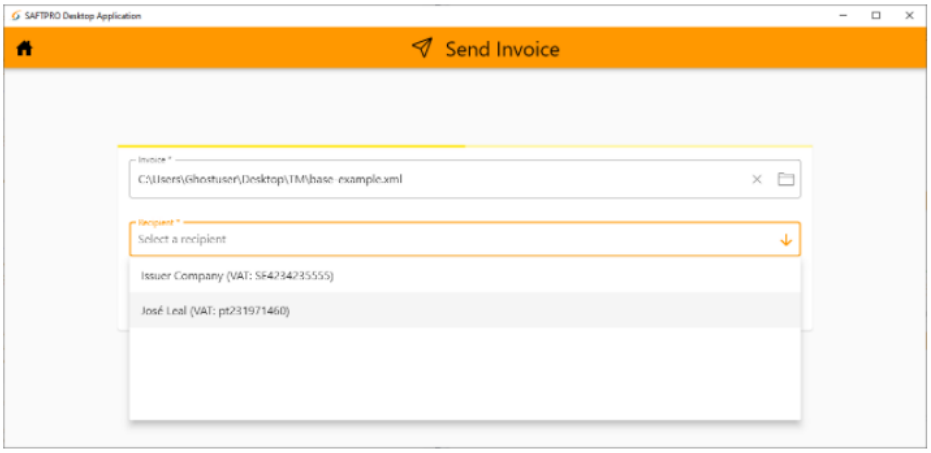
Identificador Peppol do Comerciante: 9915:phase4-test-issuer

Identificador Peppol do Consumidor: 9946:pt231971460

Número da Fatura: 1234567

Data de Emissão: 2021-10-28

Ficheiro: base-example.xml



Será executado o mesmo caso do cenário 1, visto que se pretende detetar a submissão de duplicados.

Figura 5.45: US1-CA1-C2 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

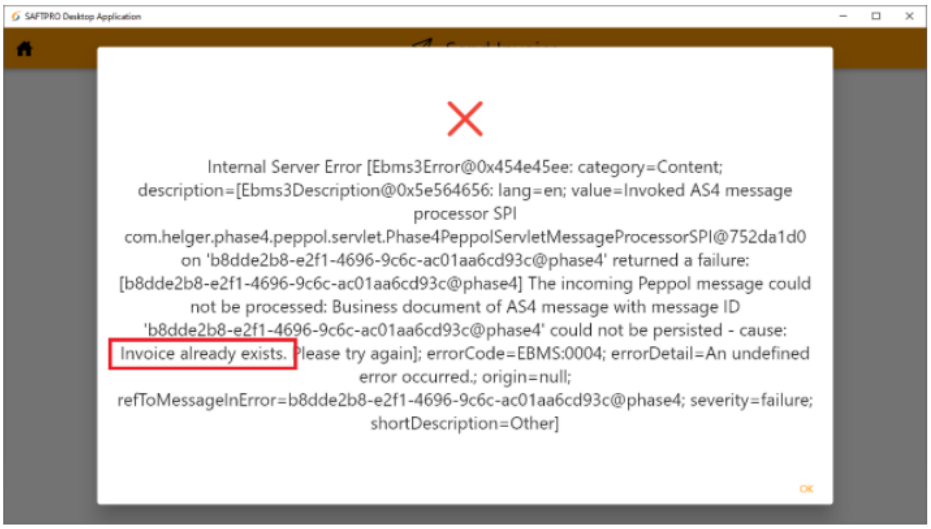


Figura 5.46: US1-CA1-C2 - Dados Obtidos

5.4.2.1.3 Cenário 3

Indicando os campos obrigatórios do *webservice* e um ficheiro que seja inválido ele não é transmitido ao destinatário final.

Dados de Teste / Observações

Identificador Peppol do Comerciante: 9915:phase4-test-issuer

Identificador Peppol do Consumidor: 9946:pt231971460

Número da Fatura: 1234/78/901

Data de Emissão: 2016-06-16

Ficheiro: invalid-example.xml

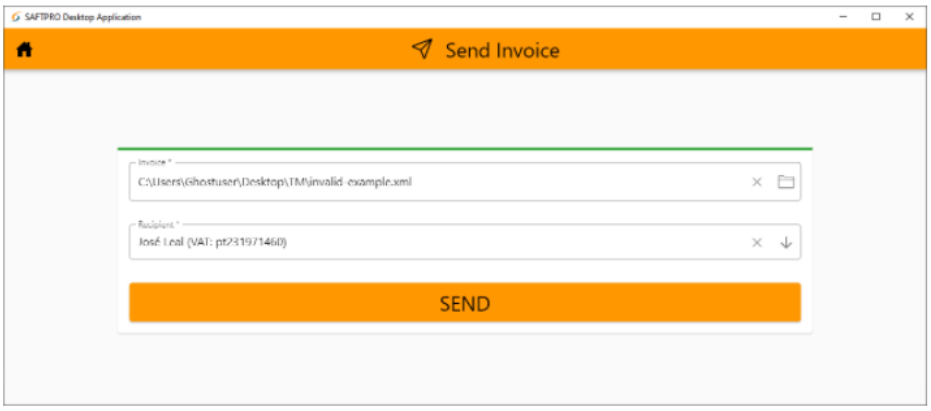
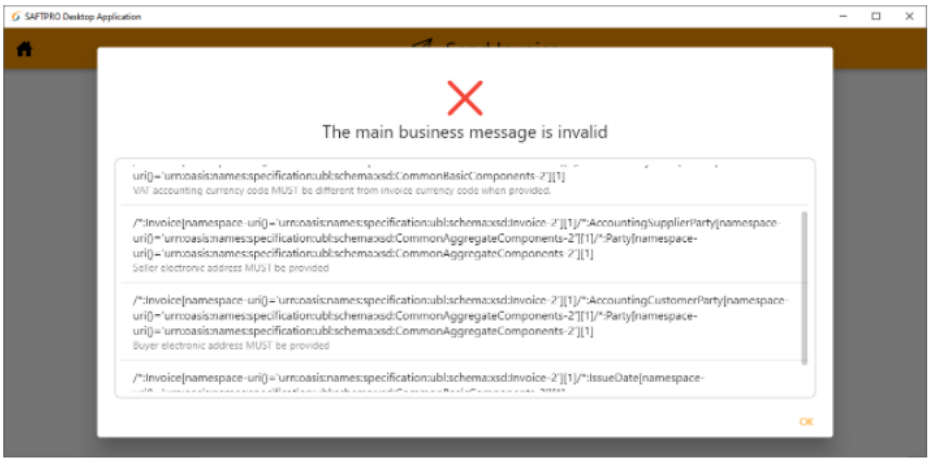


Figura 5.47: US1-CA1-C3 - Dados de Teste

Dados Obtidos/ Observações

Resposta:



Quando a fatura não está correta, o webservice devolve o conjunto de erros assinalados na validação do documento.

Figura 5.48: US1-CA1-C3 - Dados Obtidos

5.4.2.2 Critério de Aceitação - CA2

Implementado de forma transparente na aplicação, pelo que não é possível invocar o *webservice* sem que toda a informação de *routing* esteja presente.

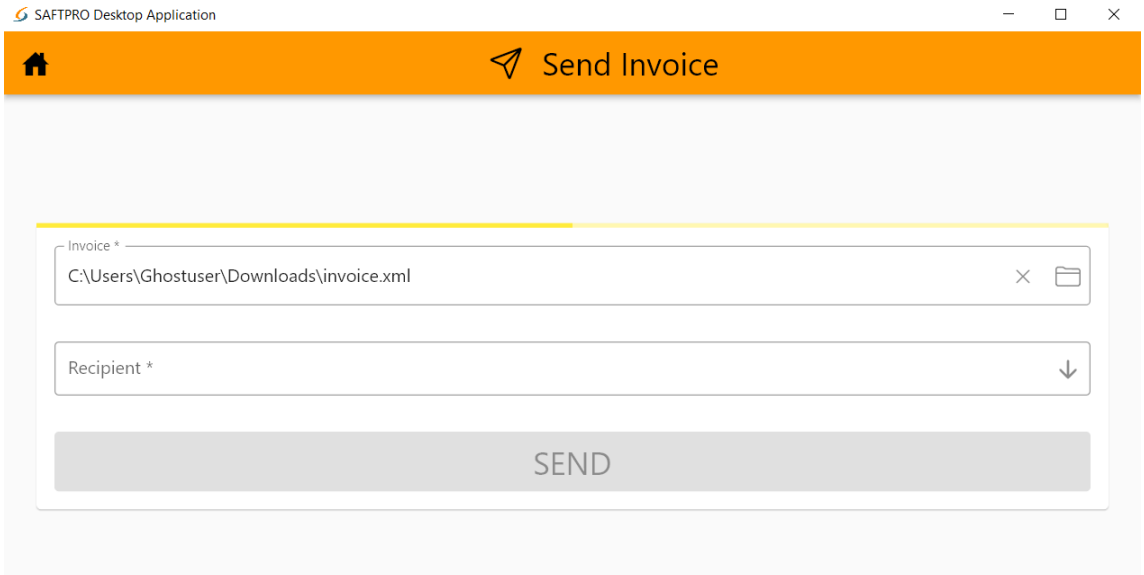


Figura 5.49: US1-CA2 - Formulário de submissão

Como demonstrado na figura acima, caso o *Recipient* não seja escolhido, o botão para envio fica desativado.

5.4.2.3 Critério de Aceitação - CA3

5.4.2.3.1 Cenário 1

Indicando os campos obrigatórios do *webservice* e um ficheiro cujo o emissor e destinatário são diferentes dos que constam na fatura, esta não é entregue.

Dados de Teste / Observações

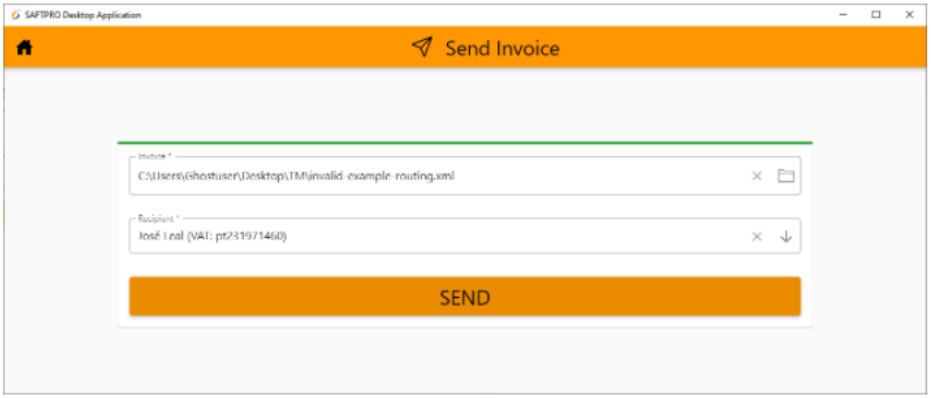
Identificador Peppol do Comerciante: 9915:phase4-test-issuer

Identificador Peppol do Consumidor: 9946:pt231971460

Número da Fatura: 1234567

Data de Emissão: 2021-10-28

Ficheiro: invalid-example-routing.xml



Neste cenário o identificador Peppol do Comerciante na fatura, não corresponde ao que está presente na informação de routing.

Figura 5.50: US1-CA3-C1 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

Figura 5.51: US1-CA3-C1 - Dados Obtidos

5.4.2.4 Critério de Aceitação - CA4

Para testar este critério é necessário efetuar uma pequena alteração nas configurações do *Access Point*.

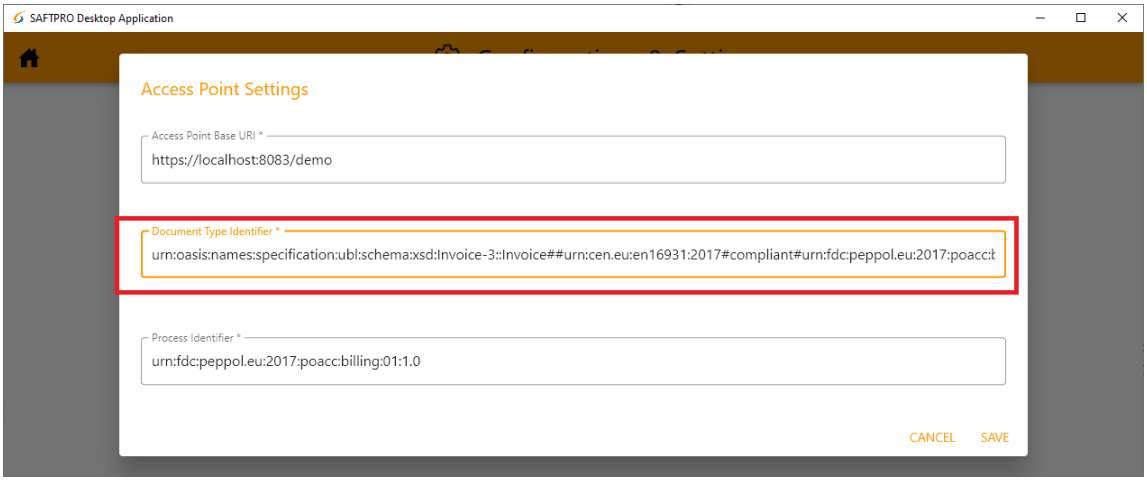


Figura 5.52: Alteração do tipo de documento

5.4.2.4.1 Cenário 1

Indicando os campos obrigatórios do *webservice* e uma fatura válida, ela não é entregue.

Dados de Teste / Observações

Identificador Peppol do Comerciante: 9915:phase4-test-issuer

Identificador Peppol do Consumidor: 9946:pt231971460

Número da Fatura: 1234567

Data de Emissão: 2021-10-28

Ficheiro: base-example.xml

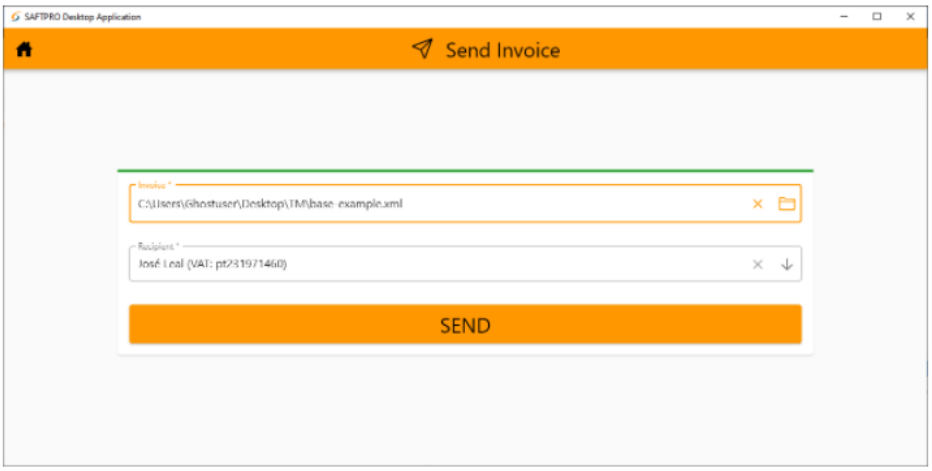
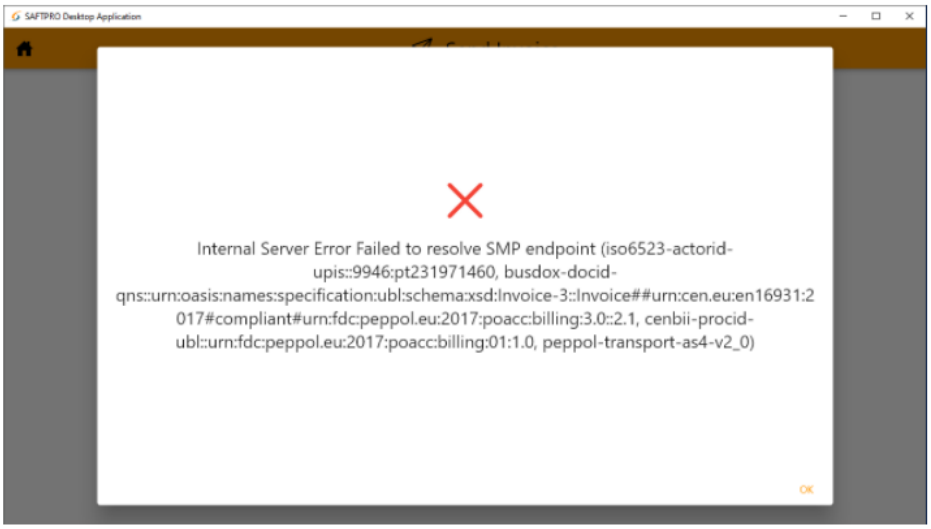


Figura 5.53: US1-CA4-C1 - Dados de Teste

Dados Obtidos/ Observações

Resposta:



Apesar da fatura ser válida, o tipo de documento que foi utilizado para construir o pedido não é igual ao que foi utilizado pelo Consumidor no processo de registo com o SMP. Deste modo, o lookup falha e não é possível encaminhar o documento.

Figura 5.54: US1-CA4-C1 - Dados Obtidos

5.4.3 Resultados - *User Story* 2

Os cenários abaixo necessitam de uma fase de configuração do *Access Point* e as credenciais de um utilizador registado como Consumidor.

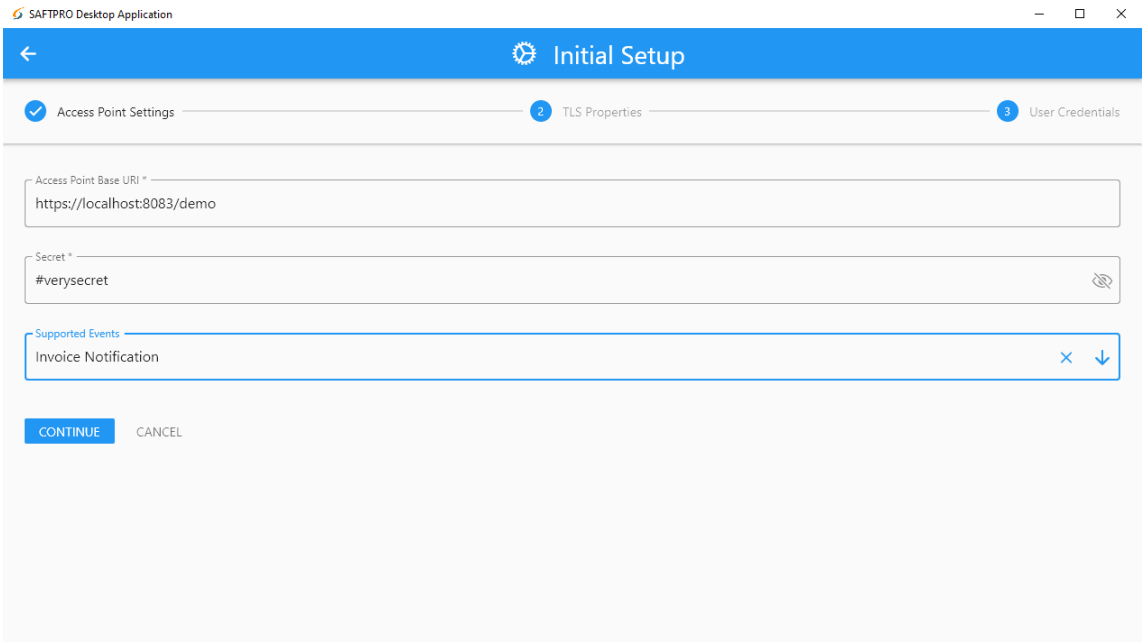


Figura 5.55: US2 - Configuração das propriedades do *Access Point*

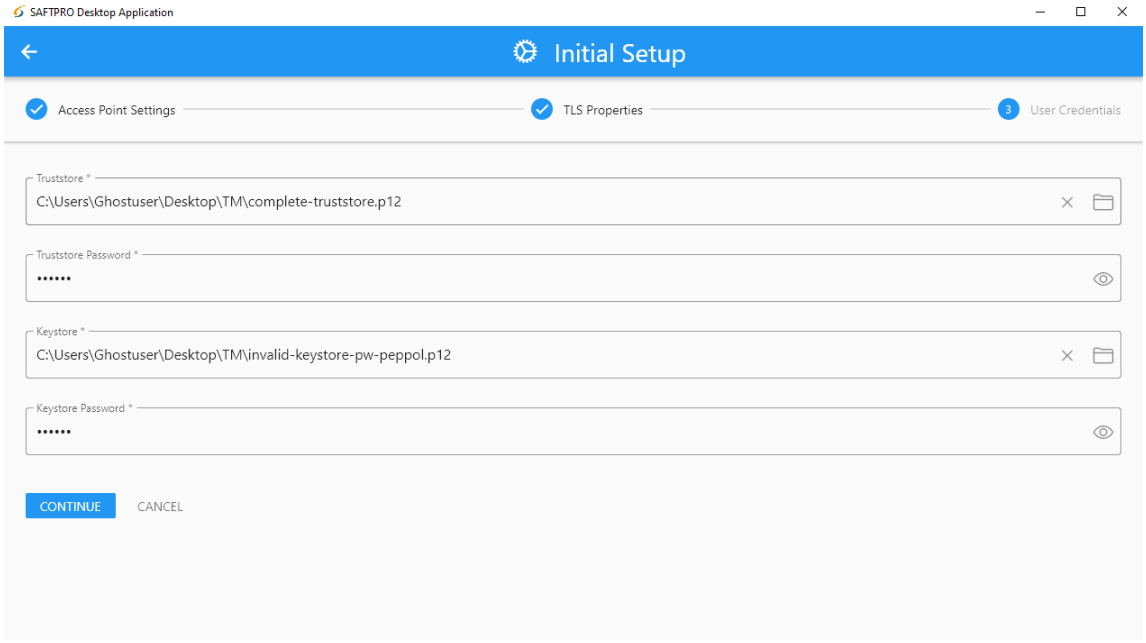


Figura 5.56: US2 - Configuração das propriedades do TLS

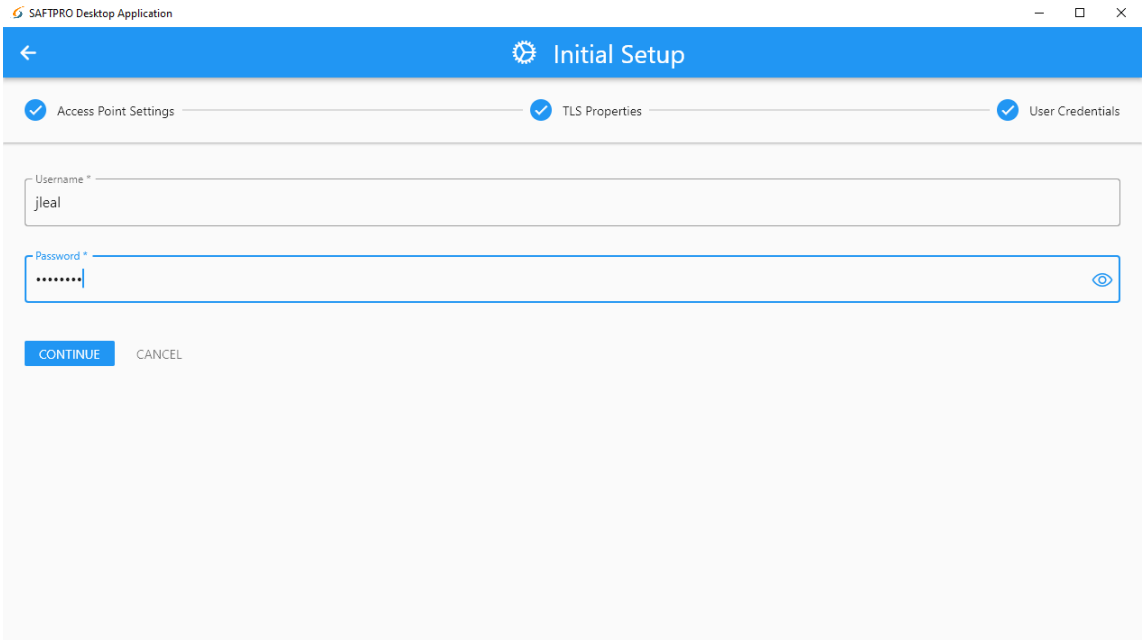


Figura 5.57: US2 - *Login* com um utilizador credenciado

5.4.3.1 Critérios de Aceitação - CA1 e CA2

Ambos os critérios foram considerados na construção da aplicação. Assim, um utilizador só terá disponível a funcionalidade para aceitar ou rejeitar uma fatura, caso o estado atual da mesma ainda esteja pendente. Em relação ao formulário, quando o utilizador rejeita a fatura o botão de submissão só é disponibilizado após o preenchimento do campo de observações.

5.4.3.1.1 Cenário 1

O Consumidor recebe uma fatura, através de uma notificação e aprova.

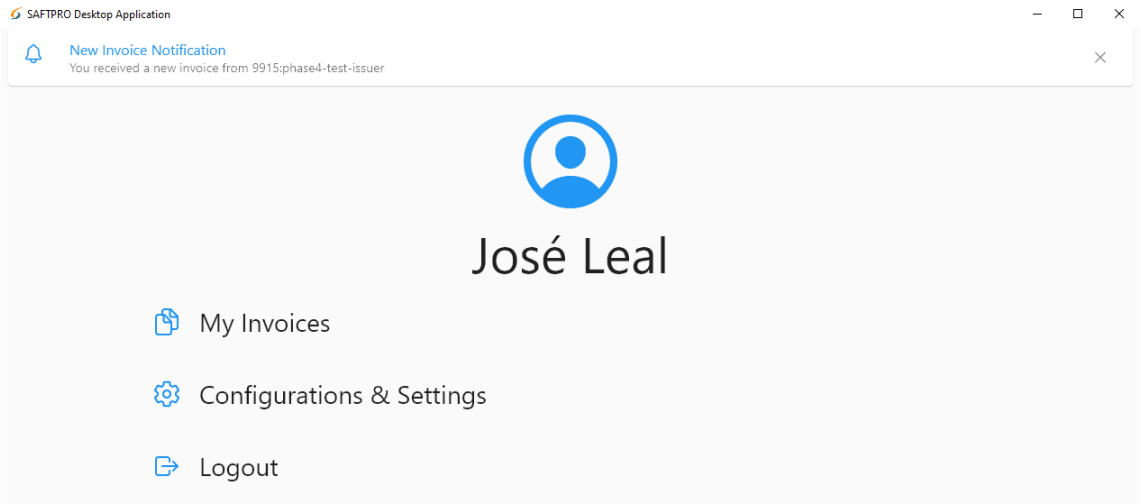


Figura 5.58: US2-CA1/CA2-C1 - Notificação de nova fatura

Ao clicar na notificação, o utilizador é redirecionado para os detalhes da fatura.

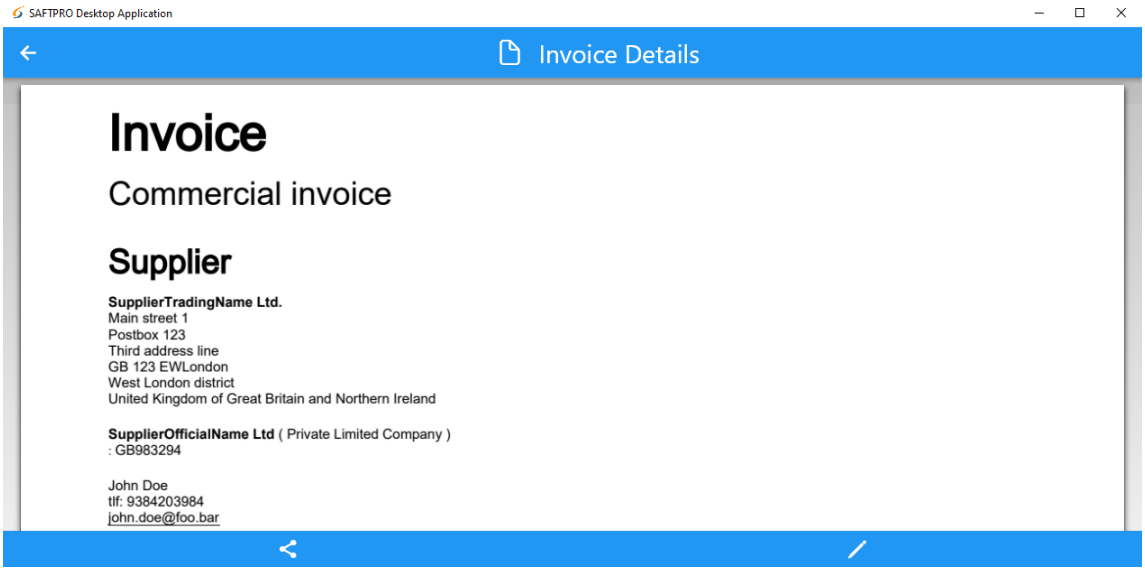


Figura 5.59: US2-CA1/CA2-C1 - Detalhes da fatura

Dados de Teste / Observações

Identificador Peppol do Consumidor: 9946:pt231971460

Número da Fatura: 1234567

Data de Emissão: 2021-10-28

Estado: Accepted

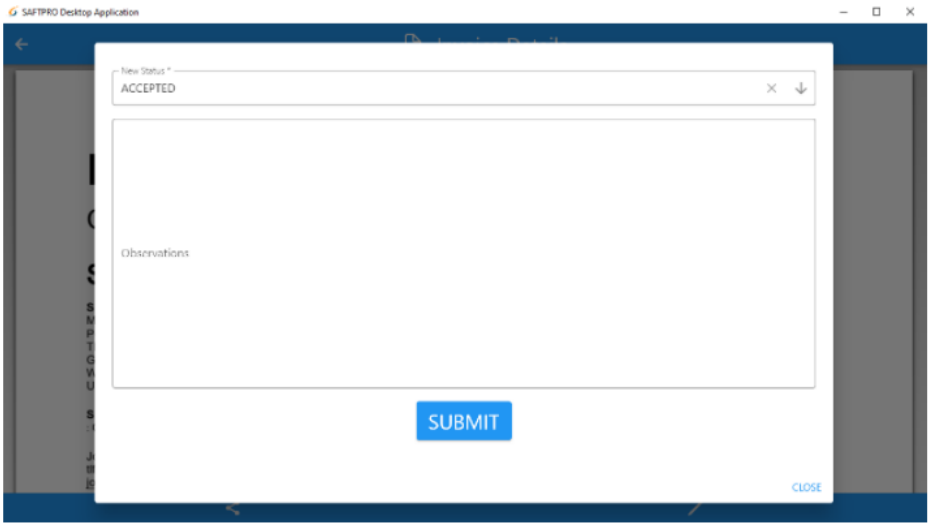


Figura 5.60: US2-CA1/CA2-C1 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

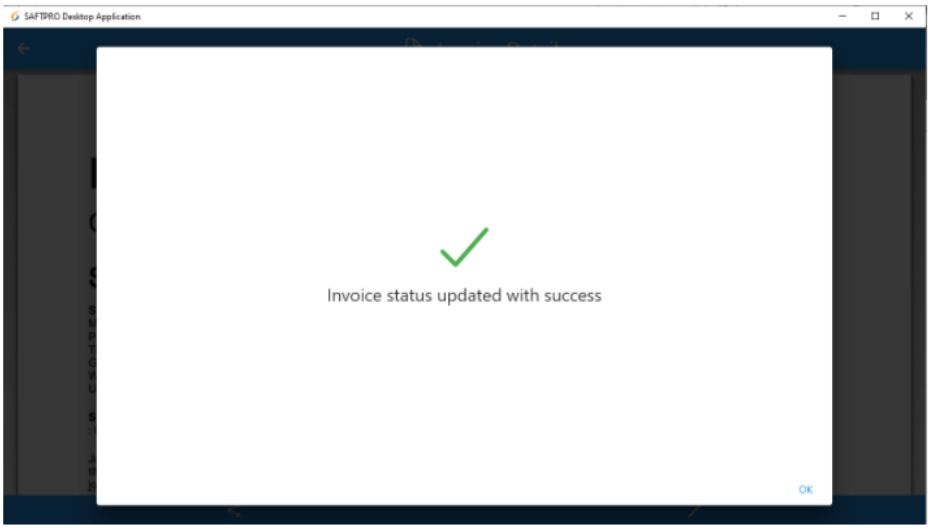


Figura 5.61: US2-CA1/CA2-C1 - Dados Obtidos

5.4.3.1.2 Cenário 2

O Consumidor acede às suas faturas, verifica que tem uma nova e rejeita.

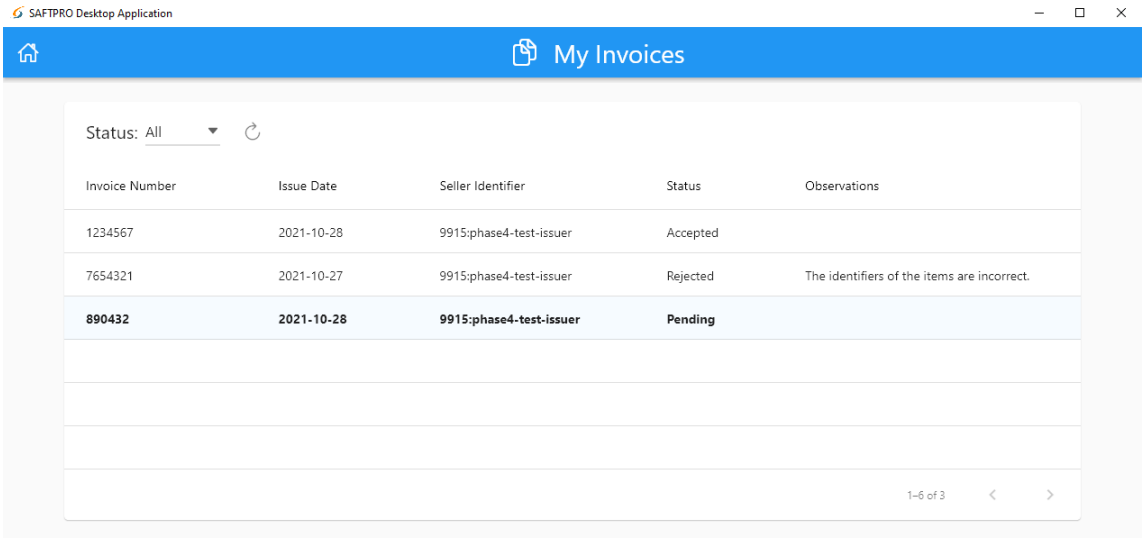


Figura 5.62: US2-CA1/CA2-C2 - Consulta da lista de faturas

À semelhança do cenário anterior se o utilizador clicar na fatura será redirecionado para os seus detalhes.

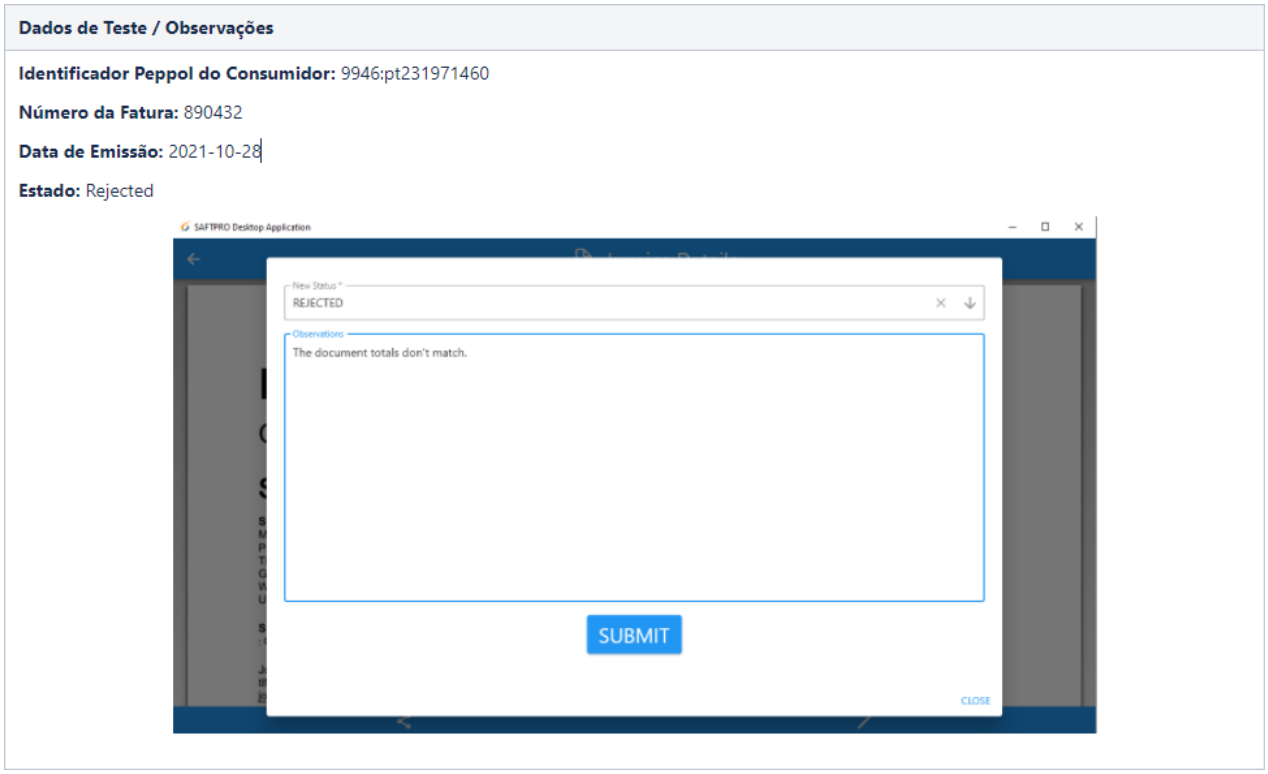


Figura 5.63: US2-CA1/CA2-C2 - Dados de Teste

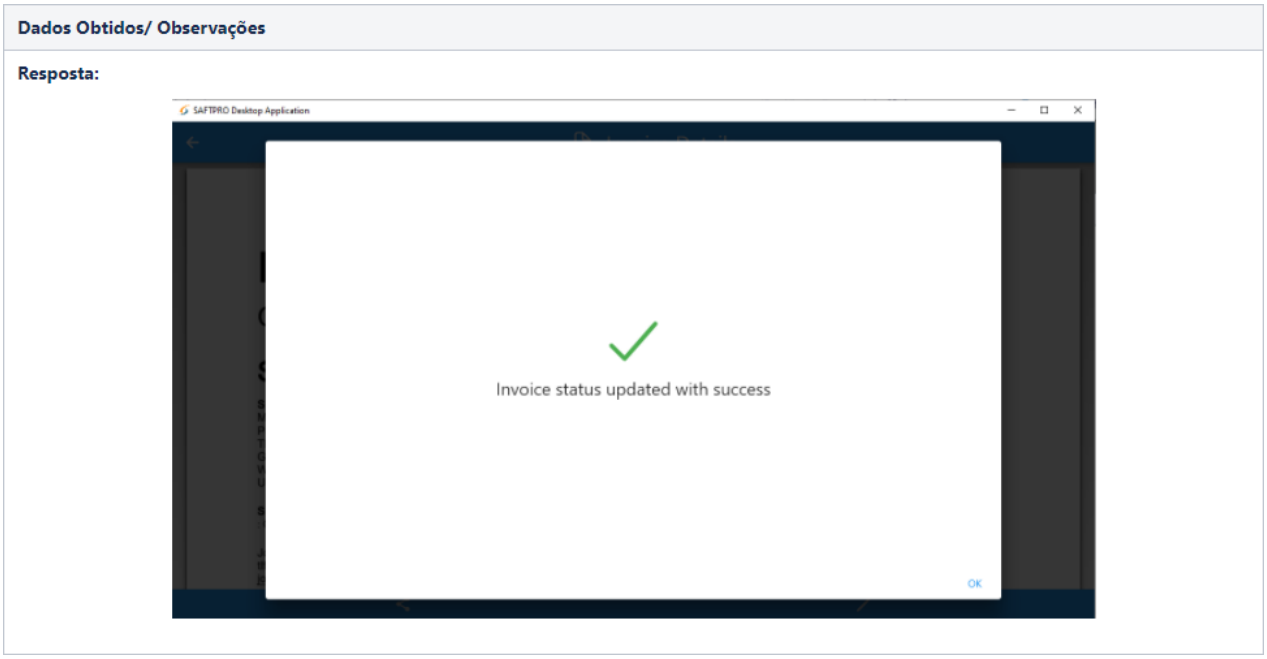


Figura 5.64: US2-CA1/CA2-C2 - Dados Obtidos

5.4.4 Resultados - *User Story* 3

Os cenários descritos abaixo necessitam de um utilizador com acesso ao *backoffice* do SAFTPRO Portais.

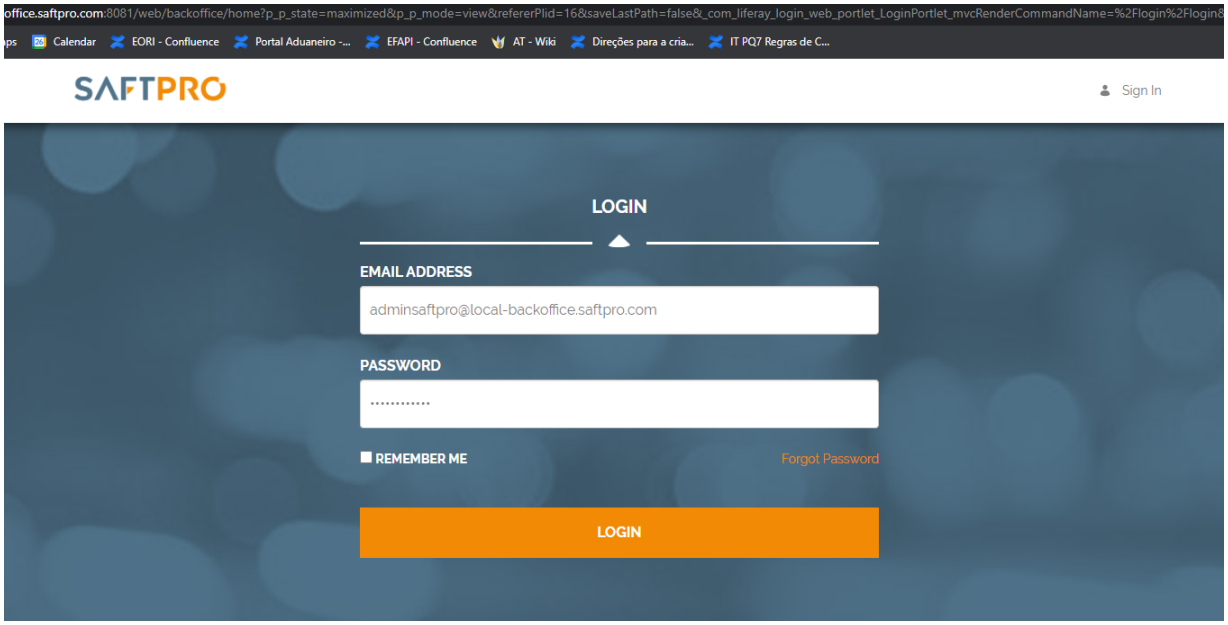


Figura 5.65: US3 - Acesso ao backoffice do SAFTPRO Portais

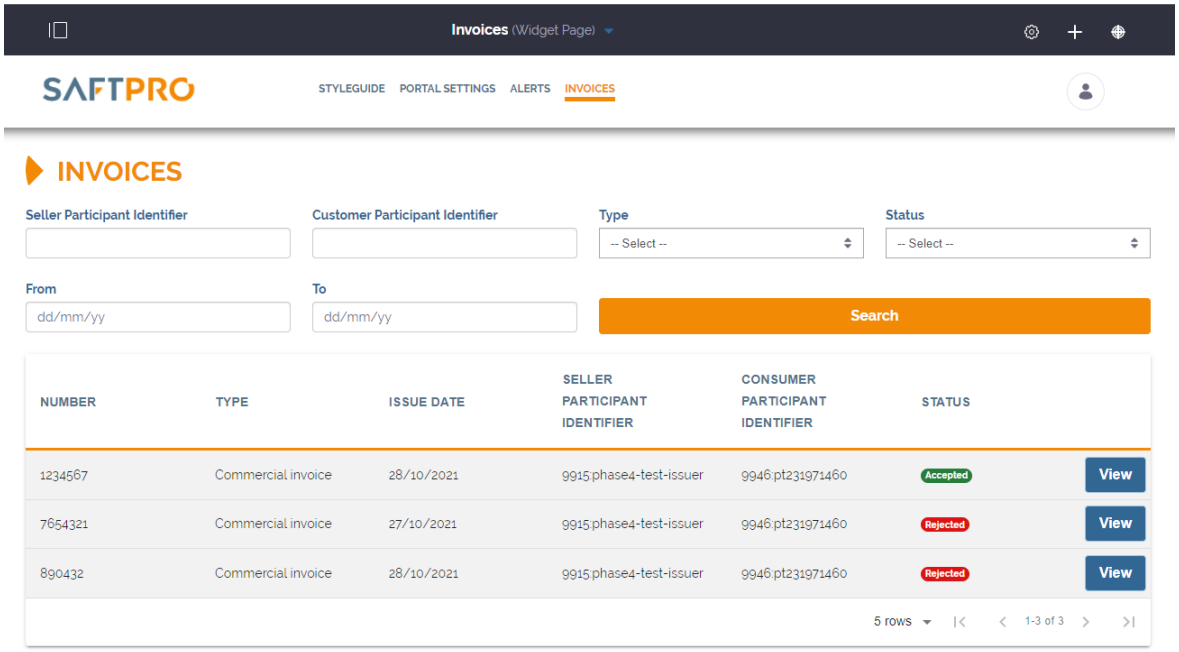


Figura 5.66: US3 - Página de consulta de faturas submetidas

5.4.4.1 Critério de Aceitação - CA1

5.4.4.1.1 Cenário 1

O funcionário efetua uma pesquisa válida e restringe o número de faturas que aparecem na consulta.

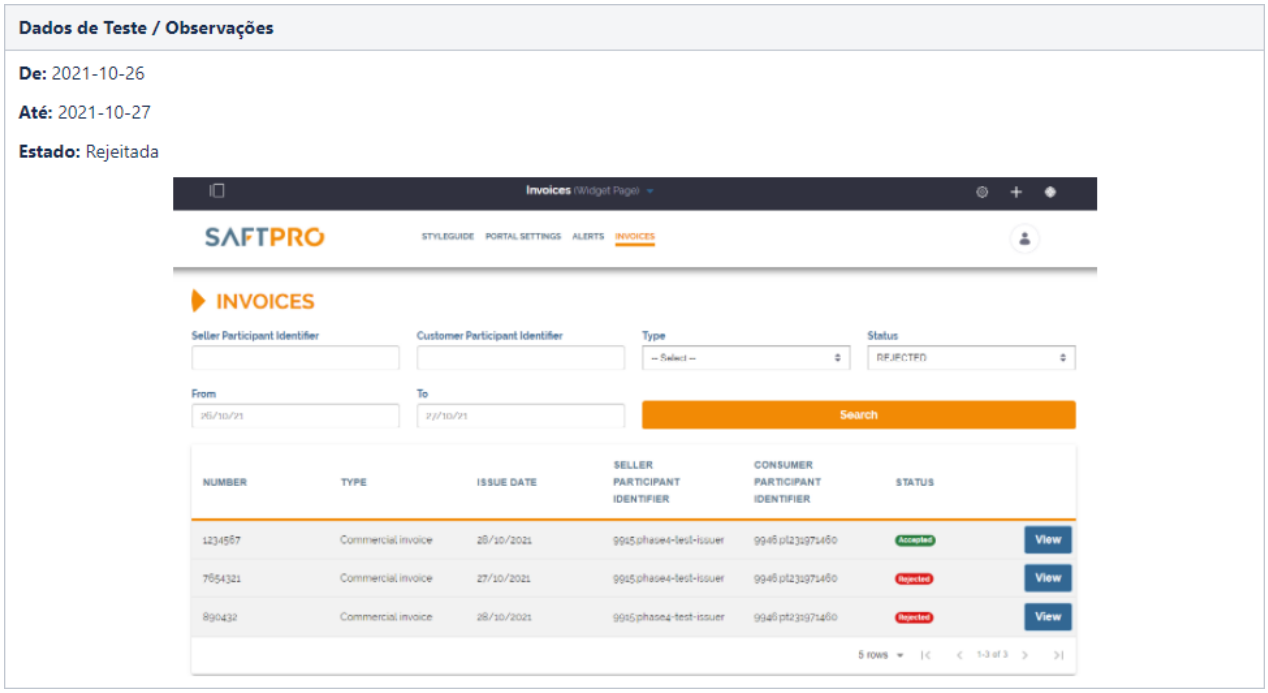


Figura 5.67: US3-CA1-C1 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

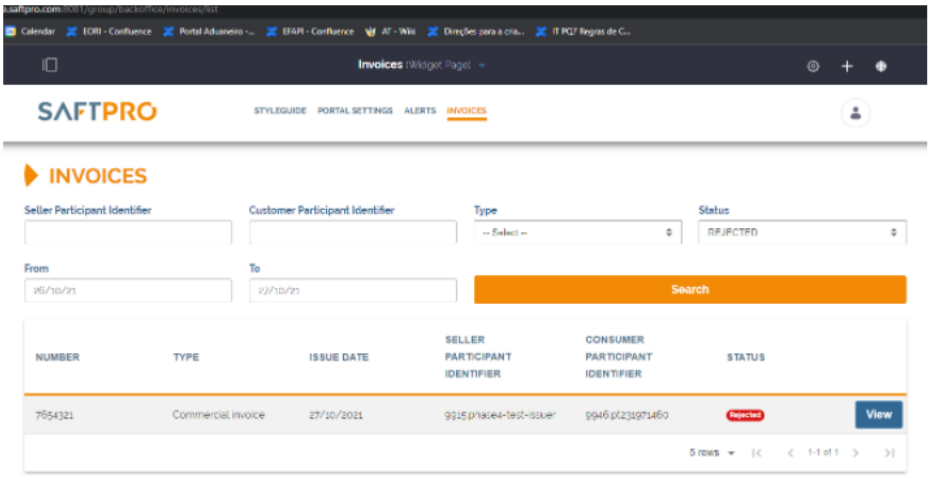


Figura 5.68: US3-CA1-C1 - Dados Obtidos

5.4.4.1.2 Cenário 2

O funcionário tenta efetuar uma pesquisa com um identificador inválido e recebe uma mensagem de erro.

Dados de Teste / Observações

Identificador Peppol do Consumidor: id12345

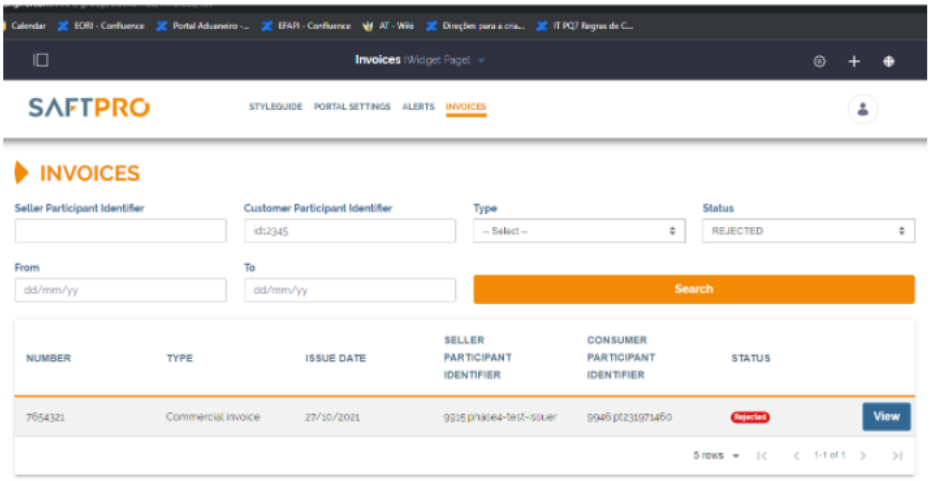


Figura 5.69: US3-CA1-C2 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

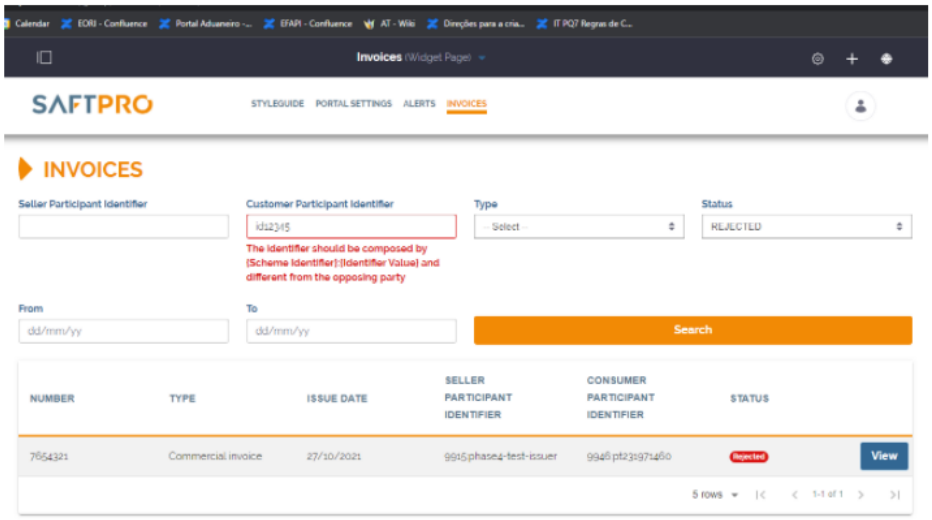


Figura 5.70: US3-CA1-C2 - Dados Obtidos

5.4.4.1.3 Cenário 3

O funcionário tenta efetuar uma pesquisa com um intervalo de datas inválido e recebe uma mensagem de erro.

Dados de Teste / Observações

De: 2021-11-20

Até: 2021-09-13

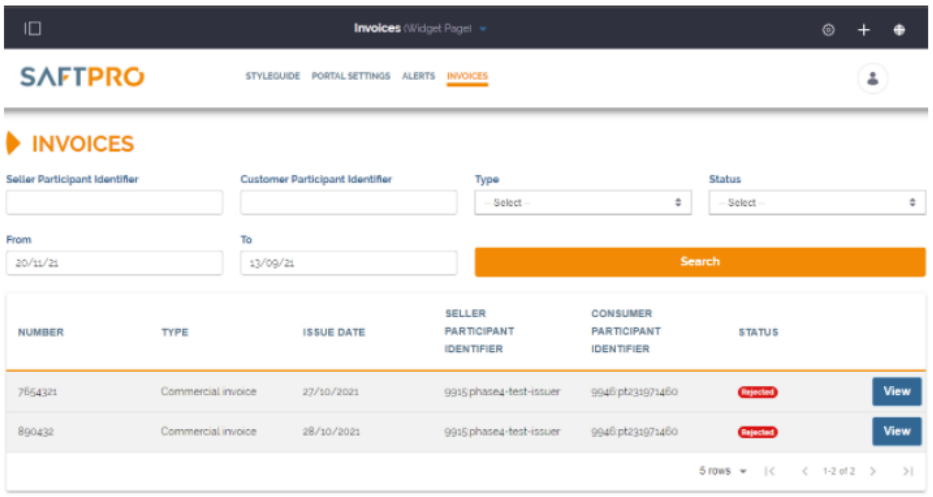


Figura 5.71: US3-CA1-C3 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

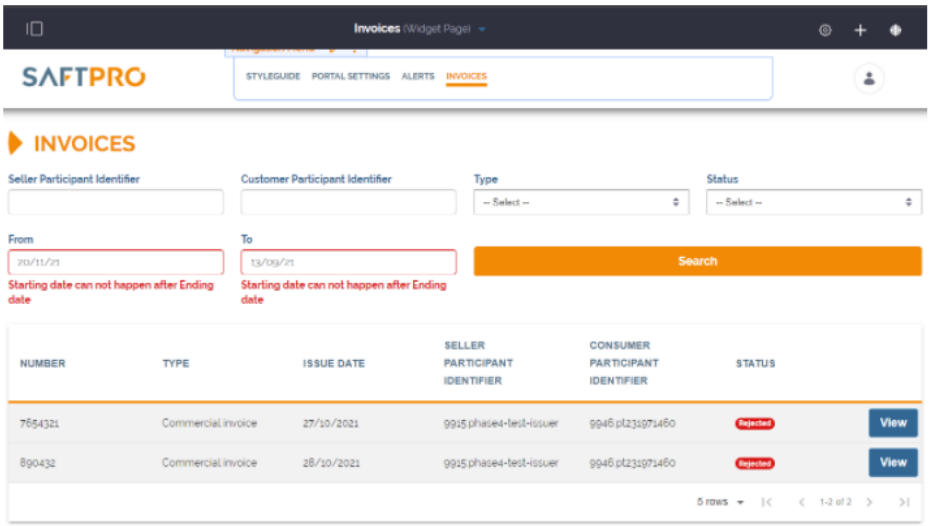


Figura 5.72: US3-CA1-C3 - Dados Obtidos

5.4.4.2 Critério de Aceitação - CA2

5.4.4.2.1 Cenário 1

O funcionário faz uma pesquisa válida, mas não encontra nenhum resultado.

Dados de Teste / Observações

Identificador Peppol do Consumidor: 9946:pt231971460

Estado: Pendente

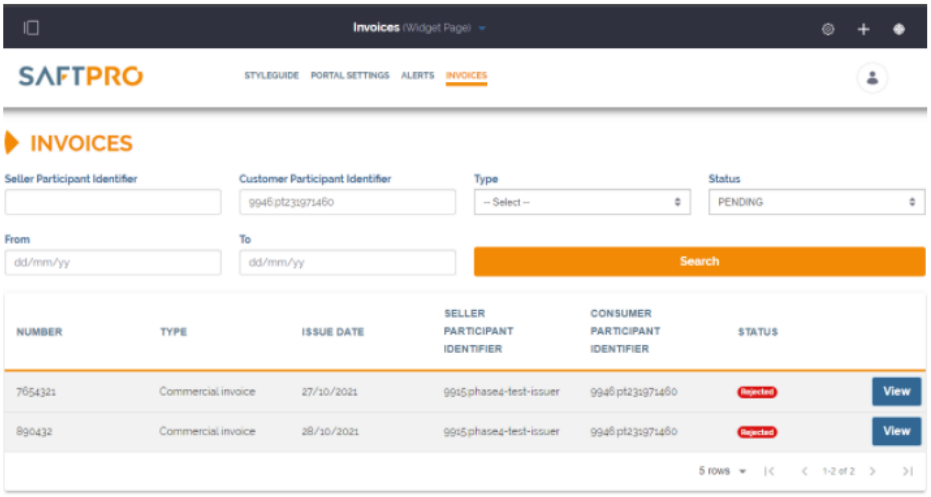


Figura 5.73: US3-CA2-C1 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

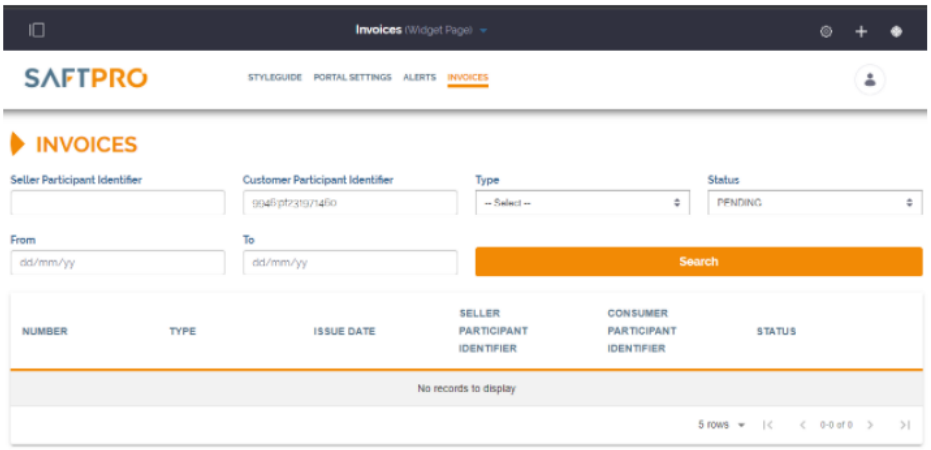


Figura 5.74: US3-CA2-C1 - Dados Obtidos

5.4.4.3 Critérios de Aceitação - CA3 e C4

Estes critérios podem ser testados nos mesmos cenários. O *webservice* devolve sempre todos os dados da fatura que ficaram persistidos na base de dados, obrigatórios ou não.

5.4.4.3.1 Cenário 1

O funcionário seleciona uma fatura e consulta os seus detalhes.

Dados de Teste / Observações

Número da Fatura: 1234567

Estado: Aceite

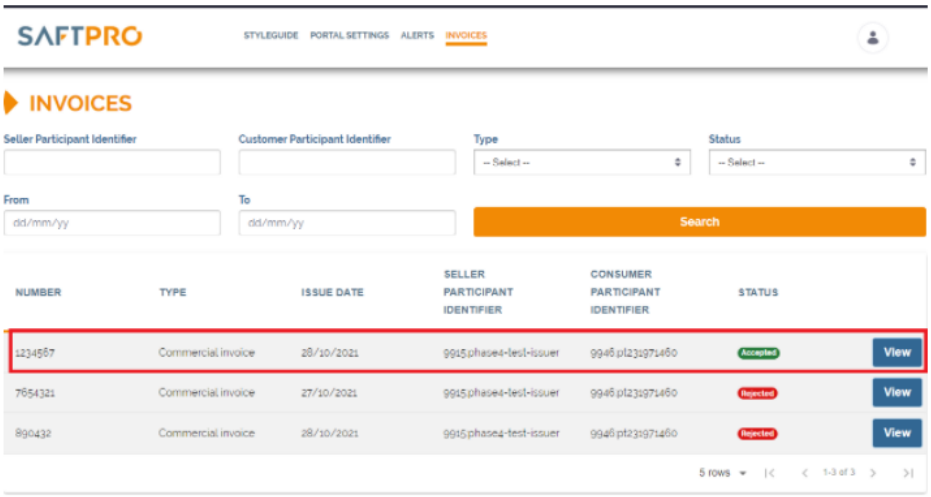
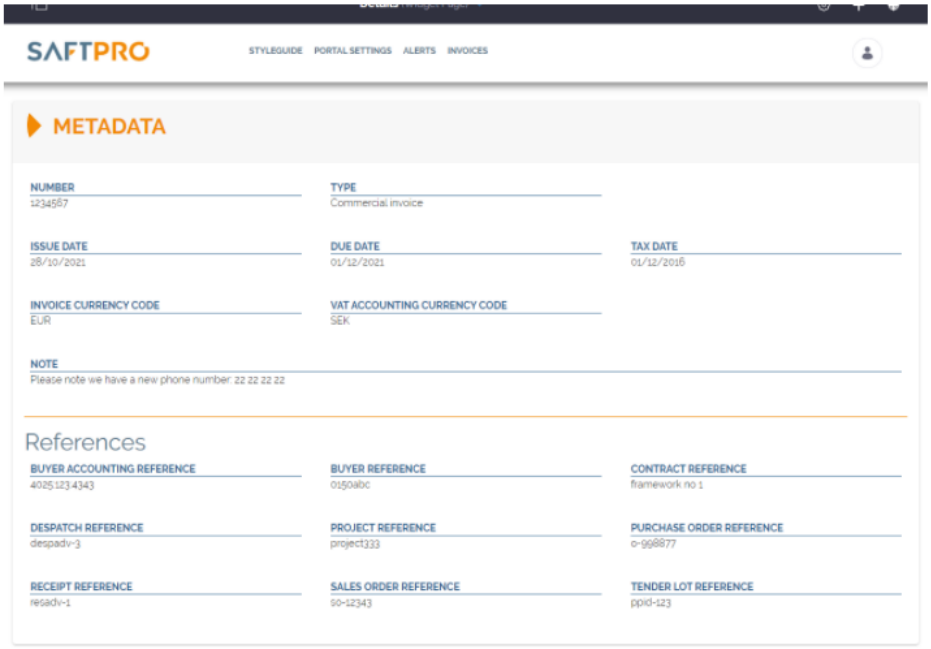


Figura 5.75: US3-CA3/C4-C1 - Dados de Teste

Dados Obtidos/ Observações

Resposta:

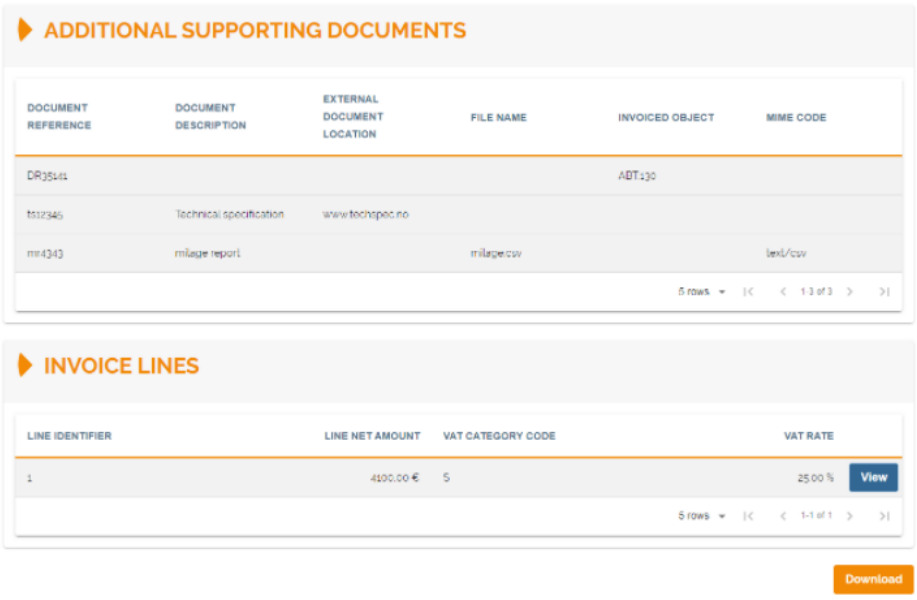


A página de detalhe dá para dar scroll, de modo a consultar todos os detalhes.

Figura 5.76: US3-CA3/C4-C1 - Dados Obtidos (1/2)

Dados Obtidos/ Observações

Resposta:



No fim da página de detalhe existe a possibilidade de fazer download do documento em pdf.

Figura 5.77: US3-CA3/C4-C1 - Dados Obtidos (2/2)

6

Conclusões e Trabalho Futuro

6.1 Conclusões

Nesta tese, abordou-se a problemática da faturação eletrónica e implementou-se uma prova de conceito, que serve de referência, para transformar o SAFTPRO num produto mais completo. De raíz, o SAFTPRO é um *software* direcionado para a recolha e processamento de ficheiros SAF-T, com o objetivo de fornecer informações críticas, sobre transações comerciais, às Autoridades Tributárias. A solução disponibiliza duas interfaces para a submissão de ficheiros, *upload* e *webservice*, tornando a sua adoção relativamente simples pelos Agentes Económicos. Contudo, o modelo tributário suportado permite que o prazo de transmissão deste tipo de ficheiros varie entre alguns dias a alguns meses, o que pode resultar em baixa visibilidade sobre as transações e auditorias tardias.

Para mitigar este problema e no sentido de evoluir funcionalmente o produto, foi desenvolvido um módulo de *e-Invoicing* para viabilizar o processo de validação/aprovação das faturas, no momento em que as transações estão a decorrer.

Grande parte do trabalho incidiu sobre a exploração da infraestrutura disponibilizada pelo Peppol, que permite a interoperabilidade na troca de documentos comerciais. O projeto foi concebido inicialmente para operar na União Europeia, mas atualmente é utilizado um pouco por todo mundo.

O Peppol é composto por três grandes pilares que trabalham em sintonia para permitir a comunicação, de forma coordenada e segura, entre sistemas que podem ser heterogéneos entre si. São eles:

- a Peppol *e-Delivery Network*;
- os *Transport Infrastructure Agreements*;
- e as Peppol *Business Interoperability Specifications*.

Assim, relativamente aos objetivos iniciais propostos, as contribuições foram as seguintes:

- Adaptou-se o formato das faturas submetidas. O SAFTPRO não suportava o formato mais adequado à submissão de faturas em tempo real. Pelo que foi necessário,

dotar o sistema com capacidade para processar e validar um formato que deriva do modelo internacional UBL 2.1, sem perder a flexibilidade para se adaptar às invariantes de um país, mediante configurações do sistema. O formato escolhido foi a CIUS Peppol BIS Billing 3.0, que resulta da transposição do modelo semântico da norma europeia EN16931 e do modelo internacional UBL 2.1;

- Disponibilizou-se canais para o Consumidor confirmar a informação presente nos campos da fatura, antes de esta ser utilizada para efeitos fiscais pela Autoridade Tributária;
- Alargou-se as funcionalidades da API, para facilitar a integração do SAFTPRO, como *Access Point*, na rede Peppol. Deste modo, após o registo no sistema é possível utilizar o SAFTPRO como *hub*, para submeter diretamente as faturas acabadas de emitir e receber simultaneamente as notificações das validações efetuadas, assim como a aprovação das faturas por parte dos contribuintes.

A utilização de *Open-Standards* permite assegurar a longevidade e o bom funcionamento da solução, assim como a sua manutenção no futuro. Para além disso, a adoção de bibliotecas *Open-Source* também foram uma mais valia, pois abriram portas para contribuir com os conhecimentos adquiridos ao longo da tese. Destaca-se a abordagem utilizada para integrar o Spring com a biblioteca phase4, que se encontra disponível no repositório da mesma, sobre a licença Apache 2.0[1], como referência para os seus utilizadores.

De momento, a Opensoft já está a promover a solução de *Tax Clearance*, no artigo *"VAT COMPLIANCE REPORTS: POST-AUDIT AND TAX CLEARANCE"*[2], como um modelo suportado pelo SAFTPRO. Pelo que, os próximos passos são contactar autoridades tributárias que pretendem implementar este modelo de reporte fiscal.

6.2 Trabalho Futuro

De um modo geral, atingiram-se todos os objetivos propostos, como é possível verificar pela validação dos critérios de aceitação descritos no capítulo 5. Não obstante, existe sempre espaço para melhorar e refinar o que foi desenvolvido.

Relativamente aos *workflows* do Peppol, existe alguns pontos que devem ser melhorados. Por exemplo, o registo dos participantes deverá ser acessível aos Agentes Económicos e não simulado, pelo que deverá ser explorada a disponibilização de uma API, que permita aos Consumidores e Comerciantes registarem-se no sistema. Mais, após a conclusão do *onboarding* da solução pelo Peppol, a delegação do registo dos participantes no *Service Metadata Locator*, deve ser implementada ou garantida pelo *Service Metadata Publisher* que se venha a utilizar. Para além disso, poderá também ser interessante analisar a viabilidade de suportar outras Peppol *Business Interoperability Specifications*.

[1] https://github.com/phax/phase4/blob/master/LICENSE.txt
[2] https://www.saftpro.com/blog/vat-compliance-reports-post-audit-and-clearance/

No que diz respeito aos *Webhooks*, deve também ser explorada a disponibilização de funcionalidades que permitam ao Consumidor alterar ou reconfigurar o seu *endpoint*.

Para terminar, a aplicação *desktop* é simplista e está condicionada pelo suporte oferecido pelo *Flutter*. Alguns *plugins* para o *Windows* ainda são limitados e nem todos os *Widgets* disponíveis são 100% compatíveis. Portanto, como trabalho futuro a aplicação poderá ser reestruturada para melhorar alguns *Widgets* utilizados, como por exemplo os campos dos formulários, que de momento estão mais direcionados para aplicações *mobile* e consequentemente algumas interações não são tão intuitivas na aplicação *desktop*.

Bibliografia

[1] T. Aihkisalo e T. Paaso. "A Performance Comparison of Web Service Object Marshalling and Unmarshalling Solutions". Em: *2011 IEEE World Congress on Services*. 2011, pp. 122–129. DOI: 10.1109/SERVICES.2011.61.

[2] *All you need to know about Peppol?* Pagero. URL: https://www.pagero.pt/blog/all-you-need-to-know-about-peppol/ (acedido em 03/09/2021).

[3] S. Benda, J. Klımek e M. Nečask. "Using schematron as schema language in conceptual modeling for xml". Em: *Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling-Volume 143*. Citeseer. 2013, pp. 31–40.

[4] *Celerio - Documentation*. Jaxio. URL: https://www.jaxio.com/documentation/celerio/introduction.html (acedido em 03/09/2021).

[5] B. Christophe. *Electronic Invoicing: state of the art in Russia*. URL: https://www.generixgroup.com/en/blog/e-invoicing-russia (acedido em 24/01/2021).

[6] W. W. W. Consortium et al. "Xsl transformations (xslt) version 2.0". Em: (2007).

[7] I. Costa, C. Rasmussen e M. Forsberg. *Live Webinar3: The European norm and its content*. European Comission. Set. de 2017. URL: https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=44893107 (acedido em 24/01/2021).

[8] L. Davison. *Tax Cheats Are Costing the U.S. $1 Trillion a Year, IRS Estimates*. Bloomberg. Abr. de 2021. URL: https://www.bloomberg.com/news/articles/2021-04-13/tax-cheats-are-costing-u-s-1-trillion-a-year-irs-estimates (acedido em 03/09/2021).

[9] DNRE. *Fatura Eletrónica de Cabo Verde - Manual Técnico; versão: 1.0.0*. URL: https://efatura.cv/docs/manual (acedido em 24/01/2021).

[10] L. Dodds. "Schematron: Validating XML using XSLT". Em: (abr. de 2001). URL: http://ldodds.com/papers/schematron_xsltuk.html.

[11] K. Dooley. *Designing Large Scale Lans: Help for Network Designers*. "O'Reilly Media, Inc.", 2001.

[12] *eDelivery AS4 - 1.14*. European Comission. Out. de 2018. URL: https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eDelivery+AS4+-+1.14 (acedido em 24/01/2021).

[13] EDICOM. *A Fatura Eletrónica na América Latina. White Paper* (em português do Brasil). URL: https://edicom.pt/biblioteca-conteudo/white-papers/einvoicing-latam (acedido em 24/01/2021).

[14] EDICOM. *Aspetos a considerar para adaptar os seus sistemas à faturação global*. URL: https://edicom.pt/fatura-eletronica (acedido em 24/01/2021).

[15] EDICOM. *CFDI White Paper*. URL: https://cfdi.edicomgroup.com/en/resources/ten-years-e-invoicing-mexico/ (acedido em 24/01/2021).

[16] European Comission. *e-Invoicing Standardisation: Overview, issues and conclusions for future actions*. Set. de 2012. URL: https://ec.europa.eu/docsroom/documents/10472 (acedido em 24/01/2021).

[17] European Comission. *eInvoicing in Austria*. URL: https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eInvoicing+in+Austria (acedido em 24/01/2021).

[18] European Comission. *eInvoicing in Belgium*. URL: https://ec.europa.eu/cefdigital/wiki/display/CEFDIGITAL/eInvoicing+in+Belgium (acedido em 24/01/2021).

[19] J. Fialli e S. Vajjhala. "The Java architecture for XML binding (JAXB)". Em: *JSR Specification, January* (2003).

[20] Flutter. *Flutter architectural overview*. URL: https://flutter.dev/docs/resources/architectural-overview (acedido em 03/09/2021).

[21] Flutter. *Layouts in Flutter*. URL: https://flutter.dev/docs/development/ui/layout (acedido em 03/09/2021).

[22] *Global E-Invoicing Regulations*. SEEBURGER. URL: https://www.seeburger.com/platform/e-invoicing/global-e-invoicing-regulations/ (acedido em 03/09/2021).

[23] L. Guerin. *Telosys – a lightweight and pragmatic code-generator*. Mai. de 2018. URL: https://modeling-languages.com/telosys-tools-the-concept-of-lightweight-model-for-code-generation/ (acedido em 24/01/2021).

[24] E. R. Harold. *An Introduction to StAX*. Set. de 2003. URL: https://www.xml.com/pub/a/2003/09/17/stax.html (acedido em 24/01/2021).

[25] P. Helger. *Peppol practical - Technical Documentation*. URL: https://peppol.helger.com/public/locale-en_US/menuitem-docs-setup-ap (acedido em 03/09/2021).

[26] *How to set up a (Post-Award) PEPPOL Access Point (AP)*. PEPPOL. URL: https://peppol.eu/wp-content/uploads/2018/08/How-to-set-up-a-Post-Award-PEPPOL-Access-Point-.pdf (acedido em 24/01/2021).

[27] *Java XML Tutorial*. URL: https://www.tutorialspoint.com/java_xml/index.htm (acedido em 24/01/2021).

[28] J. Jenkov. *Java SAX vs. StAX*. Mai. de 2014. URL: http://tutorials.jenkov.com/java-xml/sax-vs-stax.html (acedido em 24/01/2021).

[29] B. Koch. *The e-invoicing journey 2019-2025*. Rel. téc. Billentis, set. de 2019. URL: https://www.billentis.com/e-invoicing_ebilling_market_report_EN.htm (acedido em 24/01/2021).

[30] KPMG. *The mandate is growing for e-invoicing adoption*. Ago. de 2018. URL: https://assets.kpmg/content/dam/kpmg/za/pdf/2018/August/mandate-for-e-invoicing-adoption-kpmg-tax.pdf (acedido em 24/01/2021).

[31] O. Lee. *3 Major Challenges of Using Paper-Based Invoices*. Set. de 2015. URL: https://www.invensis.net/blog/3-major-challenges-using-paper-based-invoices/ (acedido em 24/01/2021).

[32] D. Mackier. *Flutter Architecture - My Provider Implementation Guide*. Mai. de 2019. URL: https://www.filledstacks.com/post/flutter-architecture-my-provider-implementation-guide (acedido em 03/09/2021).

[33] M. Mernik, J. Heering e A. M. Sloane. "When and how to develop domain-specific languages". Em: *ACM computing surveys (CSUR)* 37.4 (2005), pp. 316–344.

[34] OASIS. *UBL 2 guidelines for customization*. First edition. Dez. de 2009. URL: http://docs.oasis-open.org/ubl/guidelines/UBL2-Customization1.0cs01.pdf (acedido em 24/01/2021).

[35] OASIS. *Universal Business Language Version 2.1*. Nov. de 2013. URL: http://docs.oasis-open.org/ubl/UBL-2.1.html (acedido em 24/01/2021).

[36] *OASIS ebXML Messaging Services Version 3.0: Part 1, Core Features*. OASIS Standard. Out. de 2007. URL: http://docs.oasis-open.org/ebxml-msg/ebms/v3.0/core/os/ebms_core-3.0-spec-os.html (acedido em 03/09/2021).

[37] *OpenPEPPOL Test and Onboarding*. OpenPEPPOL AISBL. Dez. de 2018. URL: https://peppol.eu/wp-content/uploads/2018/12/PEPPOL-Testbed-and-Onboarding_v1p2.pdf (acedido em 24/01/2021).

[38] Oracle. *javax.xml.validation (Java Platform SE 8)*. URL: https://docs.oracle.com/javase/8/docs/api/javax/xml/validation/package-summary.html (acedido em 24/01/2021).

[39] Oracle. *Why StAX?* URL: https://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/SJSXP2.html (acedido em 24/01/2021).

[40] E. Ort e B. Mehta. *Java Architecture for XML Binding (JAXB)*. Mar. de 2003. URL: https://www.oracle.com/technical-resources/articles/javase/jaxb.html (acedido em 24/01/2021).

[41] *PEPPOL - AS4 Profile*. OpenPEPPOL AISBL. Ago. de 2019. URL: https://docs.peppol.eu/edelivery/as4/specification/ (acedido em 03/09/2021).

[42] *PEPPOL - Compliance Policy v1.0*. OpenPEPPOL AISBL. Nov. de 2018. URL: https://peppol.eu/wp-content/uploads/2019/02/PEPPOL-Compliance-Policy_v1.0_PUBLISHED.pdf (acedido em 03/09/2021).

[43] *PEPPOL - Policy for use of Identifiers v4.0*. OpenPEPPOL AISBL. Jan. de 2019. URL: https://peppol.eu/wp-content/uploads/2017/12/PEPPOL_Policy-for-use-of-identifiers-300-11_certificates.pdf (acedido em 03/09/2021).

[44] *PEPPOL Transport Infrastructure - AS4 Profile*. OpenPEPPOL AISBL. 2017. URL: https://github.com/OpenPEPPOL/edec-specifications/blob/master/releases/as4/ICT-Transport-AS4_Service_Specification-1.0-2017-12-08.pdf (acedido em 03/09/2021).

[45] *PEPPOL Transport Infrastructure - Service Metadata Locator (SML) v1.01*. OpenPEPPOL AISBL. 2010. URL: https://docs.peppol.eu/edelivery/sml/ICT-Transport-SML_Service_Specification-101.pdf (acedido em 03/09/2021).

[46] *PEPPOL Transport Infrastructure - Service Metadata Publishing (SMP) v1.1.0*. OpenPEPPOL AISBL. Ago. de 2012. URL: https://docs.peppol.eu/edelivery/smp/ICT-Transport-SMP_Service_Specification-110.pdf (acedido em 03/09/2021).

[47] G. Poniatowski, A. Åšmietanka e M. Bonch-Osmolovskiy. *Study and Reports on the VAT Gap in the EU-28 Member States: 2020 Final Report*. CASE Reports 0503. CASE-Center for Social e Economic Research, 2020. URL: https://ideas.repec.org/p/sec/report/0503.html.

[48] C. Rasmussen e M. Forsberg. *Live Webinar8: XML validation mechanisms*. European Comission. Fev. de 2018. URL: https://ec.europa.eu/cefdigital/wiki/pages/viewpage.action?pageId=55891984 (acedido em 24/01/2021).

[49] C. Rasmussen e M. Forsberg. *SML and SMP*. European Comission. Fev. de 2018. URL: https://ec.europa.eu/cefdigital/wiki/download/attachments/82773320/%28eDelivery%29%28SML%2BSMP%29%28COD%29%281.10%29.pdf?version=1&modificationDate=1538384866455&api=v2 (acedido em 24/01/2021).

[50] E. Robertsson. *An Introduction to Schematron*. Nov. de 2003. URL: https://www.xml.com/pub/a/2003/11/12/schematron.html (acedido em 24/01/2021).

[51] SERES. *Global trends in electronic invoicing*. Mai. de 2019. URL: https://blog.groupseres.com/en/global-trends-in-electronic-invoicing-seres-global (acedido em 24/01/2021).

[52] W. Smith. *How Peppol IDs Work*. ecosio. Jul. de 2021. URL: https://ecosio.com/en/blog/how-peppol-ids-work/ (acedido em 03/09/2021).

[53] *Telosys templates*. Telosys. URL: https://www.telosys.org/templates.html (acedido em 03/09/2021).

[54] G. Tomassetti. *A Guide to Code Generation*. Mai. de 2018. URL: https://www.javacodegeeks.com/2018/05/a-guide-to-code-generation.html (acedido em 24/01/2021).

[55] E. Visser. "WebDSL: A Case Study in Domain-Specific Language Engineering". Em: vol. 5235/2008. Jan. de 2007, pp. 291–373. ISBN: 978-3-540-88642-6. DOI: 10.1007/978-3-540-88643-3_7.

[56] *What is Acceleo?* Acceleo. URL: https://www.eclipse.org/acceleo/overview.html (acedido em 03/09/2021).

[57] *What is AS4?* Justransform. URL: https://justransform.com/edi-essentials/what-is-as4/ (acedido em 03/09/2021).

[58] *What is Peppol?* SEEBURGER. URL: https://www.seeburger.com/info/what-is-peppol/ (acedido em 03/09/2021).

[59] *What is Peppol?* OpenPeppol. URL: https://peppol.eu/what-is-peppol/ (acedido em 03/09/2021).

[60] *XML Validation*. 2015. URL: http://www.usingxml.com/Basics/XmlValidation (acedido em 24/01/2021).

[61] R. Yigit. *B2G e-invoicing - overview across EU 2020*. Jul. de 2020. URL: https://www.linkedin.com/pulse/b2g-e-invoicing-overview-across-eu-2020-ridvan-yigit/?trackingId=%5C%2FKvDKmu%5C%2B4JF3BceXK4Mueg%5C%3D%5C%3D (acedido em 24/01/2021).

[62] R. Yigit. *Tax clearance and e-invoicing revolutionize the VAT compliance world*. Ago. de 2020. URL: https://www.linkedin.com/pulse/tax-clearance-e-invoicing-revolutionize-vat-compliance-ridvan-yigit/ (acedido em 24/01/2021).

I

Análise de Soluções

Fatura Eletrónica na Áustria

Autoridade Fiscal	Federal Ministry of Finance
Obrigatoriedade	O *e-Invoicing* atingiu a obrigatoriedade em 2014.
Formato da fatura	Suporta dois formatos: • ebInterface - Considerado o padrão nacional, a versão mais recente (ebInterface v.5) tem em consideração o standard Europeu; • PEPPOL-BIS.
Sistema	Pode ser utilizada qualquer plataforma desenvolvida por terceiros desde que esta esteja conectada com os serviços do USP (Portal dos serviços tributários). As faturas eletrónicas baseadas nos dois formatos descritos acima podem ser enviados através do USP, mas antes é necessário um processo de registo gratuito na plataforma. A entrega das faturas eletrónicas também não incorre em custos adicionais para os utilizadores. Existem cerca de 60 produtores de software de faturação capazes de produzir *e-Invoices* estruturados em conformidade com o país. Qualquer um destes necessita de se conectar ao USP para ter acesso aos serviços de autenticação necessários à submissão de faturas e para o e-Rechnung.gv.at. Este último corresponde a um método de submissão específico para o setor público. As faturas eletrónicas podem ser recebidas e processadas de duas maneiras: • Provenientes de *upload* direto; • Criadas manualmente através de *webform* ou *webservice*.
Assinatura Eletrónica	O USP fornece serviços de autenticação que suportam o processo de submissão das faturas e portanto estas não precisam de ser assinadas digitalmente.
Armazenamento	Após o envio, as faturas são armazenadas como PDFs gerados automaticamente. Depois seguem para o fluxo de processamento que é automatizado, exceto nos casos em que as organizações não têm sistemas de ERP adotados.

Figura I.1: Fatura eletrónica na Áustria

ebInterface

Campo	Descrição	Nome XML	Obrigatório
Número de Fatura	N/A	InvoiceNumber	Sim.
Data de Emissão	N/A	InvoiceDate	Sim.
Documento Original Cancelado	N/A	CancelledOriginalDocument	Não.
Documentos Relacionados	Campo composto, que serve para especificar um ou mais documentos ebInterface relacionados com o que se pretende emitir.	RelatedDocument	Não.
Informação Adicional	Contém qualquer informação adicional, que não conste em nenhum campo do formato definido.	AdditionalInformation	Não.
Detalhes de Entrega	N/A	Delivery	Não.
Cobrador	Campo composto que contém informações sobre o cobrador, nomeadamente: o seu contacto, morada, NIF, entre outros.	Biller	Sim.
Destinatário	Campo composto semelhante ao do emissor.	InvoiceRecipient	Sim.
Solicitante	Quando o destinatário da fatura não é o consumidor.	OrderingParty	Não.
Detalhes	Elemento composto pelos itens individuais que constam na fatura.	Details	Sim.
Descontos/Sobretaxas	Contém informações sobre descontos ou sobretaxas, que são do valor líquido total faturado.	ReductionAndSurchargeDetails	Não.
Imposto	Possui um resumo do imposto sobre vendas e quaisquer outros que se apliquem.	Tax	Sim.
Valor Bruto Total	N/A	TotalGrossAmount	Sim.
Valor Pré-pago	N/A	PrepaidAmount	Não.
Diferença de Arredondamento	Deve ser incluído na fatura quando existe diferenças de arredondamento nos cálculos.	RoundingAmount	Não.
Valor a Pagar	Corresponde ao valor bruto total, menos o valor pré-pago, mais a diferença de arredondamento.	PayableAmount	Sim.
Método de Pagamento	Especifica o método de pagamento.	PaymentMethod	Não.
Condições de Pagamento	Informações sobre as condições de pagamento, como descontos ou valor mínimo a ser cobrado.	PaymentConditions	Não.
Comentários	Observações sofre a fatura.	Comment	Não.

Figura I.2: Formato de representação da fatura eletrónica na Áustria

Fatura Eletrónica na Bélgica

Autoridade Fiscal	Federal Public Service Policy
Obrigatoriedade	O e-Invoicing atingiu a obrigatoriedade em 2019.
Formato da fatura	PEPPOL-BIS - Este formato tem como objetivo permitir harmonizar o processo de comunicação entre as empresas europeias e qualquer comprador do setor público durante o processo de contratação. A diretiva da UE, não estabeleceu um formato eletrónico específico para a troca de faturas. Mas, constatando o facto que a maioria dois países membros utiliza UBL, o PEPPOL serve-se dos seus mecanismos de extensão para personalizar o formato e adotá-lo às suas necessidades. A norma mais recente PEPPOL BIS *Billing* 3.0, pode ser utilizada sem alterações ou personalizada, consoante as especificações de cada país.
Sistema	A abordagem Belga não depende de uma plataforma em concreto, pois ao utilizar o modelo PEPPOL consegue cooperar com várias plataformas. No entanto, possui um portal web Mercurius, ligado à rede PEPPOL, que pode ser utilizado por qualquer entidade no país para receber faturas eletrónicas e aceder ao fluxo de faturação B2G. Este portal é mantido pela autoridade fiscal e disponibiliza adicionalmente uma interface web para intermediários e representantes de clientes, para que estes também tenham acesso a estas informações. O desenvolvimento deste portal cria alguma transparência e reduz custos de integração. Ou seja, as faturas eletrónicas podem ser emitidas por qualquer fornecedor com capacidade para enviar faturas à PEPPOL. Posteriormente, estas poderão ser consultadas no Portal Mercurius.
Assinatura Eletrónica	Não é obrigatória.
Armazenamento	Não existe informação muito concreta. O Portal Mercurius centraliza a receção das faturas para o setor público, de modo a uniformizar o método de submissão pelos operadores económicos. O mesmo não pode ser dito sobre o processamento que é efetuado, este fica à responsabilidade de cada entidade, que deverá adotar o melhor processo consoante o seu contexto.

Figura I.3: Fatura eletrónica na Bélgica

Fatura Eletrónica no México

Autoridade Fiscal	SAT - Servicio de Administración Tributaria
Obrigatoriedade	É obrigatória para 100% dos emissores e recetores. Foi desenvolvida em 2014 e tem vindo a sofrer alterações para corrigir as inconsistências detetadas ao longo do tempo.
Formato da fatura	CDFI
Sistema	O modelo CDFI simplifica o processo de emissão de faturas servindo-se de PACs (Provedores Autorizados de Certificação), que se responsabilizam pela validação das faturas e a sua posterior declaração ao sistema da SAT. Ou seja, o processo requer que o emissor contrate um PAC de modo a poder emitir faturas eletrónicas. Essencialmente, o elemento que confere a validade a uma fatura eletrónica é o "Timbre". Este é fornecido após validação do conteúdo e estrutura da fatura pelo PAC, em nome do governo mexicano. Caso a fatura seja aprovada, recai também sobre o PAC a responsabilidade de reportá-la ao sistema da SAT, para que a fatura seja disponibilizada ao emissor e ao destinatário através dos seus e-mails tributários.
Assinatura Eletrónica	A assinatura digital é obrigatória para todos os contribuintes e é utilizado um sistema proprietário. que serve-se de alguns campos extraídos da fatura emitida. Existe adicionalmente o "Timbre", que deverá ser fornecido por uma empresa que possua certificados emitidos pelo SAT, com a política para assinar CDFIs.
Armazenamento	As faturas emitidas devem ser armazenadas pelo menos 5 anos, isto é responsabilidade do emissor e do destinatário. Existem sistemas complementares que facilitam este armazenamento e oferecem garantias adicionais.

Figura I.4: Fatura eletrónica no México

CDFI

Campo	Descrição	Nome XML	Obrigatório
Versão	Indica a versão do CDFI a ser emitido.	Version	Sim.
Série	Especifica a série para controlo interno do contribuinte. Corresponde a uma sequência de caracteres com tamanho máximo 25.	Serie	Não.
Fatura	Especifica o número da fatura para controlo interno do contribuinte. Aceita uma sequência de caracteres de tamanho máximo 40.	Folio	Não.
Data e Hora de Emissão	Corresponde à data e hora de emissão do documento.	Fecha	Sim.
Assinatura Digital	Permite identificar e autentificar o emissor e garantir a integridade do documento. Corresponde a uma cadeia de caracteres em Base 64.	Sello	Sim.
Forma de Pagamento	N/A	FormaPago	Não.
Número de Certificado	Descreve o número de série do certificado, concedido pelo SAT, que é utilizado para assinar o documento.	NoCertificado	Sim.
Certificado	Conteúdo do certificado utilizado para assinar o documento em Base 64.	Certificado	Sim.
Condições de Pagamento	Expressa as condições comerciais que se aplicam ao pagamento do documento.	CondicionesDePago	Não.
Subtotal	N/A	SubTotal	Sim.
Desconto	Representa o valor total dos descontos aplicáveis antes dos impostos.	Descuento	Não.
Moeda	Representação do tipo de moeda, tem de ser compatível com a especificação ISO 4217.	Moneda	Sim.
Taxa de Conversão	Representa a taxa de conversão de acordo com a moeda utilizada.	TipoCambio	Não.
Total	Representa a soma do subtotal, menos os descontos aplicáveis e os impostos retidos, mais as contribuições recebidas.	Total	Sim.
Tipo de Comprovativo	N/A	TipoDeComprobante	Sim.
Método de Pagamento	Especifica o método de pagamento que se a aplica ao documento, no caso de ser uma fatura.	MetodoPago	Não.
Local de Emissão	Código postal do local de emissão do documento.	LugarExpedicion	Sim.
Confirmação	Utilizado quando são emitidos documentos com grande valor, ou com códigos de moeda fora dos estabelecidos.	Confirmacion	Não.

Figura I.5: Formato de representação da fatura eletrónica no México

Fatura Eletrónica em Cabo Verde

Autoridade Fiscal	Direção Nacional de Receitas do Estado
Obrigatoriedade	O lançamento da fatura eletrónica está agendado para final de 2020. Irão ser obrigados a processar faturas eletrónicas todos os sujeitos passivos, previamente credenciados pelas Autoridades Tributárias.
Formato da fatura	DFE
Sistema	Para emitir faturas eletrónicas será necessário adquirir software de faturação. Contudo, nem todos os contribuintes terão esta possibilidade e portanto a DNRE irá disponibilizar um emissor público e gratuito para ajudar os pequenos contribuintes. Adicionalmente, quem já possui sistemas de faturação deverá garantir que estes conseguem adotar as especificações técnicas publicadas pela DNRE, para poder gerar faturas de acordo com o formato DFE. Após a geração da fatura esta deve ser assinada digitalmente e enviada através de canais de transmissão seguros para o sistema da DNRE para validação e autorização. Ou seja, após a receção das faturas estas serão alvo de validações antes de serem autorizadas. Este processo envolve várias fases, em que uma delas consiste em verificar a assinatura da fatura emitida. Faz também parte deste processo a verificação da estrutura e os tipos de dados do XML de acordo com o XSD da fatura. A plataforma disponibilizada pela DNRE, para processar e armazenar as fatura emitidas, irá possuir mecanismos de alta disponibilidade e contingência para acomodar os casos em que os sistemas do contribuinte ou da DNRE apresentem condições que impossibilitem o normal funcionamento das mesmas. Por exemplo, a falta de eletricidade e/ou internet. Será também nesta plataforma que serão disponibilizadas um certo conjunto de operações a realizar sobre as faturas, nomeadamente: a sua consulta, o seu download, entre outras.
Assinatura Eletrónica	As faturas deverão encontrar-se assinadas digitalmente, pelo que cada contribuinte deverá adquirir um Certificado Digital válido dentro da hierarquia de certificados digitais da Infraestrutura de Chaves Públicas de Cabo Verde.
Armazenamento	As faturas serão armazenadas na plataforma disponibilizada pela DNRE após serem validadas e aprovadas para uso dos contribuintes, estes também as poderão descarregar para armazenamento local.

Figura I.6: Fatura eletrónica em Cabo Verde

DFE

Campo	Descrição	Nome XML	Obrigatório
IUD	Identificador Único do formato DFE. Constituído por 45 dígitos. Corresponde à concatenação de alguma informação nomeadamente: a data de emissão, o tipo do DFE, entre outros.	IUD	Sim.
Versão	Versão do documento fiscal.	Version	Sim.
Modo de Emissão	Modo como o DFE é emitido para efeitos de autorização. Existem 3 modos: Online; Offline; Off.	IssueMode	Sim.
Indicador de Consumidor Final	N/A	?	Sim.
LED	Possuí tamanho fixo de 4 dígitos e indica os locais/sistemas que emitem/geram os documentos eletrónicos.	LED	Sim.
Série	Código utilizado pelo contribuinte para classificar a numeração dos DFE. Formado no máximo por 10 carateres alfanuméricos.	Serie	Sim.
Referência de Encomenda	Código que permite a um consumidor controlar a encomenda de bens/serviços.	OrderReference	Não.
Data Geradora de Imposto	Data em que o imposto associado ao DFE é considerado como devido, de acordo com a legislação do imposto.	TaxPointDate	Não.
Data e Hora de Emissão	N/A	IssueDateTime	Sim.
Código de Moeda	Código de moeda usado no DFE. Se for omitido é considerado o escudo Cabo-Verdiano.	CurrencyCode	Não.
Contingência	Apenas necessário quando o documento não for emitido online. É um campo composto que agrega os campos: LED; IUC (Identificador Único de Contingência); Data de Emissão; Hora de Emissão.	• LED; • IUC; • IssueDate; • IssueTime	Perante uma contingência, todos são obrigatórios exceto o IssueTime.
Software	Atributo composto pelo nome do software e a versão que o emissor do DFE utiliza.	• Name: • Version	Sim.
Fornecedor	Descreve o emissor e é um campo composto por: NIF; Nome; Endereço; Contactos.	• TaxID; • Name; • Address; • Contacts	Sim.
Cliente	Muito semelhante ao Fornecedor.	• TaxID; • Name; • Address; • Contacts	Sim.
Itens de Serviços/Produtos	N/A	Tabela: Itens(Produtos/Serviços)	Sim.
Totais	Campo composto por: Total Líquido; Desconto; Total de Imposto; Soma de totais.	• NetTotal; • Discount; • TaxTotal; • GrandTotal	Sim.

Figura I.7: Formato de representação da fatura eletrónica em Cabo Verde

II

Processamento de XML - Comparação das especificações

Características	StAX	SAX	DOM
Tipo de API	*Streaming* com *Pull Mode*	Streaming com *Push Mode*	Árvore em memória
Facilidade de utilização	Fácil	Média	Fácil
Suporta XPATH	Não	Não	Sim
Eficiência em termos de CPU e memória	Boa	Boa	Varia consoante o tamanho do ficheiro
Tipo de Leitura	Sequencial	Sequencial	Bilateral
Permite escrever XML	Sim	Não	Sim
Operações CRUD no XML	Não	Não	Sim
Outras características	• Permite *subparsing* do documento, que consiste em permitir que um determinado bloco do documento seja processado de uma maneira diferente dos restantes blocos; • Melhor suporte para processamento paralelo dos documentos.	• Suporte para validação através do XSD; • Permite não processar o documento todo; • Oferece melhor suporte para *error handling* através de *callbacks*.	• Uma vez carregado em memória o documento pode ser utilizado mais que uma vez; • Suporte para validação através de XSD.
Desvantagens	• Não suporta validação através do XSD; • Não existe *random access* ao documento.	• O documento não é mapeado diretamente para objetos, sendo necessário tratar os eventos e criar os objetos explicitamente. • Não existe *random access* ao documento.	• Se o ficheiro for muito grande pode tornar-se lento; • O mesmo acontece em situações onde apenas queremos aceder a certas partes do documento e não tencionamos ler o documento ate ao fim.

Figura II.1: Características das especificações[27, 28, 39]

III

Processamento de UBL 2.1

As especificações da secção 2.2 e as suas respetivas implementações já nos permitem de certo modo processar um documento XML e fazer o seu mapeamento para objetos Java. No entanto, o UBL possui regras próprias de *design* e nomenclatura o que resultou no desenvolvimento de algumas ferramentas que tornam mais transparente o mapeamento das *Business Information Entities* para objetos Java.

Na seleção da solução preferida de entre as diferentes soluções analisadas, os critérios de seleção considerados foram os seguintes:

- **Benefícios**: a solução deve apresentar o melhor desempenho e eficiência possível durante o processamento do documento;
- **Manutenção**: a solução deve ser fácil de manter em projetos futuros, fatores como a possibilidade de alterar ou incrementar o código devem ser tidos em conta, assim como a quantidade e qualidade de documentação existente para a ferramenta;
- **Reutilização**: a ferramenta deve ser flexível;
- **Fiabilidade dos dados**: É absolutamente crítico que os dados fornecidos pela ferramenta reflitam o documento que está a ser processado para evitar erros de validação.

Descrição da solução	Ferramenta	Licenciamento	Vantagens	Desvantagens
Processamento de documentos UBL 2.1	ph-ubl	Apache-2.0 License	• Biblioteca Java que permite ler e escrever documentos UBL 2.0, 2.1, 2.2 e 2.3; • É atualizada com frequência e possui boa documentação; • Fornece objetos Java que mapeiam para as *Business Information Entities*; • Permite validar a estrutura do documento através do XSD; • *Open Source*.	• A validação estrutural pressupõe os documentos base disponibilizados na biblioteca, o que pode tornar um pouco mais lenta a validação por existirem elementos opcionais que não estão presentes para uma determinada instância. • Já tem todas as *Business Information Entitities* geradas, apesar destas puderem não ser utilizadas.
	XBuilder	Eclipse Public License 2.0	• Permite a criação de documentos UBL de uma maneira simples através do *Builder Pattern;* • Suporta de uma maneira simples a assinatura dos documentos; • Possui boa documentação com exemplos de utilização; • *Open Source*.	• Não permite ler documentos UBL; • Não permite validar a estrutura do documento através do XSD; • Atualmente apenas suporta a instância de Perú e não permite alterar o XSD para criação do documento.
	UBL4J	GNU General Public License 3.0	• Permite ler um *invoice* UBL e transformá-lo num formato mais legível como pdf; • Permite transformar um *order* UBL num *invoice* UBL de forma transparente. • Suporta métodos para persistir os documentos numa base de dados de forma transparente e suporta algumas *queries* sobre os mesmos.	• Baseia-se nos termos UBL, mas não especifica a versão utilizada; • Não permite criar documentos XML; • O mecanismo que suporta as validações é limitado.

Figura III.1: Análise de ferramentas de processamento UBL 2.1

IV

Validação de UBL 2.1

Na seleção da solução preferida de entre as diferentes soluções analisadas, os critérios de seleção considerados foram os seguintes:

- **Benefícios**: a solução deve apresentar o melhor desempenho e eficiência possível na validação dos documentos;
- **Volume**: a solução deverá *thread-safe* e suportar a validação de documentos em simultâneo;
- **Manutenção**: a solução deve ser fácil de manter em projetos futuros, fatores como a possibilidade de alterar ou incrementar o código devem ser tidos em conta, assim como a quantidade e qualidade de documentação existente para a ferramenta;
- **Reutilização**: a ferramenta deve ser flexível;
- **Fiabilidade dos dados**: É absolutamente crítico que os relatórios de validação sejam precisos e identifiquem explicitamente que campos e regras estão em incumprimento.

Descrição da solução	Ferramenta	Licenciamento	Características	Desvantagens
Validação em Schematron de instâncias de faturas UBL 2.1 em *run-time*.	ph-schematron	Apache License 2.0	• Biblioteca Java para validar XML via ISO Schematron; • Disponibiliza um tipo de verificação básico que indica se um documento cumpre as regras Schematron e um tipo de verificação mais complexo, que tem como resultado um documento SVLR (Schematron *Validation Report Language*), onde é descrito de forma precisa e em caso de falha as regras que levaram à rejeição dum documento; • A abordagem sugerida consiste em pré-processar o ficheiro Schematron, através de *scripts* XSLT. Deste modo, ele próprio torna-se um *script* XSLT direcionado para a validação dos documentos. A biblioteca oferece de raiz um *plugin* que nos permite fazer esta transformação, mas a sua utilização não é obrigatória; • Como alternativa à validação em XSLT, a biblioteca oferece uma implementação de validação em Schematron puro, onde os mesmo resultados podem ser alcançados a nível de verificações, mas com a grande vantagem de não ser necessário pré-processar o ficheiro; • Boa documentação e constante manutenção.	• Apenas suporta a versão ISO Schematron; • O processo de pré-compilação deve ser feito em *build-time*, pois consome bastante tempo e pode demorar alguns minutos para um ficheiro Schematron de tamanho médio. Contudo, o ficheiro resultante desta transformação pode ser *cached* e reutilizado para várias utilizações sucessivas; • Em Schematron puro as validações encontram-se limitadas, pois não é possível utilizar funções XSLT e a expressividade está limitada a expressões XPath.
Validação em XSD e Schematron de instâncias de faturas UBL 2.1 em *run-time*.	phive	Apache License 2.0	• Mecanismo genérico de validação de documentos comerciais; • Envolve essencialmente diferentes XML *Schemas* e Schematrons ordenados sobre algumas restrições que possibilitam a validação de documentos XML; • É composto por três sub-módulos: **phive-api** que disponibiliza uma API genérica e independente da lógica da validação, **phive-engine** que possui o mecanismo de validação e o **phive-json** que oferece classes que permitem transformar os resultados das validações em JSON, o que é bastante útil no desenvolvimento de APIs como no caso do e-Invoicing; • Os artefactos de validação são identificados de uma maneira semelhante às Maven *coordinates*; • Tem utilização comercial num produto de integração B2B (ecosio); • Disponibiliza exemplos de artefactos de validação, inclusive da CIUS utilizada em Portugal; • Boa documentação e manutenção.	• Precisa de pelo menos 1MB para o tamanho da *stack*, o que pode ser problemático ao correr numa JVM sobre um arquitetura de 32 bits. Pode ser resolvido em *run-time*, através de comandos; • É necessário ter o *engine* e a biblioteca de regras na aplicação; • Os artefactos de validação têm de ser registados inicialmente na classe **ValidationExecutorSetRegistry**, que é *thread-safe* e pode ser instanciada como um *Singleton*; • Muito mais complexo de utilizar que o **ph-schematron**, mas permite validar tanto o XSD como o Schematron com apenas um *engine*.
	KoSIT Validator	Apache License 2.0	• Mecanismo de validação para documentos XML em vários formatos através de ficheiros XSD e Schematron; • De um modo geral, começa por identificar o formato do XML e de seguida utiliza o *schema* e as regras Schematron para validar o documento; • Gera um relatório após a validação que pode ser customizado; • O validador à partida não tem qualquer conhecimento sobre os documentos XML e não possui regras próprias de validação; • Serve-se de cenários, definidos em XML, que configuram o processo de validação e podem ser desenvolvidos isoladamente; • O validador pode ser utilizado de três maneiras diferentes: como uma *standalone application*, como uma biblioteca integrado numa aplicação e como um daemon através de uma interface http; • A manutenção é boa.	• A documentação é qualitativamente inferior em comparação com as ferramentas anteriores; • Está pré-configurado para os cenários de faturação Alemão; • Curva de aprendizagem é maior quando comparado com as outras duas alternativas; • Não pode ser utilizado, pois a versão do Java utilizada é superior à do produto **SAFTPRO**.

Figura IV.1: Análise de ferramentas de validação UBL 2.1

V

Peppol IDs[1]

Os três principais tipos de identificadores, encontrados nas mensagens trocadas via Peppol, são[52]:

- **Peppol *Participant Identifiers*:** identificadores únicos utilizados para estabelecer a entidade das partes conectadas à rede Peppol;
- ***Document Type Identifiers*:** utilizados para indicar que tipo de documento consta no *payload* de uma mensagem;
- ***Process Identifiers*:** especificam o processo a ser considerado para uma determinada transação na rede.

V.1 Peppol *Participant Identifiers*

O Peppol emprega um sistema federado, assente no formato ISO 15459[2], para identificar os utilizadores da rede.

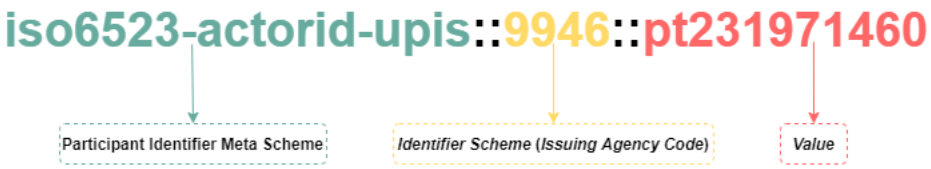


Figura V.1: Sintaxe do Peppol *Participant Identifier*

A sintaxe para este tipo de identificadores está representada na figura V.1 e pode ser dividida em 3 partes:

- ***Participant Identifier Meta Scheme*:** tem valor fixo e é utilizado como prefixo de todos os *Peppol Participant Identifiers*. Como o valor é constante, o prefixo pode ser omitido;

[1] Subsecção estruturada com base em [43].
[2] https://www.iso.org/standard/51284.html

- ***Identifier Scheme* (*Issuing Agency Code*):** representa a *Issuing Agency* que está a ser utilizada para o valor do identificador. Este código deve constar na sublista[3] definida pelo Peppol e no exemplo acima, o valor "9946" corresponde ao número de identificação fiscal português;
- ***Value*:** representa o valor único para um determinado *Identifier Scheme*.

O valor dos identificadores é *case insensitive* mesmo que o esquema exija distinção entre maiúsculas e minúsculas. Ou seja, "0088:abc" é igual a "0088:ABc", mas "0088:abc" não é igual a "9946:abc".

V.2 *Document Type Identifiers*

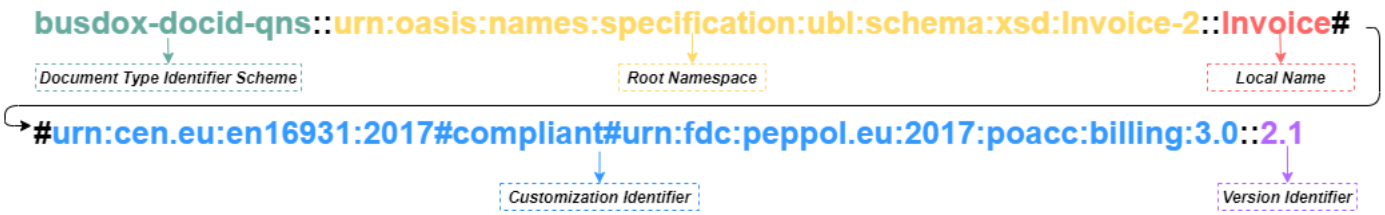


Figura V.2: Sintaxe do Peppol *Document Type Identifier*

O formato resulta do agregado dos seguintes conceitos:

- ***Document Type Identifier scheme*:** tem valor fixo e serve um propósito semelhante ao *Participant Identifier Meta Scheme*;
- ***Syntax Specific Identifier*:** explícita a síntaxe (p.e. XML) e o formato (p.e. UBL *Invoice*) do documento que está a ser trocado. Para documentos XML, o *namespace* do *schema* que define o elemento *root* e o nome do elemento *root* são concatenados através do delimitador "::";
- ***Customization Identifier*:** distingue a especificação que contem o conjunto de regras relacionadas com o conteúdo semântico, cardinalidade e regras de negócio às quais a informação do documento deve obedecer. Para o caso do Peppol, os requisitos estão documentados nas Peppol *Business Interoperability Specifications* (mais detalhes em 2.5.3);
- ***Version Identifier*:** caracteriza a versão do documento baseado nas convenções de sintaxe e formato desse tipo específico de documento.

[3] https://docs.peppol.eu/edelivery/codelists/v7.5/Peppol%20Code%20Lists%20-%20Participant%20identifier%20schemes%20v7.5.html

V.3 *Process Identifiers*

Os *Process Identifiers*, também conhecidos nas Peppol BIS por *Profile Identifiers*, definem as orquestrações nas quais os documentos são trocados. Estes identificadores são compostos por um *Process Identifier Scheme* e um *Process Identifier Value*.

No caso do Peppol BIS *Billing* 3.0, o *Process Identifier Scheme* é "cenbii-procid-ubl" e o *Process Identifier Value* é "urn:fdc:peppol.eu:2017:poacc:billing:01:1.0".

Figura V.3: Sintaxe do Peppol *Process Identifier*

VI

AS4[1]

VI.1 Estrutura das mensagens

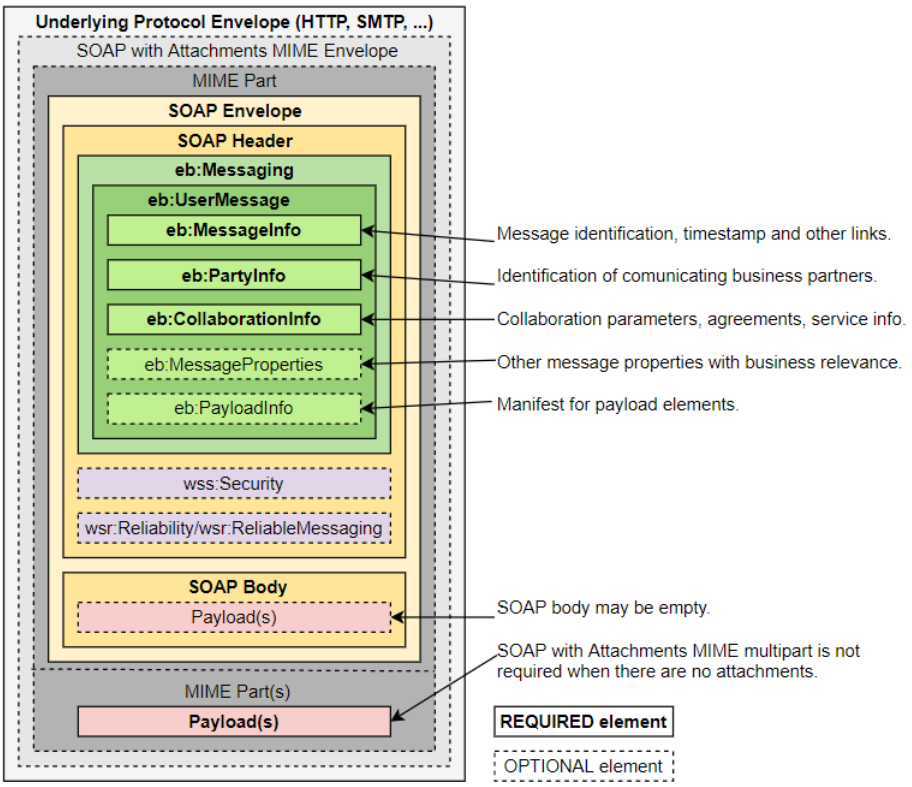


Figura VI.1: Estrutura da mensagem AS4 [12]

O AS4 utiliza envelopes SOAP codificados em UTF-8 e MIME para a estrutura das suas mensagens. Como é possível verificar na figura VI.1, a mensagem consiste num MIME *multipart* cuja primeira *part* corresponde a um envelope SOAP, que é utilizado para encapsular uma `UserMessage` ou uma `SignalMessage`.

[1] Subsecção estruturada com base em [12, 44].

A UserMessage permite transmitir o *payload* que é interpretado pelo *Consumer* e a SignalMessage, como o nome indica, transporta uma mensagem para ser interpretada como sinal ao nível dos MSH. No caso do envelope SOAP encapsular uma UserMessage, podemos encontrar os seguintes elementos:

- **MessageInfo:** identifica univocamente a mensagem através de um MessageID e *timestamp*;
- **PartyInfo:** identifica o emissor e destinatário da mensagem;
- **CollaborationInfo:** composto pelos elementos AgreementRef, Service, Action e ConversationId que são utilizados para descrever o contexto de negócio que regula a troca de mensagens;
- **MessageProperties:** disponibiliza informação adicional sobre o contexto da troca de mensagens;
- **PayloadInfo:** referencia o *payload* que consta no SOAP *body* ou nos *attachments*.

Segundo a especificação ebMS 3.0, as implementações devem permitir configurar o valor do elemento AgreementRef e o seu atributo *type* no parâmetro PMode.Agreement. Deste modo, é possível selecionar o P-Mode e estabelecer que elementos presentes no *header* ebMS 3.0 devem ser validados pelos *Receiving* MSH.

Na rede Peppol a troca de mensagens é regulada pelos *Transport Infrastructure Agreements* e portanto o parâmetro PMode.Agreement deve ter o valor "urn:fdc:peppol.eu:2017:agreements:tia:ap_provider" [41]. Contudo, o atributo opcional do elemento AgreementRef que permite selecionar o P-Mode não deve ser utilizado, pois nada impede que os *Access Points* da rede Peppol utilizem um P-Mode mais abrangente para a receção das mensagens.

No que diz respeito ao *payload*, a especificação ebMS 3.0 alega que pode ser empacotado ao nível do SOAP *body* ou num MIME *attachment*. Porém, o perfil AS4 do Peppol utiliza uma funcionalidade de compressão que se aplica apenas aos *payloads* presentes nos MIME *attachments*, o que descarta a possibilidade de transportá-los no SOAP *body*.

Por último, na presença de uma UserMessage o *header* Content-Disposition, descrito na secção 5.1.9 do perfil AS4 base, não deve ser utilizado para indicar o nome do documento que consta no *payload*. Caso uma implementação do AS4 não consiga omitir este cabeçalho o seu valor deverá ser fixo e igual a "payload.xml" [41].

Relativamente aos envelopes SOAP que encapsulam uma SignalMessage, podemos encontrar os seguintes elementos:

- **MessageInfo:** neste caso possuí o MessageId e a *timestamp* para a UserMessage que esta mensagem referencia;
- **Um dos seguintes elementos:**
  - Uma *Receipt*, que possui uma prova de não-repúdio relacionada à receção de uma UserMessage;

– Um *Error*, que é definido por um código e uma descrição apropriada de acordo com a especificação ebMS 3.0;
– Ou uma *Pull Request*, que no caso do Peppol não é utilizada pois a funcionalidade designada *Message Particion Channel* não é necessária entre *Access Points*[41].

VI.2 Tratamento de Erros

Quanto ao tratamento de erros, a especificação ebMS 3.0 indica que devem ser comunicados sincronamente ao *Sending* MSH e potencialmente reportados ao *Consumer* e *Producer*.

Todavia, as interfaces entre o *Producer* e o *Sending* MSH, assim como o *Receiving* MSH e o *Consumer* não estão definidas e ficam ao critério de quem disponibiliza a solução. Por esse motivo, no perfil AS4 do Peppol recomenda-se que os erros gerados tanto no envio como na receção de documentos comerciais sejam reportados diretamente a quem opera o *Access Point*. Deste modo, os erros podem ser resolvidos fora da banda o que ajuda a mitigar eventuais problemas de interoperabilidade entre os diferentes componentes da rede.

Ainda assim, quando um *Access Point* recebe um documento, deve verificar se serve o participante que consta como destinatário em relação ao *Document Type Identifier* e *Process Identifier*. Caso contrário, não poderá entregar a mensagem com o documento e deverá emitir um erro, de acordo com a especificação ebMS 3.0, que adverte o facto do participante não ser servido pelo *Access Point*.

Para a situação descrita o código de erro é "`EBMS:0004`" (utilizado para descrever erros não contemplados pelo ebMS 3.0) e a sua descrição tem o valor "`PEPPOL:NOT_SERVICED`"[41].

Quando se comunica um erro ao *Sending* MSH da mensagem errónea, este deve ser assinado sempre que for possível atribuir um P-Mode à mensagem recebida. Não obstante, no Peppol os *Access Points* têm a obrigatoriedade de aceitar tanto os erros assinados como os não assinados. Pois, podem existir erros impossíveis de relacionar com um `P-Mode` (que define o algoritmo de assinatura e certificado a utilizar).

Por último, quando uma mensagem de erro está assinada o *Access Point*, após a sua validação, deverá reportar o erro localmente e não tornar a responder com uma mensagem de erro.

VI.3 Fiabilidade

Para garantir a fiabilidade na troca de mensagens o AS4 utiliza dois tipos de *acknowledgments*, enviados sincronamente, para cada tipo de mensagens:

- ***Receipts* (*acknowledgements* positivos):** indicam que a mensagem foi processada com sucesso pelo *Receiver* MSH.

- ***Errors* (*acknowledgements* negativos):** indicam que o *Receiver* MSH encontrou um problema durante o processamento de uma mensagem.

Para os participantes da rede Peppol é importante ter a garantia de que os documentos são entregues com sucesso aos seus parceiros comerciais. Na especificação AS4 base, um *acknowledgment* positivo não reflete necessariamente que a mensagem tenha sido entregue com sucesso ao *Consumer*, i.e., a entrega do *acknowledgment* pode acontecer antes do *Receiving* MSH entregar a mensagem à *business application* do *Consumer*. O Peppol contorna esta limitação através dos seus TIAs, onde se declara que um *Access Point* age em nome dos participantes que serve. Deste modo, a receção de uma *receipt* indica que o documento foi entregue ou será entregue ao parceiro comercial dentro de instantes.

Para sustentar o princípio "*Once and only once delivery*" os *acknowledgements* são complementados com mais duas funcionalidades. Uma delas é implementada ao nível do *Sending* MSH e intitula-se *Reception Awareness*. O seu propósito é detetar a omissão de *receipts* para uma determinada *UserMessage*, dentro de um intervalo de tempo pré-determinado e comunicá-la ao *Producer*.

Quando a *receipt* não é emitida, o erro gerado deverá ter o código "`EBMS:0301`" e a descrição "*Missing Receipt*". No entanto, a especificação ebMS 3.0 não é muito explícita sobre que erro deve ser emitido quando existe um problema na entrega. Ou seja, quando o *Receiving* MSH responde com uma `SignalMessage` cujo *payload* é um *Error*.

Para situações semelhantes à de cima, o perfil AS4 do Peppol age em conformidade com o perfil CEF *e-Delivery* 1.14, que dita que o erro retornado ao *Sending* MSH deverá ter código "`EBMS:0202`" e descrição "*Delivery Failure*".

A deteção de omissões ao nível do *Sending* MSH permite complementá-lo com um mecanismo de *Retry*, para que se volte a enviar uma mensagem quando a sua *receipt* não chega. Existem diferentes políticas de *retry* que podem ser implementadas e todas elas são caracterizadas pelo número de *retries*, assim como o intervalo de tempo que se espera entre *retries*.

A conjunção da *Reception Awareness* com o mecanismo de *Retry* assegura que o *Sending* MSH tem a certeza que determinada mensagem foi recebia com sucesso pelo *Receiving* MSH. Porém, a potencialidade de existir um *retry* pode introduzir um duplicado na rede. Basta considerar o cenário onde, devido à elevada latência da rede, uma mensagem chega ao *Receiving* MSH fora do intervalo de tempo estipulado, originando uma nova tentativa por parte do *Sending* MSH.

O problema descrito acima é resolvido com a ajuda da funcionalidade *Duplicate Detection*, implementada através dos identificadores das mensagens (`MessageId`), ao nível do *Receiving* MSH. Idealmente, esta deteção leva à eliminação dos duplicados no *Receiving* MSH. Mas caso a implementação não o faça explicitamente, deverá no mínimo comunicar o evento ao *Consumer*, para que este não considere a mesma mensagem duas vezes.

Em relação ao perfil AS4 do Peppol, como o *Receiver Access Point* engloba o *Receiving* MSH e o *Consumer*, os duplicados são filtrados ao nível do *Access Point* e não chegam a

ser comunicados aos participantes.

VI.4 Segurança

O perfil AS4 do Peppol cobre apenas a segurança na troca de mensagens entre *Access Points*. Como tal, é responsabilidade do provedor de serviços e dos participantes assegurar que a troca de informação entre ambos está suficientemente protegida.

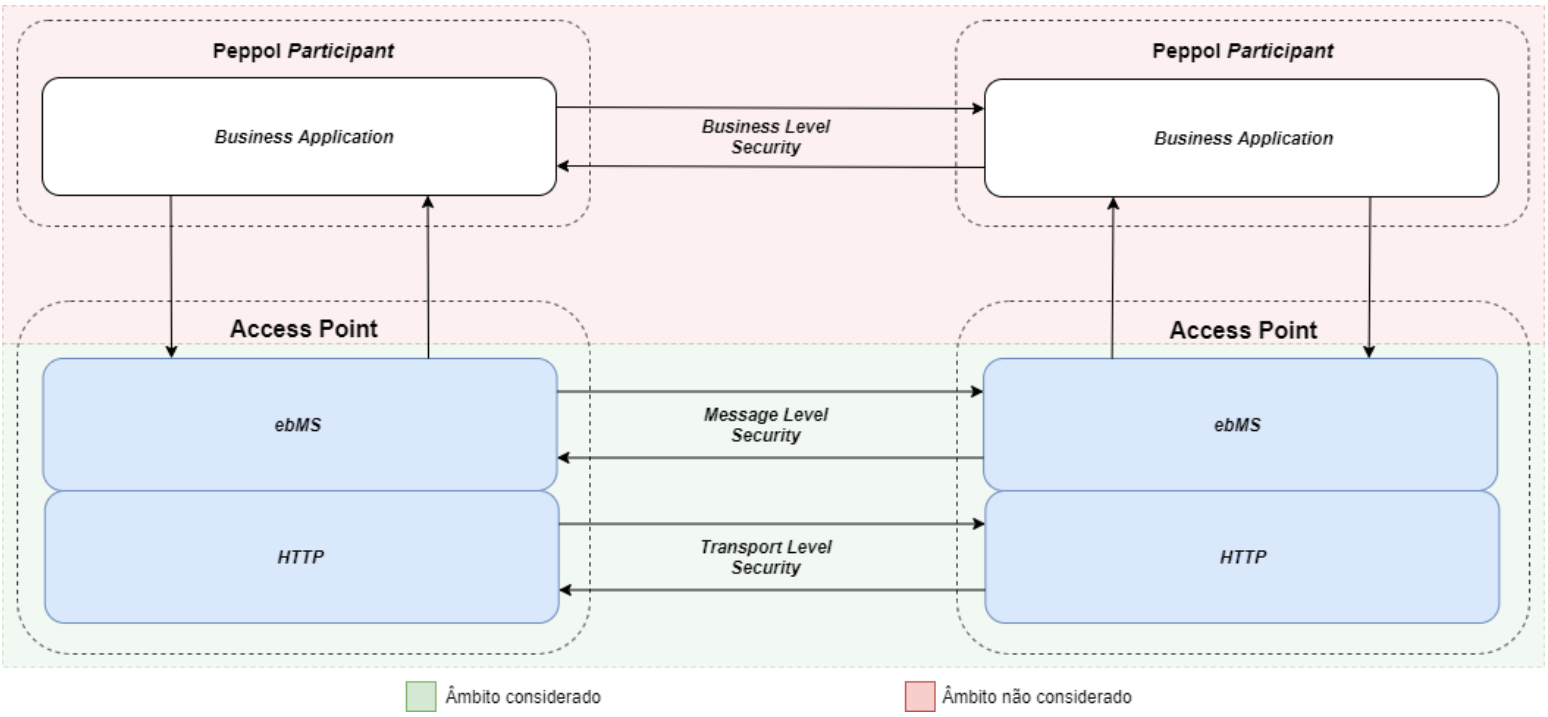


Figura VI.2: Peppol - Âmbito de segurança

Relativamente à confidencialidade da informação, esta pode ser assegurada no protocolo de comunicação através de *Transport Layer Security* (TLS)[2], ou ao nível do protocolo de mensagens através de técnicas criptográficas presentes no *standard* WS-Security.

Se for utilizado TLS, as versões anteriores à 1.1 deverão ser descartadas devido às sua potenciais vulnerabilidades e este poderá ser deliberado para um outro componente da infraestrutura (p.e., *proxy*), pelo que o AS4 MSH não necessita de o oferecer explicitamente.

No protocolo de mensagens a confidencialidade é obtida através de criptografia assimétrica. Ou seja, o *Sending* MSH cifra a mensagem com a chave pública do *Receiving* MSH, para que apenas este a possa ler com a sua chave privada. A cifra é aplicada ao SOAP *body* e a todos os *attachments* presentes na mensagem.

[2] https://datatracker.ietf.org/doc/html/rfc2246

Quando não é possível decifrar uma mensagem, o MSH deverá enviar uma `SignalMessage` com um *Error* ao *Sending* MSH, de acordo com a configuração presente no P-Mode. Em simultâneo, a mensagem deve ser assinada para que a sua integridade e não-repúdio sejam assegurados. Mais uma vez, a especificação ebMS 3.0 disponibiliza mecanismos que cobrem estas conveniências e poderá ser redundante concretizá-las também ao nível do protocolo de comunicação entre *Access Points*.

A integridade das mensagens pode ser garantida se o repúdio por parte de quem as transmite for evitado. No AS4 isto significa que, o *Receiving* MSH ao receber uma mensagem consegue identificar a fidedignidade de quem a enviou e que a mesma não sofreu alterações. Esta necessidade é concretizada através da funcionalidade de assinatura disponibilizada pelo WS-Security aplicada à `UserMessage`. Mais concretamente, a mensagem é assinada através de criptografia assimétrica com a chave privada do *Sending* MSH e de acordo com o perfil AS4, o *input* da assinatura são os ebMS *Messaging Headers*, o SOAP *body* e todos os *attachments*.

Para verificar a integridade da mensagem, o *Receiving* MSH obtém o certificado do *Sending* MSH através de um `BinarySecurityToken` embutido na mensagem, que contém o certificado utilizado para assinar a mensagem, como especificado no X.509 *Token Profile* do WS-Security.

No Peppol, a assinatura é efetuada pelo *Access Point* com o certificado emitido pela Public Key Infrastructure (PKI) do Peppol e disponibilizado na fase de registo no SMP. Deste modo, o *Receiver Access Point* pode proceder à validação das assinaturas através da Peppol PKI CA, sem necessitar de saber previamente o certificado do *Sender Access Point*.

Para terminar, é igualmente importante garantir que o *Sending* MSH está seguro de que o *Receiving* MSH recebeu uma determinada mensagem, sem que esta tenha sido alterada durante a troca. O não-repúdio dos *acknowledgments* pode lograr-se com a presença, nos ebMS *Messaging Headers*, dos *hashes* dos *payloads* assinados presentes na mensagem original mais a assinatura das *receipts* retornadas ao *Sending* MSH.

VII

SAFTPRO - Visão geral do produto

Este capítulo sintetiza a estrutura inicial do SAFTPRO, em relação aos componentes que o constituem, as funcionalidades fornecidas e as tecnologias envolvidas na sua conceção.

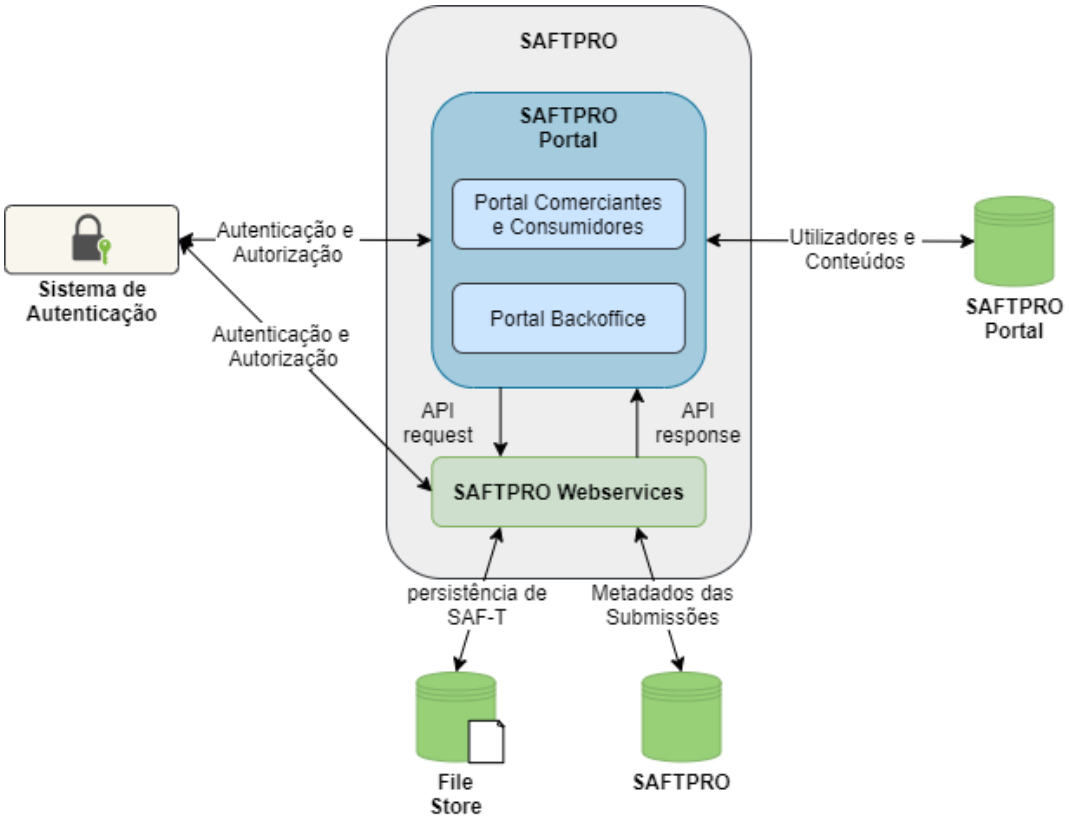


Figura VII.1: SAFTPRO - Solução Atual

VII.1 Componentes da Solução

A tabela VII.2 fornece uma visão de alto nível relativamente às responsabilidades, características, processos e critérios dos componentes.

Componentes Internos	
Portal Comerciantes	Subcomponente do **SAFTPRO Portal**, que se encontra disponível para os comerciantes. Permite a entrega e consulta de ficheiros SAF-T, assim como a entrega e consulta de faturas.
Portal Consumidores	Subcomponente do **SAFTPRO Portal**, que se encontra disponível para os agentes económicos. Permite a entrega e consulta de faturas.
Portal *Backoffice*	Subcomponente do **SAFTPRO Portal**, também disponível na *internet* direcionado para as autoridades tributárias. Permite a consulta de ficheiros SAF-T e acompanhamento do *upload* de ficheiros em tempo real.
SAFTPRO *Webservice*	Disponibiliza uma API para o **SAFTPRO Portal**, que fornece funcionalidades como a entrega de ficheiros SAF-T.
Componentes Externos	
Sistema de Autenticação	Para autenticação e autorização de utilizadores nas aplicações.
File Store	Para armazenamento de ficheiros.
Base de Dados SAFTPRO	Armazena informação referente aos formatos dos ficheiros SAF-T, parâmetros de controlo (*bandwith, max. size* dos ficheiros, etc.) e informação das submissões.
Base de Dados SAFTPRO Portal	Armazena informação sobre o gestor de conteúdos do portal e os utilizadores registados no sistema.

Figura VII.2: SAFTPRO - Componentes

VII.2 Operações Suportadas

O produto disponibiliza diferentes funcionalidades consoante o tipo de utilizador (fig VII.3).

Comerciantes e Consumidores (Agentes Económicos)	• Envio de ficheiros XML por *upload*; • Consulta de informação derivada do envio de ficheiros XML por *webservice*; • Consulta de informação submetida por múltiplos critérios; • Informação sobre o processo de envio; • Cancelamento do processo de envio; • Visualização dos erros detetados em tempo real.
Autoridade Tributárias (em contexto de Operação de Sistemas)	• *Dashboard* para consulta de indicadores gerais e estatísticas; • Consola para monitorização em tempo real dos processos de envio; • Gestão dos recursos por processo de envio (largura de banda atribuída a um processo); • Possibilidade de cancelamento de processo de envio; • Definição da largura de banda atribuída ao sistema; • Configuração do limite máximo de erros por processo de envio.
Autoridade Tributárias (em contexto de Desenvolvimento)	• Configuração de validação sintática (*schema*); • Configuração de validação semântica; • Possibilidade de incremento de funcionalidades através de extensões OSGi; • Possibilidade de inclusão de mais campos ao formulário de upload do ficheiro; • Integração com outros sistemas (autenticação, autorização, cadastro, ...); • Integração com os portais de serviços das Autoridades Tributárias, adaptando-se às normas de identidade gráfica e interface definidas por cada uma.

Figura VII.3: SAFTPRO - Operações Suportadas

VII.3 Camadas do Sistema

A solução adota um modelo de camadas que fornece uma abstração a nível hierárquico, em que a camada "de baixo"fornece uma interface às camadas "de cima". Na figura VII.4 descreve essas camadas.

SAFTPRO Portais	
Camadas	• **Camada de apresentação**: camada superior da aplicação web responsável por processar os pedidos do utilizador e devolver a resposta adequada. Esta camada lida ainda com exceções lançadas pelas restantes camadas. Sendo o ponto de entrada da aplicação, é responsável também pela autenticação, impedindo a entrada de utilizadores não autorizados. • **Camada de serviços**: disponibiliza um conjunto de operações que mais não são do que invocações da **SAFTPRO REST API**, onde é implementada a lógica para a gestão de faturas e ficheiros SAF-T. • **Camada de persistência**: camada inferior da aplicação web responsável pela comunicação com a base de dados.
SAFTPRO *Webservices*: Serviço API	
Camadas	• **Camada de Apresentação:** esta camada fornece os *endpoints* da API REST que disponibiliza a maioria das funcionalidades oferecidas pela solução SAFTPRO. Nesta camada são realizados os mapeamento das respostas enviadas pelo **SAFTPRO Webservices**, incluindo também tratamento de exceções que possam surgir. Adicionalmente, disponibiliza um *endpoint* SOAP onde são recebidas e processadas faturas UBL 2.1 através do **módulo de *e-Invoicing***. • **Camada de Serviços:** esta camada implementa a lógica aplicacional que permite o envio, validação e acesso de documentos SAF-T por parte dos comerciantes, assim como o registo, consulta e edição de faturas presentes nos documentos SAF-T. Ao nível do **módulo de *e-Invoicing***, permite aos consumidores consultar as faturas que receberam da rede Peppol, fazer a sua entrega em *real-time* através de *webhooks*, alterar o estado das faturas e fazer o seu *download* num formato legível. Mais, permite aos comerciantes enviar faturas pela rede Peppol e obter a lista dos participantes servidos pelo *Access Point*. Para além disso, disponibiliza também operações para os utilizadores do *backoffice*, que permitem configurar o modo da submissão e validação dos ficheiro SAF-T, acompanhar a submissão de ficheiros em tempo real (SAF-T e UBL 2.1) e fornece estatísticas que permitem a construção de *dashboards* por parte do **SAFTPRO Portais**. • **Camada de persistência**: camada inferior da aplicação web responsável pela comunicação com a base de dados e file storage.
SAFTPRO *Webservices*: Serviço Batch	
Camadas	• **Camada de Serviços:** esta camada implementa um serviço batch e contém serviços automatizados (não necessitam de ser despoletados pelos utilizadores), que realizam operações periódicas que envolvem validações e persistência de informação na base de dados. • **Camada de persistência**: camada inferior da aplicação web responsável pela comunicação com a base de dados e file storage.
SAFTPRO *Webservices*: Serviço CLI	
Camadas	• **Camada de Apresentação:** disponibiliza o *endpoint* que permite o upload de ficheiros SAF-T pela linha de comandos. Esta camada é também responsável pelo mapeamento das repostas e do tratamento de eventuais exceções. • **Camada de Serviços:** Muito semelhante à camada de serviços da API, no entanto limita-se a oferecer funcionalidades de validação e submissão de ficheiros SAF-T. • **Camada de persistência**: camada inferior da aplicação web responsável pela comunicação com a base de dados e file storage.

Figura VII.4: SAFTPRO - Camadas do sistema

VII.4 Requisitos não funcionais (outros)

Além dos requisitos de desempenho, identificados no capítulo 3, existem outros igualmente importantes. Nomeadamente, os requisitos de Segurança (fig. VII.5, Documentação (fig. VII.6) e Manutenção (fig. VII.7).

ID Need	ID RQ	Requisito	Descrição
ND02 **ND03** **ND04**	**RS1**	Confidencialidade/Comunicação Segura	Os *webservices* e os canais disponibilizados para comunicação pelo **SAFTPRO** deverão ser cifrados através de protocolos seguros e que possibilitem a evolução do protocolo utilizado.
	RS2	Autenticação	Os utilizadores terão que se autenticar previamente para poderem aceder às funcionalidades disponíveis. A troca de informações com sistemas externos também prossupõe um processo de autenticação mútua, para que ambos os sistemas possam provar a sua legitimidade. Este processo de autenticação deverá ser realizado por meio de certificados digitais.
	RS3	Autorização	Os utilizadores poderão aceder apenas às funcionalidades para as quais estão autorizados. O **SAFTPRO** deverá ser capaz de gerir que informações e funcionalidades disponibiliza a uma certa entidade.
	RS4	Integridade	Os *payloads* trocados durante as comunicações deverão possuir mecanismos que garantam a integridade da informação transmitida, para evitar dados corrompidos e/ou alterados.
	RS5	Proteção contra ataques web	A solução deverá estar protegida contra ataques *SQL Injection*.
	RS6	Monitorização/Auditoria	Guardar o registo de atividade (*log*) de todas as operações para permitir auditorias sobre as operações efetuadas. Nomeadamente, deverão ser registadas de forma persistente todas as operações associadas ao utilizador que as efetuou assim como a respetiva data/hora.

Figura VII.5: SAFTPRO - Requisitos de Segurança

ID Need	ID RQ	Requisito	Descrição
ND01	**RD01**	Documentação das Configurações Suportadas	Deve ser produzida documentação simples que defina o processo de adaptação de uma fatura ao formato UBL 2.1, assim como as validações suportadas pelo sistema nos ficheiros de configuração.
ND03	**RD01**	Documentação das APIs	Deve ser produzida documentação sobre as operações implementadas na API, para que esta possa ser tornada pública e disponibilizada aos sistemas externos que irão utilizar a API.

Figura VII.6: SAFTPRO - Requisitos de Documentação

ID Need	ID RQ	Requisito	Descrição
ND01	**RM1**	Extensibilidade	A solução deverá ser flexível para permitir adotar novas faturas, sem a necessidade de reconstruir o produto.
ND01	**RM2**	Modularidade	Deverá ser possível reutilizar os componentes desenvolvidos para novas funcionalidades, com pouco esforço.
ND02			
ND03			
ND04			

Figura VII.7: SAFTPRO - Requisitos de Manutenção

VII.5 Tecnologias

A solução é suportada pelas ferramentas listadas na tabela da figura VII.8.

Linguagens de Programação	Java, JavaScript, SQL, XML, XSD HTML.
Bases de Dados	MySQL.
***Frameworks* de Desenvolvimento**	Spring, Apache Maven, Apache Kafka, Hibernate, React, NodeJS, Webpack.
Protocolos de Comunicação	HTTPS, WSS (WebSocket).
Middleware	Apache HTTP Server, Liferay.

Figura VII.8: SAFTPRO - Tecnologias utilizadas

VIII

Templates Xtend e Classes geradas

```
1  class ElementModelExtractorTemplate {
2      ...
3      def generateElementModelExtractorClass(ElementMapping mappingElement) {
4          ...
5          '''
6          package «package»;
7
8          import ...
9
10         @Component
11         public class «className» {
12             «IF isRoot»
13             public «countryModel» extract«countryModel»(Node root){
14                 Node current = this.xPathExtractor.getNode("/*[local-name()='«countryElement»']", root);
15                 ...
16
17                 «countryModel» model = new «countryModel»();
18
19                 //Basic fields
20                 «FOR basicField: mappingElement.basicFieldMappingsList»
21                 «var basicFieldTag = '"'+basicField.tag+'"'»
22                 «switch basicField.fieldType {
23                 case 'String': "model.set"+basicField.countryField+
24                         '(this.xPathExtractor.getString('+basicFieldTag+', current));'
25                 case 'Float': "model.set"+basicField.countryField+
26                         '(this.xPathExtractor.getFloat('+basicFieldTag+', current));'
27                 ...
28                 default: '' }»
29                 «ENDFOR»
30                 ...
31
32                 return model;
33             }
34             «ELSE»
35              ...
36             «ENDIF»
37         }
38         '''
39     }
40     ...
41 }
```

Listagem 48: ElementModelExtractorTemplate.xtend

O *ElementModelExtractorTemplate* (listagem 48) ilustra algumas potencialidades dos templates Xtend. Por exemplo, é possível definir *loops* (linha 20 a 29) que iteram sobre

os campos básicos declarados num elemento e com base nessa informação invocar o método apropriado do `XPathExctrator` para obter o valor. Em adição, também é possível aceder às relações dos elementos e extrair recursivamente a sua informação até chegar à representação final da fatura.

```
1  package pt.opensoft.phase4core.xmlparser.handler;
2
3  import ...
4
5  @Component
6  public class InvoiceModelExtractor {
7
8      private XPathExtractor xPathExtractor;
9      private AdditionalSupportingDocumentModelExtractor additionalSupportingDocumentModelExtractor;
10     private AllowanceChargeModelExtractor allowanceChargeModelExtractor;
11     ...
12     public InvoiceModel extractInvoiceModel(Node root) {
13         Node current = this.xPathExtractor.getNode("/*[local-name()='Invoice']", root);
14         ...
15
16         InvoiceModel model = new InvoiceModel();
17
18         //Basic fields
19         model.setBuyerAccountingReference(this.xPathExtractor.getString("cbc:AccountingCost", current));
20         model.setBuyerReference(this.xPathExtractor.getString("cbc:BuyerReference", current));
21         model.setContractReference(this.xPathExtractor.getString("cac:ContractDocumentReference/cbc:ID", current));
22         ...
23
24         //Child complex fields
25         model.setAdditionalSupportingDocuments(this.additionalSupportingDocumentModelExtractor
26                     .extractAdditionalSupportingDocuments(current, "cac:AdditionalDocumentReference"));
27         model.setAllowanceCharges(this.allowanceChargeModelExtractor
28                     .extractAllowanceCharges(current, "cac:AllowanceCharge"));
29         ...
30
31         return model;
32     }
33 }
```

Listagem 49: InvoiceModelExtractor.java

No `EntityClassTemplate` (listagem 50) o foco é definir corretamente as *entities* para os elementos criados na DSL, sendo que estes devem respeitar as tabelas da base de dados. Por omissão, as entidades têm um identificador que é incrementado automaticamente (linhas 16 a 19) e o script DDL já considera esse aspeto, pelo que os elementos devem omitir a definição desse campo.

À semelhança do `ElementModelExtractorTemplate`, os campos básicos são iterados e assinalados com a anotação `@Column` do JPA (linha 22 a 33). A definição das relações é um pouco mais complexa, uma vez que no JPA navega-se através de referências e na base de dados a navegação acontece com recurso a chaves estrangeiras. Não obstante, o JPA disponibiliza as anotações que permitem identificar o tipo da relação, como por exemplo `@OneToMany`, ou `@ManyToOne`, entre outras.

```
class EntityClassTemplate {
    ...
    def generateEntityClass(ElementMapping mappingElement) {
        var standardElement = mappingElement.standardElement;
        ...

        '''
        package «package»;

        import ...

        @Entity
        @Table(name = "«StringUtils.camelCaseToSnakeLowerCase(standardElement)»")
        public class «standardEntity» implements Serializable {

            @Id
            @GeneratedValue(strategy = GenerationType.IDENTITY)
            @Column(name = "ID")
            private Long id;

            ...
            «FOR basicField : mappingElement.basicFieldMappingsList»
            «var basicFieldUncap = StringUtils.uncapitalize(basicField.standardField)»
            «IF basicField.fieldType.equalsIgnoreCase("byte []")»
            @Lob
            «ENDIF»
            «IF basicField.mandatory»
            @Column(name = "«StringUtils.camelCaseToSnakeUpperCase(basicFieldUncap)»", nullable = false)
            «ELSE»
            @Column(name = "«StringUtils.camelCaseToSnakeUpperCase(basicFieldUncap)»")
            «ENDIF»
            private «basicField.fieldType» «basicFieldUncap»;
            «ENDFOR»
            ...
            «FOR fatherLink : fatherLinks»
            «var linkUncap = StringUtils.uncapitalize(fatherLink.referedElement.standardElement)»
            «IF fatherLink.cardinality == RelationsCardinality.MANY_TO_ONE»
            @«fatherLink.cardinality»(fetch = FetchType.LAZY)
            «IF fatherLink.mandatory»
            @JoinColumn(name = "«StringUtils.camelCaseToSnakeUpperCase(fatherLink.standardField)»_ID",
                        referencedColumnName = "ID", nullable = false)
            «ELSE»
            @JoinColumn(name = "«StringUtils.camelCaseToSnakeUpperCase(fatherLink.standardField)»_ID",
                        referencedColumnName = "ID")
            «ENDIF»
            private «fatherLink.standardField»Entity «linkUncap»;
            «ELSE»
            ...
            «ENDIF»
            «ENDFOR»
            ...
         }
        '''
    }
    ...
}
```

Listagem 50: EntityClassTemplate.xtend

```java
package pt.opensoft.phase4core.persistence.entity.en16931;

import ...

@Entity
@Table(name = "invoice")
public class InvoiceEntity implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "ID")
    private Long id;

    @Column(name = "BUYER_ACCOUNTING_REFERENCE")
    private String buyerAccountingReference;

    @Column(name = "BUYER_REFERENCE")
    private String buyerReference;
    ...
    @OneToMany(mappedBy = "invoice", fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    private List<AllowanceChargeEntity> allowanceCharges;

    @ManyToOne(fetch = FetchType.LAZY, cascade = CascadeType.ALL)
    @JoinColumn(name = "BUYER_ID", referencedColumnName = "ID", nullable = false)
    private BuyerEntity buyer;
    ...
    public InvoiceEntity() {
        this.allowanceCharges = new LinkedList<>();
        ...
    }
    // Getter & Setters
}
```

Listagem 51: InvoiceEntity.java

```
class UBLNamespaceContextConfig {
    ...
    def generateClass() {
        var rootPackage = path.replace("/", ".")
        '''
        package «rootPackage»;

        import ...

        @Configuration
        public class UBLNamespaceContextConfig {

            @Bean
            public SimpleNamespaceContext ublNamespaceContext() {
                SimpleNamespaceContext namespaceContext = new SimpleNamespaceContext();
                namespaceContext.bindDefaultNamespaceUri("urn:oasis:names:specification:ubl:schema:xsd:Invoice-2");
                namespaceContext.bindNamespaceUri("cbc",
                "urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2");
                namespaceContext.bindNamespaceUri("cac",
                "urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2");

                return namespaceContext;
            }
        }
        '''
    }
    ...
}
```

Listagem 52: UBLNamespaceContextConfig.xtend

```java
package pt.opensoft.phase4core.xmlparser;

import ...

@Configuration
public class UBLNamespaceContextConfig {

    @Bean
    public SimpleNamespaceContext ublNamespaceContext() {
        SimpleNamespaceContext namespaceContext = new SimpleNamespaceContext();
        namespaceContext.bindDefaultNamespaceUri("urn:oasis:names:specification:ubl:schema:xsd:Invoice-2");
        namespaceContext.bindNamespaceUri("cbc",
                "urn:oasis:names:specification:ubl:schema:xsd:CommonBasicComponents-2");
        namespaceContext.bindNamespaceUri("cac",
                "urn:oasis:names:specification:ubl:schema:xsd:CommonAggregateComponents-2");

        return namespaceContext;
    }

}
```

Listagem 53: UBLNamespaceContextConfig.java

IX

REST API

A documentação é extremamente importante na disponibilização de APIs. Para o módulo de *e-Invoicing* optou-se por utilizar a ferramenta SpringDoc[1], que através da especificação OpenAPI 3.0[2] permite gerar informação acerca das operações implementadas nos *Controllers*. Como é hábito, o Spring configura automaticamente a ferramenta e portanto basta incluir a sua dependência no ficheiro POM do projeto, mas existe sempre possibilidade de refinar os *defaults* da configuração.

IX.1 *Sender* API

API disponibilizada para enviar faturas através do protocolo AS4.

POST: /rest/bearer/e-invoicing/sbd

Endpoint disponível para enviar um documento para o *business partner* na rede Peppol. De acordo com os *Transport Infrastructure Agreements* do Peppol e os valores *default* da biblioteca phase4, as garantias para quem invoca a operação são:

- ***Connection Request Timeout* (ms):** 30000;
- ***Connection Timeout* (ms):** 5000;
- ***Socket Timeout* (ms):** 30000;

[1] https://springdoc.org/
[2] https://swagger.io/specification/

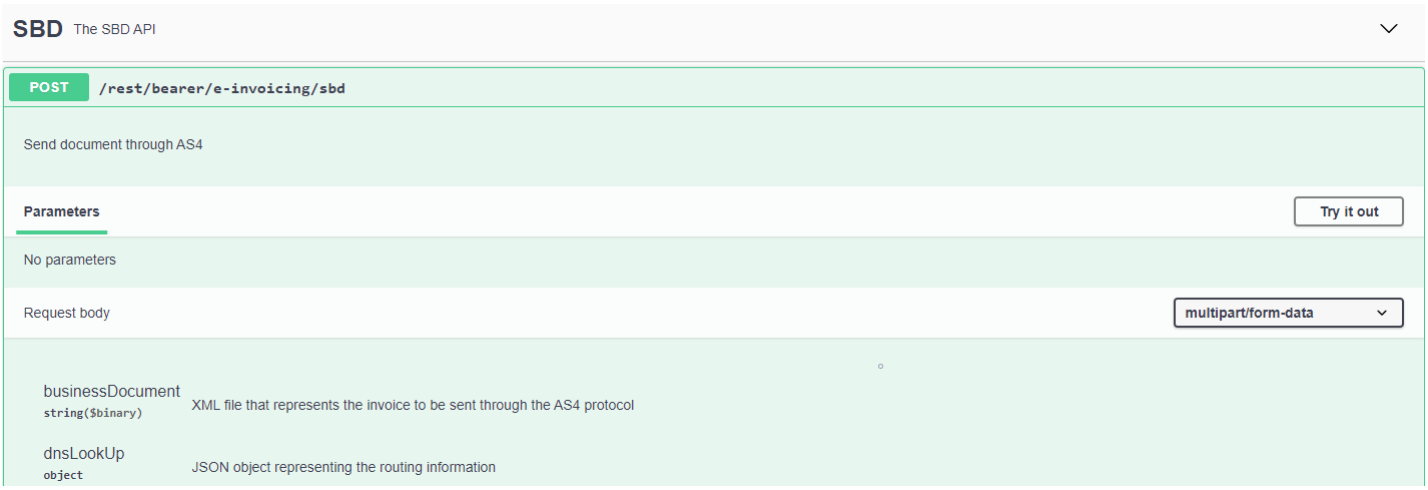


Figura IX.1: Descrição da operação POST

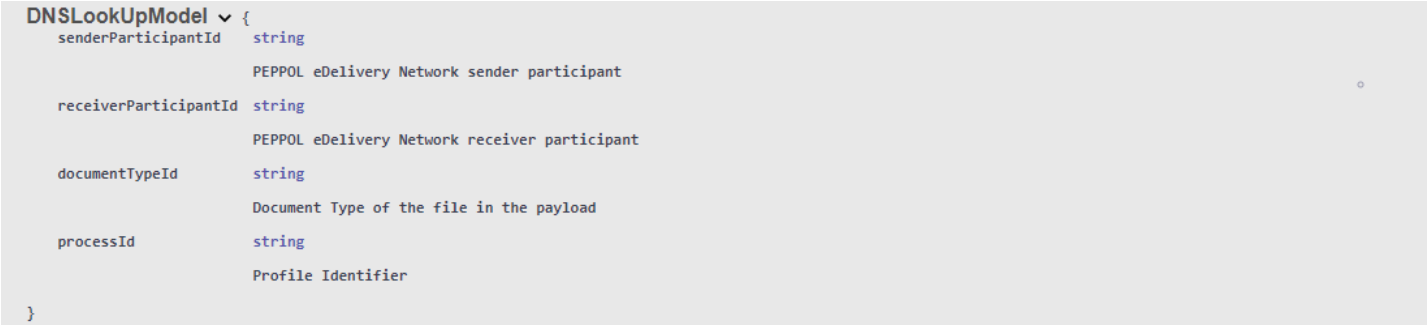


Figura IX.2: Descrição do modelo que representa a informação de *routing*

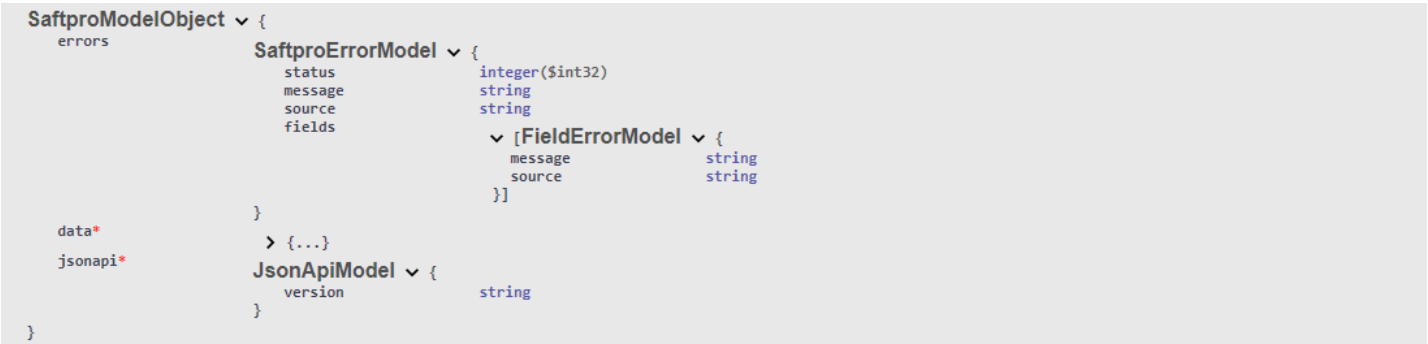


Figura IX.3: Descrição do modelo resposta (Content-Type "application/json")

Status Code	Descrição
200	O documento foi validado de acordo com as regras do Peppol, a informação de *routing* está correta e a transmissão do documento para o *Receiver* AP foi concluída com sucesso.
400	O documento XML ou a informação de *routing* é inválido(a).
401	Quem está a tentar efetuar o pedido não possui credenciais válidas e portanto não poderá utilizar o serviço.
403	Ocorre quando o utilizador que efetua um pedido com um *token* inválido ou quando este pertence a um Peppol *Participant* com um identificador diferente do que consta na informação de *routing* (senderParticipantId).
500	Ocorreu um erro inesperado no *Sender* AP ou o *Receiver* AP recusou-se a entregar o documento ao destinatário.

Figura IX.4: Descrição dos *Status Codes*

GET: /rest/bearer/e-invoicing/directory

Endpoint disponível para obter os participantes do Peppol registados no SMP.

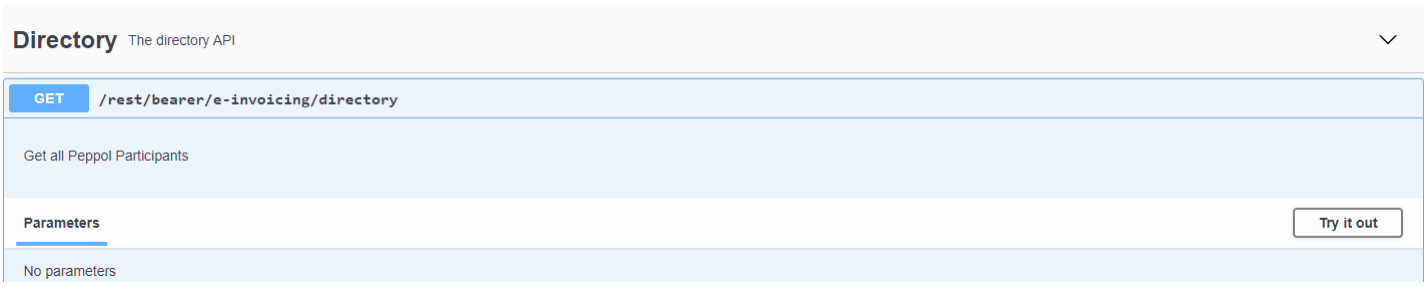


Figura IX.5: Descrição da operação GET

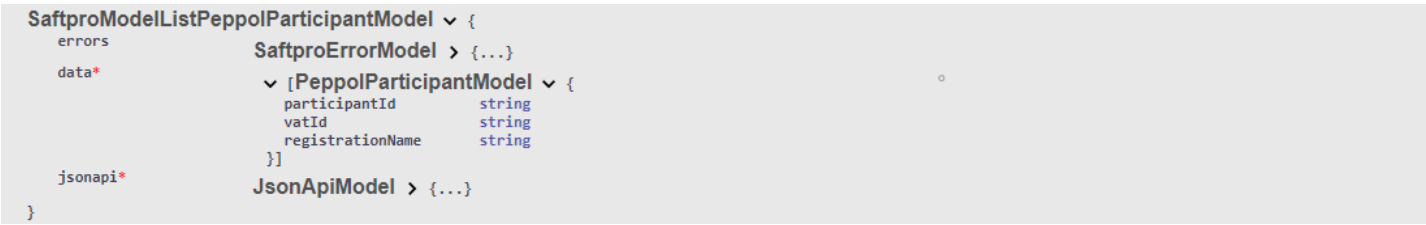


Figura IX.6: Descrição do modelo resposta (`Content-Type` “application/json”)

Status Code	Descrição
200	Os participantes foram retornados com sucesso (a lista pode estar vazia).
401	Quem está a tentar efetuar o pedido não possui credenciais válidas e portanto não poderá utilizar o serviço.
403	Ocorre quando o utilizador que efetua um pedido com um token inválido ou quando este não tem permissão para aceder ao recurso.
500	Ocorreu um erro inesperado no *Sender* AP, que não o permitiu retornar os participantes.

Figura IX.7: Descrição dos *Status Codes*

IX.2 *Receiver* API

API disponibilizada para consultar/alterar o estado das faturas.

GET: /rest/bearer/e-invoicing/invoices/{invoiceNumber}/status

Endpoint disponível para obter o estado de uma fatura.

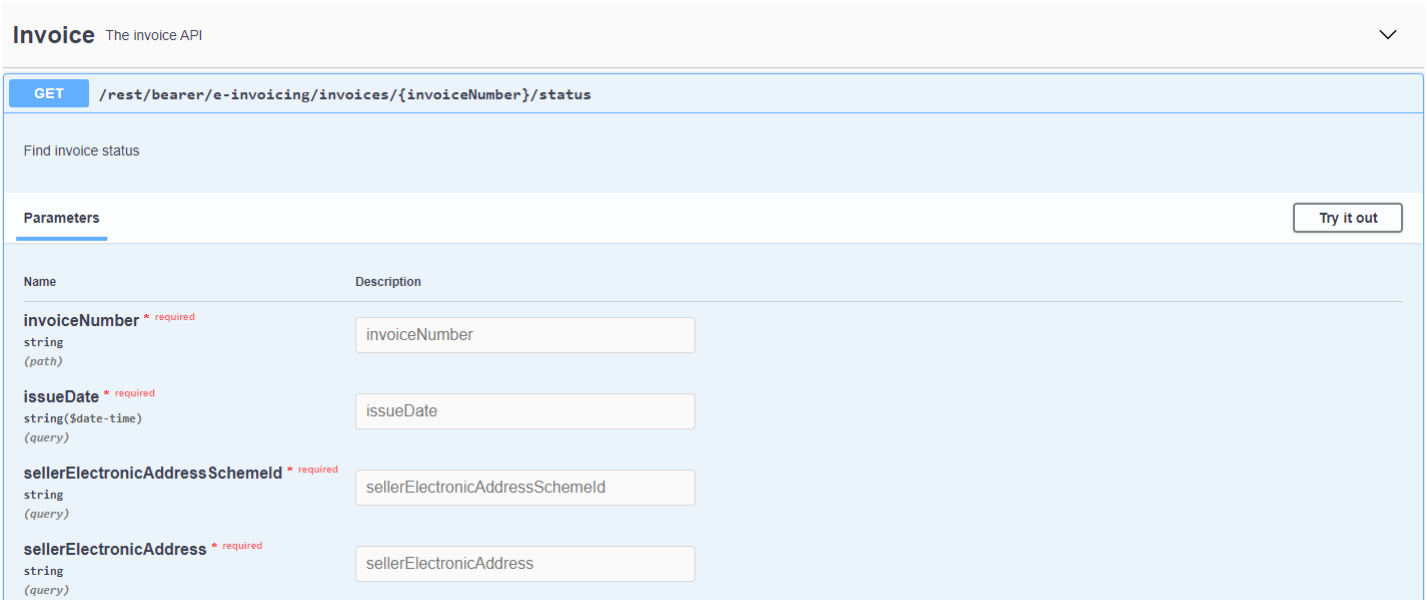


Figura IX.8: Descrição da operação GET

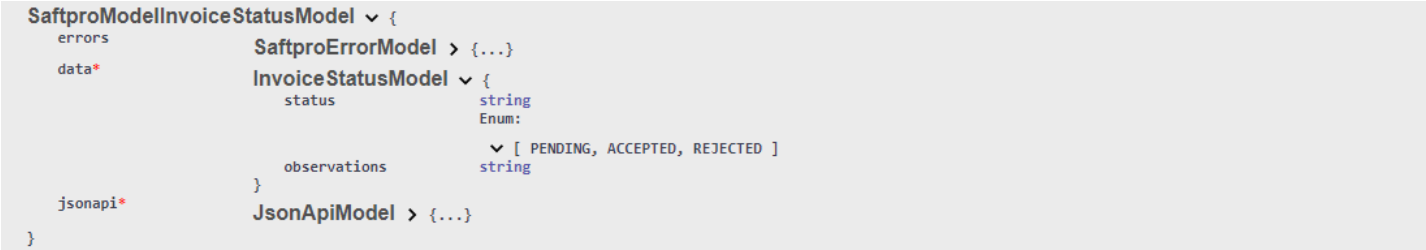


Figura IX.9: Descrição do modelo resposta (`Content-Type` "application/json")

Status Code	Descrição
200	A fatura existe e foi retornado o seu estado mais recente.
400	Faltam parâmetros que permitem identificar a fatura.
401	Quem está a tentar efetuar o pedido não possui credenciais válidas e portanto não poderá utilizar o serviço.
403	O utilizador que efetuou o pedido não tem permissões para o fazer.
404	A fatura não existe.
500	Ocorreu um erro inesperado.

Figura IX.10: Descrição dos *Status Codes*

PUT: /rest/bearer/e-invoicing/invoices/{invoiceNumber}/status

Endpoint disponível para alterar o estado de uma fatura.

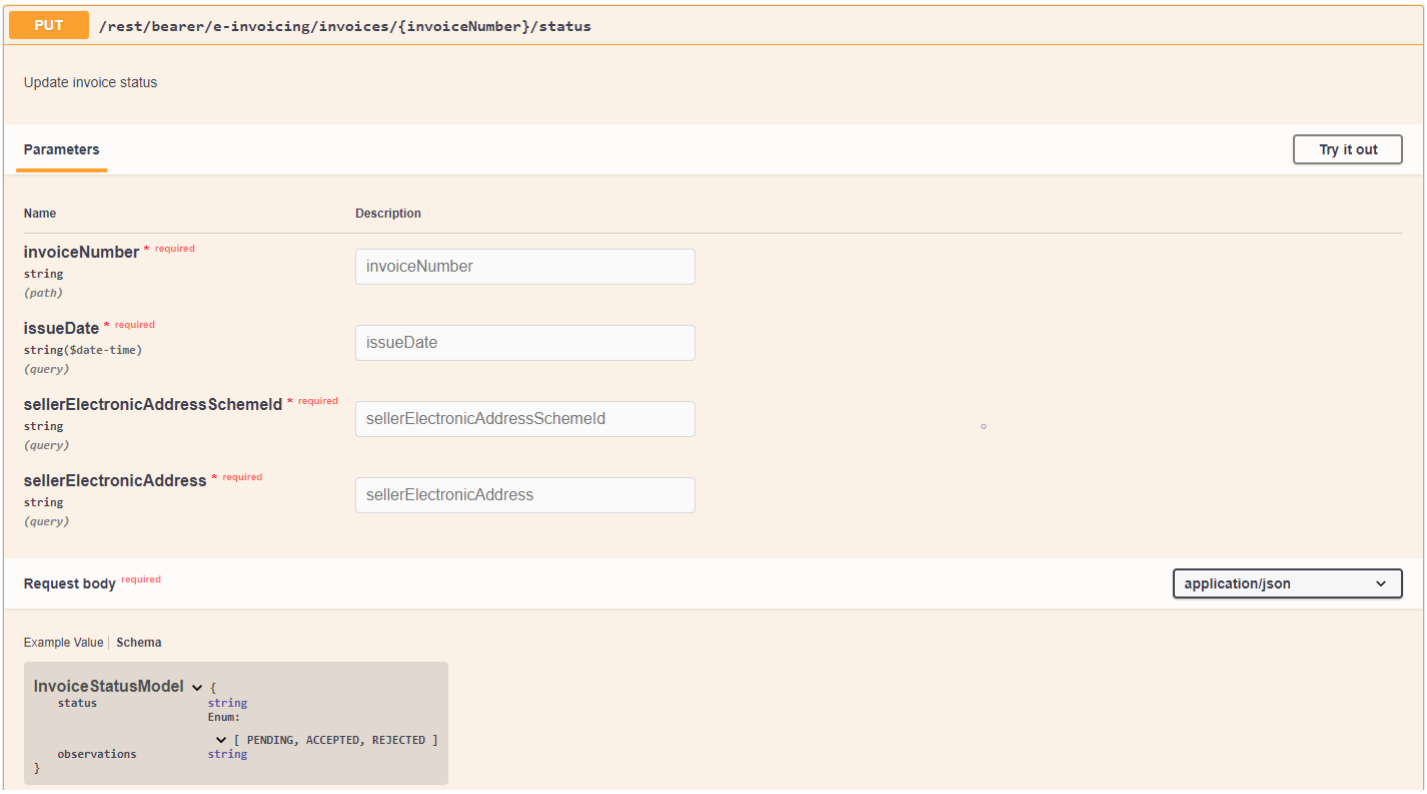


Figura IX.11: Descrição da operação PUT

A resposta é semelhante à da operação GET e corresponde ao estado da fatura atualizado.

Status Code	Descrição
200	O estado da fatura foi atualizado com sucesso e foi retornado no corpo da resposta o seu estado mais recente.
400	Faltam parâmetros que permitem identificar a fatura, ou o body têm campos que não são válidos.
401	Quem está a tentar efetuar o pedido não possui credenciais válidas e portanto não poderá utilizar o serviço.
403	O utilizador que efetuou o pedido não tem permissões para o fazer.
404	A fatura não existe.
500	Ocorreu um erro inesperado.

Figura IX.12: Descrição dos *Status Codes*

GET: /rest/bearer/e-invoicing/invoices/{invoiceNumber}/pdf

Endpoint que permite transformar a fatura UBL 2.1 em PDF, através de *stylesheets* XSLT[3].

[3] https://github.com/OpenPEPPOL/peppol-bis-invoice-3/tree/master/stylesheet

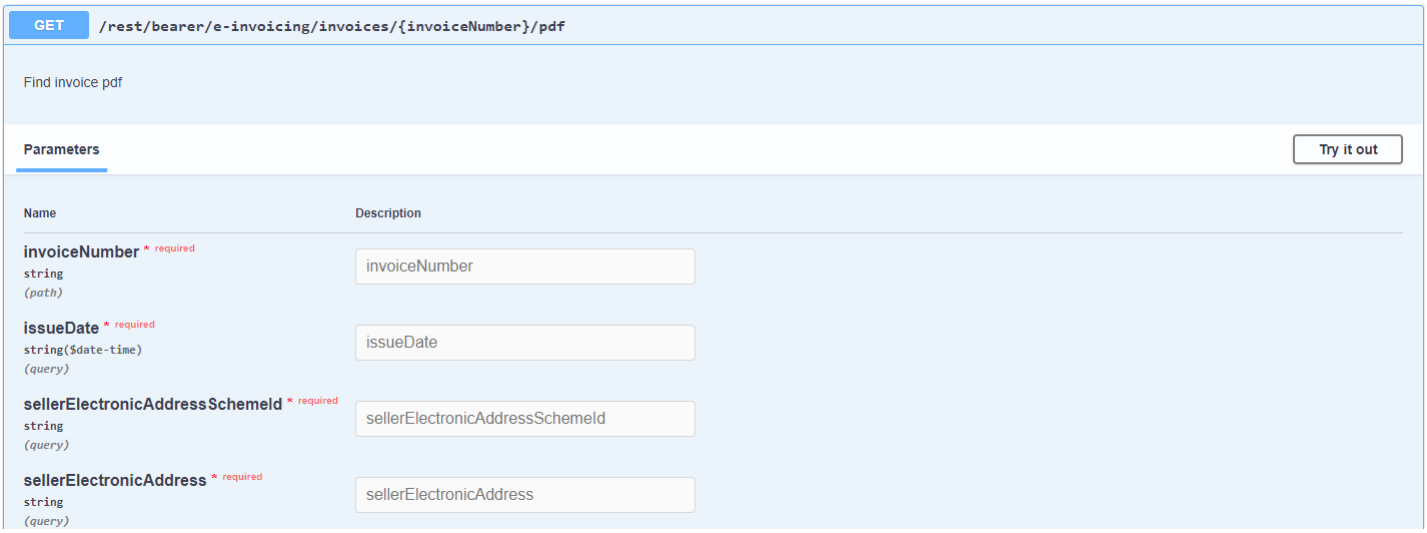


Figura IX.13: Descrição da operação GET

A resposta tem `Content-Type` "application/pdf". O nome do ficheiro encontra-se nos parâmetros do *header* `Content-Disposition` e o *body* corresponde aos detalhes da fatura em PDF.

Status Code	Descrição
200	A fatura existe e foi possível converter o seu conteúdo em PDF.
400	Faltam parâmetros que permitem identificar a fatura.
401	Quem está a tentar efetuar o pedido não possui credenciais válidas e portanto não poderá utilizar o serviço.
403	O utilizador que efetuou o pedido não tem permissões para o fazer.
404	A fatura não existe.
500	Ocorreu um erro inesperado.

Figura IX.14: Descrição dos *Status Codes*

GET: /rest/bearer/e-invoicing/invoices/summary

Endpoint que suporta a aplicação *desktop* desenvolvida em Flutter, uma vez que a linguagem Dart não possui um processamento de XML tão elaborado como o Java, dificultando a extração de informação proveniente das faturas. Para além disso, resolve também questões de *performance*, quando existe um número elevado de faturas registadas, porque o modelo devolvido é muito mais simples que a representação completa da fatura.

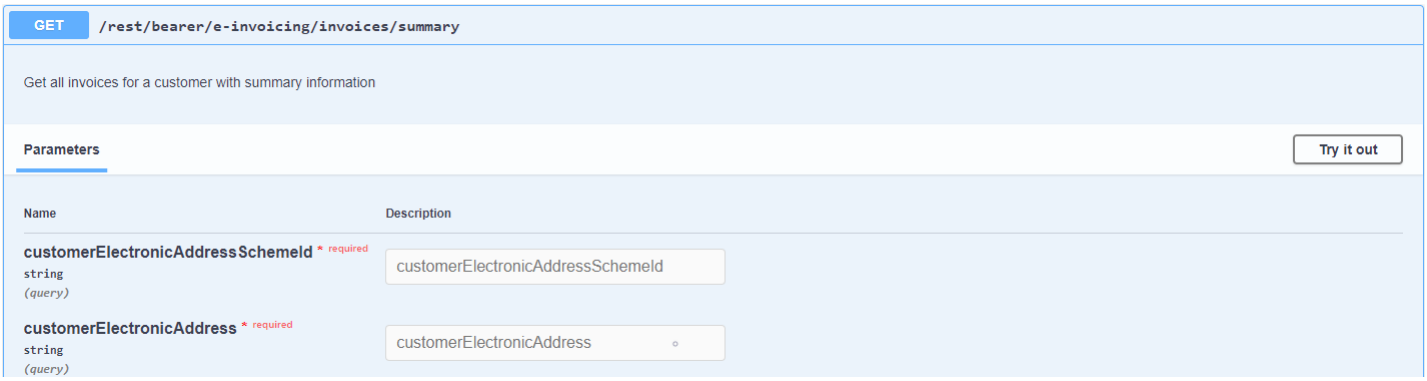


Figura IX.15: Descrição da operação GET

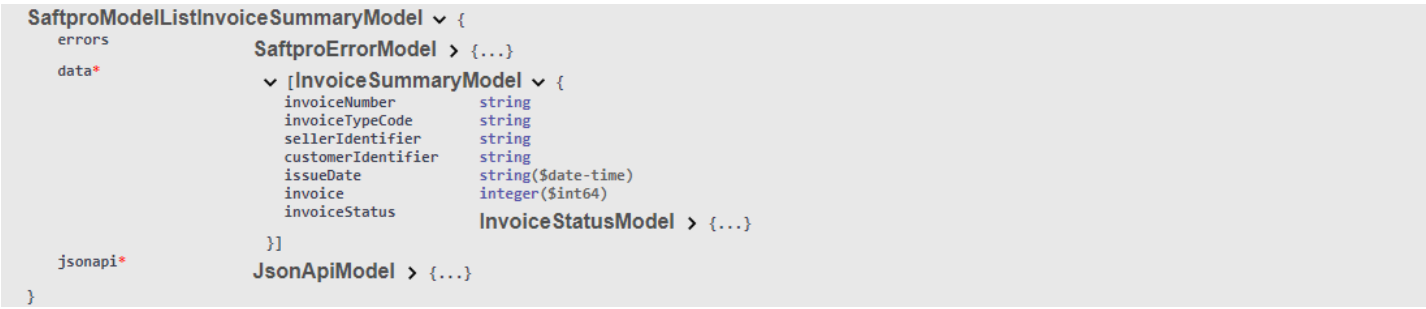


Figura IX.16: Descrição do modelo resposta (`Content-Type` "application/json")

Status Code	Descrição
200	O utilizador existe e foi retornado um resumo de todas as faturas em que este consta como consumidor.
400	Faltam parâmetros que permitem identificar a fatura.
401	Quem está a tentar efetuar o pedido não possui credenciais válidas e portanto não poderá utilizar o serviço.
403	O utilizador que efetuou o pedido não tem permissões para o fazer.
404	O consumidor não existe.
500	Ocorreu um erro inesperado.

Figura IX.17: Descrição dos *Status Codes*

IX.3 *Backoffice* API

API que torna possível a integração das funcionalidades do módulo de *e-Invoicing* com o SAFTPRO Portais. Na realidade, esta API é apenas uma reiteração de alguns *endpoints* já descritos, mas que estão protegidos com outro mecanismo de autenticação.

GET: /rest/e-invoicing/invoices/summary

Endpoint que permite obter informação sucinta sobre um conjunto de faturas. Para o *backoffice*, os critérios de pesquisa são mais minuciosos quando comparados com os disponibilizados para o Consumidor.

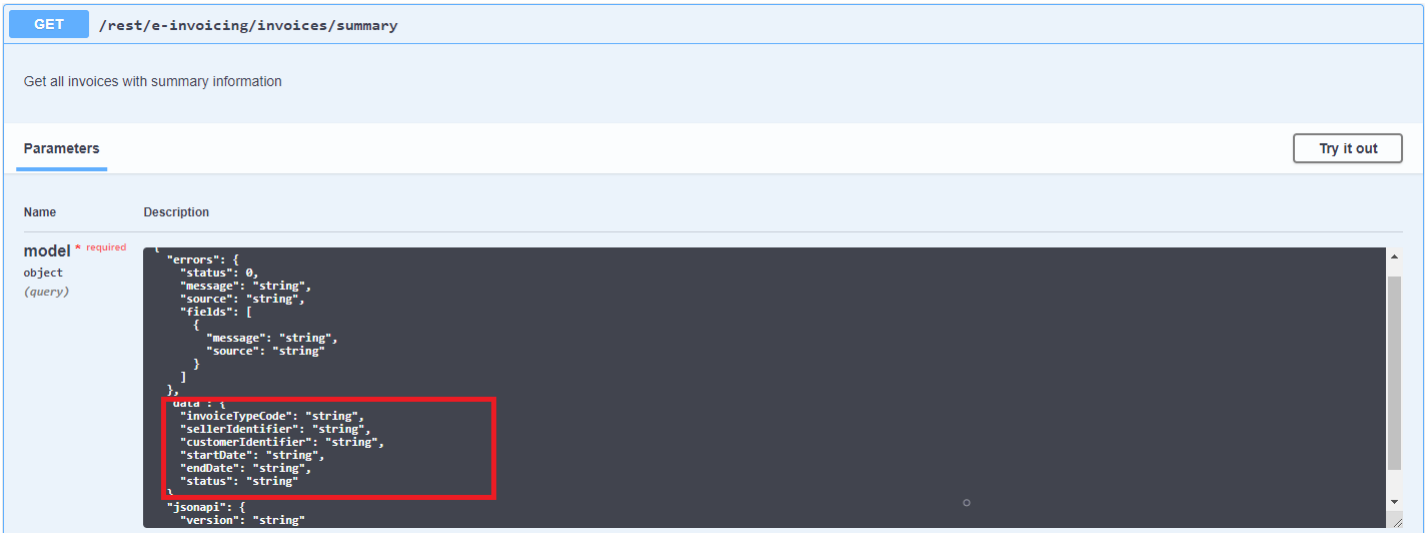


Figura IX.18: Descrição da operação GET

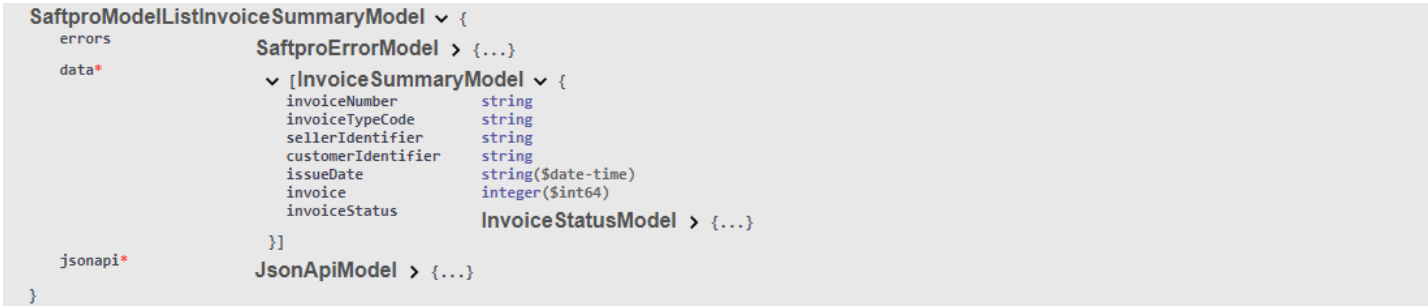


Figura IX.19: Descrição do modelo resposta (`Content-Type` "application/json")

Status Code	Descrição
200	Os critérios de pesquisa são válidos e foi retornado um conjunto resumido de faturas.
400	Faltam parâmetros que permitem filtrar as faturas.
401	Quem está a tentar efetuar o pedido não possui credenciais válidas e portanto não poderá utilizar o serviço.
403	O utilizador que efetuou o pedido não tem permissões para o fazer.
500	Ocorreu um erro inesperado.

Figura IX.20: Descrição dos *Status Codes*

GET: /rest/e-invoicing/invoices/{id}

Endpoint que permite aceder aos detalhes das faturas recebidas pela rede Peppol e destinadas a Consumidores registados no sistema.

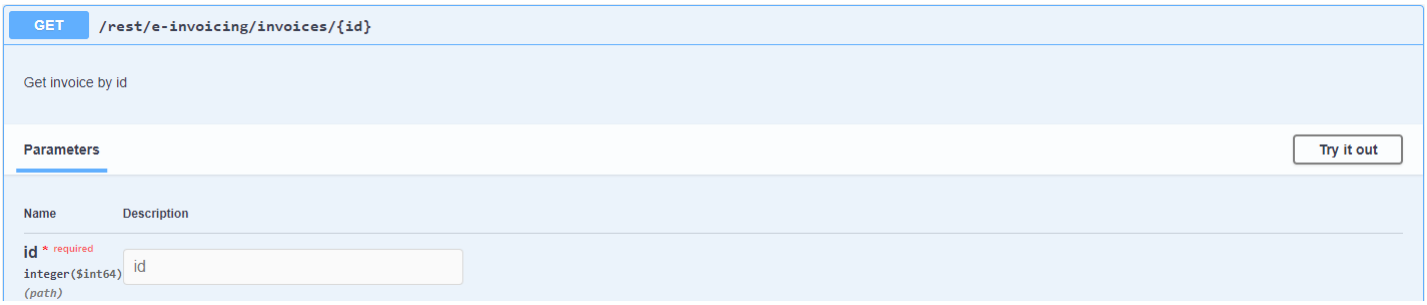


Figura IX.21: Descrição da operação GET

Figura IX.22: Descrição do modelo resposta (`Content-Type` “application/json”)

Status Code	Descrição
200	A fatura existe e foi retornada.
401	Quem está a tentar efetuar o pedido não possui credenciais válidas e portanto não poderá utilizar o serviço.
403	O utilizador que efetuou o pedido não tem permissões para o fazer.
404	A fatura não existe.
500	Ocorreu um erro inesperado.

Figura IX.23: Descrição dos *Status Codes*

X

Token-Based Authentication e Políticas de Segurança

X.1 Concretização da *Token-Based Authentication*

Para uma aplicação *Web* usufruir das funcionalidades presentes no Spring *Security*, basta incluir a sua dependência Maven no projeto. No entanto, é recomendado uma configuração mais fina posteriormente, para redefinir algumas propriedades básicas de acesso.

```
1  package pt.opensoft.saftpro.configuration.einvoicing.security;
2
3  import ...
4
5  @Order(1)
6  @EnableWebSecurity
7  public class SecurityConfig extends WebSecurityConfigurerAdapter {
8
9      @Value("${phase4.url-mapping}")
10     private String phase4UrlMapping;
11     private JWTService jwtService;
12     private PasswordEncoder passwordEncoder;
13     private EInvoicingUserDetailsService userDetailsService;
14     ...
15     @Override
16     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
17         auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
18     }
19
20     @Override
21     protected void configure(HttpSecurity http) throws Exception {
22         http.csrf().disable().authorizeRequests()
23                 .antMatchers(HttpMethod.POST, "/e-invoicing/login", phase4UrlMapping).permitAll()
24                 .antMatchers("/rest/bearer/e-invoicing/**").authenticated().and()
25                 .httpBasic().authenticationEntryPoint((request, response, exception) -> {...}).and()
26                 .addFilterBefore(new UserPasswordAuthenticationFilterToJWT("/e-invoicing/login",
27                         super.authenticationManager(), jwtService), BasicAuthenticationFilter.class)
28                 .addFilterBefore(new JWTAuthenticationFilter(jwtService), BasicAuthenticationFilter.class);
29     }
30 }
```

Listagem 54: SAFTPRO *Webservices* - SecurityConfig.java

No módulo de comunicação, o Spring *Security* foi configurado através da extensão da classe base `WebSecurityConfigurerAdapter` (listagem 54). Do código acima, destacam-se as seguintes linhas:

- **Linha 5:** a anotação `@Order` define a precedência de configurações, quando existe mais do que uma com o mesmo propósito. Quanto menor o valor da anotação, maior a sua prioridade. Antes do módulo de *e-Invoicing* já existia uma configuração deste tipo, tornando necessário recorrer ao `@Order` para que ambas pudessem coexistir;
- **Linha 6:** a anotação `@EnableWebSecurity` indica ao Spring que as configurações de segurança declaradas na classe devem ser aplicadas;
- **Linha 17:** o `EInvoicingUserDetailsService` implementa a interface `UserDetailsService` e é utilizado para obter os dados de um utilizador. A interface possui o método `loadUserByUsername()`, que é personalizado para utilizar a base de dados com o intuito de encontrar os participantes da rede Peppol registados no sistema. O `passwordEncoder` é utilizado para codificar a *password* dos utilizadores na fase de registo e autenticação;
- **Linha 23:** os métodos `antMatchers()` e `permitAll()` especificam os URLs que podem ser acedidos sem autenticação. Neste caso, o URL "/e-invoicing/login" corresponde ao *endpoint* para obter o *token* JWT e a variável `phase4UrlMapping` contém o valor do *endpoint* SOAP utilizado para receber documentos do protocolo AS4. Este último não é acessível a todos, no entanto as políticas de segurança são implementadas pela biblioteca phase4, pelo que não existe necessidade de aplicar-lhe os filtros;
- **Linhas 26 a 28:** a integração dos *tokens* JWT é feita através da adição de dois filtros na cadeia de filtros de segurança, que intercetam os pedidos e as tentativas de autenticação. O filtro `UserPasswordAuthenticationFilterToJWT` (listagem 55) processa o URL "/e-invoicing/login" e autentica o utilizador através do `UserDetailsService` configurado previamente. Caso a operação tenha sucesso, o *token* JWT é adicionado ao *header Authorization* da resposta e deverá ser utilizado nos pedidos seguintes, sujeito à validação do filtro `JWTAuthenticationFilter` (listagem 56).

```java
package pt.opensoft.saftpro.security.einvoicing.filter;

import ...

@Slf4j
public class UserPasswordAuthenticationFilterToJWT extends AbstractAuthenticationProcessingFilter {

    private JWTService jwtService;
    ...
    @Override
    public Authentication attemptAuthentication(HttpServletRequest httpServletRequest,
                                                HttpServletResponse httpServletResponse)
                                                throws AuthenticationException, IOException {
        LoginModel loginModel = new ObjectMapper().readValue(httpServletRequest.getInputStream(), LoginModel.class);
        Authentication authentication = getAuthenticationManager()
            .authenticate(new UsernamePasswordAuthenticationToken(loginModel.getUsername(), loginModel.getPassword()));

        if (authentication.isAuthenticated()) {
            SecurityContextHolder.getContext().setAuthentication(authentication);
            return authentication;
        } else {
            log.error("Could not authenticate the login request for the username " + loginModel.getUsername());
            return null;
        }
    }

    @Override
    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
                                            FilterChain chain, Authentication authentication) {
        jwtService.addResponseToken(response, authentication);
    }
}
```

Listagem 55: SAFTPRO *Webservices* - UserPasswordAuthenticationFilterToJWT.java

```java
package pt.opensoft.saftpro.security.einvoicing.filter;

import ...

@Slf4j
public class JWTAuthenticationFilter extends GenericFilterBean {

    private JWTService jwtService;
    ...
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)
                         throws IOException, ServletException {
        String authorizationHeader = ((HttpServletRequest) servletRequest).getHeader(HttpHeaders.AUTHORIZATION);

        if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {
            try {
                String token = authorizationHeader.substring(7); // Skip 7 characters for "Bearer "
                Claims claims = jwtService.parseClaims(token);
                String username = claims.get("username", String.class);
                Authentication authentication = new UserAuthenticationToken(username);

                SecurityContextHolder.getContext().setAuthentication(authentication);

                // Renew token with extended time here. (before doFilter)
                jwtService.addResponseToken((HttpServletResponse) servletResponse, authentication);
            } catch (JwtException exception) { ...}
        }
        filterChain.doFilter(servletRequest, servletResponse);
    }
}
```

Listagem 56: SAFTPRO *Webservices* - JWTAuthenticationFilter.java

X.2 Concretização das políticas de segurança

O Spring Security oferece semânticas de autorização ao nível dos métodos através de expressões SpEL[1], o que permite construir um mecanismo de *Expression-based Access Control*[2], que possibilita a definição de lógica complexa numa única expressão para posterior avaliação, a partir de um *Root Object* dentro de um *Evaluation Context*.

O mecanismo de *Expression-based Access Control* é mais intrincado do que permitir ou negar o acesso a uma operação e existem duas anotações que são utilizadas juntamente com as expressões SpEL, para determinar a sua validade. O `@PreAuthorize` é a anotação utilizada com mais frequência, porque valida uma determinada expressão antes de invocar o método protegido. Por oposição, o *@PostAuthorize* verifica a expressão após a invocação do método e pode ter efeitos colaterais sobre o resultado de uma operação.

Durante a avaliação do conteúdo das anotações, o *Root Object* permite aceder a alguns valores tais como os dados do utilizador autenticado. Mas, também é possível utilizar argumentos do método anotado, o que aumenta consideravelmente a expressividade na definição de novas políticas de acesso.

Frequentemente, existem situações onde métodos diferentes são protegidos pelas mesmas políticas de segurança. Para evitar a duplicação de código, sujeita a problemas de manutenção no futuro, o Spring resolve este problema através de *Meta Annotations*, que exprimem as regras uma única vez. Deste modo, o código pode ser reutilizado para diferentes métodos e promove o desacoplamento entre a lógica de negócio e os mecanismos de segurança.

Juntando tudo, o primeiro passo é recorrer a uma classe de configuração (listagem 57), para que seja possível utilizar as anotações *pre*/*post* no Spring.

```
1  package pt.opensoft.saftpro.configuration;
2
3  import ...
4
5  @EnableGlobalMethodSecurity(jsr250Enabled = true, prePostEnabled = true)
6  public class ControllerMethodAccessSecurityConfig extends GlobalMethodSecurityConfiguration {
7
8  }
```

Listagem 57: SAFTPRO *Webservices* - ControllerMethodAccessSecurityConfig.java

De seguida, definem-se as políticas de segurança sobre a forma de anotações. A listagem 58 exemplifica a definição da anotação `@MatchesReceiverParticipantId`, que verifica se o utilizador que está a tentar invocar uma determinada operação corresponde ao destinatário final da fatura identificada pelo `invoiceNumber`, `issueDate` e `sellerElectronicAddress`.

Muito sucintamente, o símbolo "#" permite aceder aos argumentos presentes no método anotado e o símbolo "@", em conjunção com o *Bean Resolver* do Spring, permite

[1] https://docs.spring.io/spring-framework/docs/3.0.x/reference/expressions.html
[2] https://docs.spring.io/spring-security/site/docs/4.2.x/reference/html/el-access.html

```java
package pt.opensoft.saftpro.configuration.einvoicing.security.policy;

import ...

@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Inherited
@Documented
@PreAuthorize(MatchesReceiverParticipantId.condition)
public @interface MatchesReceiverParticipantId {
    String condition = "@SecurityService.matchesReceiverParticipantId(#invoiceNumber, #issueDate,
            #sellerElectronicAddressSchemeId, #sellerElectronicAddress, principal)";
}
```

Listagem 58: SAFTPRO *Webservices* - MatchesReceiverParticipantId.java

invocar métodos de um determinado *Bean* instanciado no Spring *Container*. Neste caso, a implementação das regras de segurança é isolada no `SecurityService` (listagem 59), que tem acesso total ao modelo de dados.

```java
package pt.opensoft.saftpro.service.einvoicing;

import ...

@Service("SecurityService")
public class SecurityService {

    private PeppolParticipantJpaRepository peppolParticipantJpaRepository;
    private InvoiceJpaRepository invoiceJpaRepository;
    ...
    @Transactional(transactionManager = "eInvoicingTransactionManager", rollbackFor = Exception.class)
    public boolean matchesReceiverParticipantId(String invoiceNumber, Date issueDate,
                                                String sellerElectronicAddressSchemeId, String sellerElectronicAddress,
                                                Authentication user) {
        Optional<PeppolParticipantEntity> participantEntity =
                peppolParticipantJpaRepository.findByUsername(user.getName());

        Optional<InvoiceEntity> invoiceEntity = invoiceJpaRepository
                .findByInvoiceNumberAndIssueDateAndSellerElectronicAddress(invoiceNumber,issueDate,
                                                                           sellerElectronicAddressSchemeId,
                                                                           sellerElectronicAddress);

        if (participantEntity.isPresent() && invoiceEntity.isPresent()) {
            BuyerEntity buyer = invoiceEntity.get().getBuyer();

            String receiverParticipantId =
                    IdentifierUtil.createPeppolParticipantIdAsString(buyer.getElectronicAddress(),
                            buyer.getElectronicAddressSchemeId());

            return receiverParticipantId.equals(participantEntity.get().getParticipantId());
        }

        return false;
    }
    ...
}
```

Listagem 59: SAFTPRO *Webservices* - SecurityService.java

Para terminar, basta colocar devidamente as anotações nos métodos que queremos proteger (linhas 10 e 14):

```
1  package pt.opensoft.saftpro.restcontroller.einvoicing;
2
3  import ...
4
5  @RestController
6  @RequestMapping("/rest")
7  public class InvoiceStatusController {
8      ...
9      @PutMapping(value = "/bearer/e-invoicing/invoices/{invoiceNumber}/status", ...)
10     @MatchesReceiverParticipantId
11     public ResponseEntity<SaftproModel<InvoiceStatusModel>> updateStatus(...) {...}
12
13     @GetMapping(value = "/bearer/e-invoicing/invoices/{invoiceNumber}/status", ...)
14     @MatchesReceiverParticipantId
15     public ResponseEntity<SaftproModel<InvoiceStatusModel>> findByInvoiceIdAndEffective(...) {...}
16 }
```

Listagem 60: SAFTPRO *Webservices* - Políticas de segurança InvoiceStatusController.java

XI

OCDE *Lang*

Este anexo demonstra alguns exemplos da manipulação de dados que é possível efetuar ao nível da DSL OCDE *Lang*.

Exemplo 1: Adicionar um novo campo

É necessário acrescentar ao elemento `Delivery` (listagem 61), o campo `DeliveryPartyName` do tipo `String`, cujo mapeamento corresponde à expressão XPath "Invoice/cac:Delivery/cac:DeliveryParty/cac:PartyName/cbc:Name":

1. Declarar o campo `field create DeliveryPartyName ...;`
2. Ajustar a expressão, porque em Java o *namespace default* não é reconhecido pelo objeto XPath. Ou seja, sempre que for declarado um elemento presente no *namespace default* deve utilizar-se a expressão "[local-name()='$nome_do_elemento$']". Neste caso, a expressão resultante é "/*[local-name()='Invoice']/cac:Delivery/cac:DeliveryParty/cac:PartyName/cbc:Name";
3. Remover o prefixo comum entre o elemento pai e o elemento filho. Por omissão, o elemento `Invoice` é considerado a *root* dos documentos e o seu prefixo é "/*[local-name()='Invoice]". No `Invoice`, o prefixo atribuído ao `Delivery` é "cac:Delivery", o que permite comprimir a expressão e chegar à tag final "cac:DeliveryParty/cac:PartyName/cbc:Name";
4. Acrescentar o tipo do campo e a sua obrigatoriedade.

É importante destacar que as *tags* não devem começar com "/", uma vez que podem existir situações onde o campo pertence a um determinado elemento, mas estruturalmente encontra-se presente noutro nó do documento. Por exemplo, o campo `TotalVatAmount` no elemento `DocumentTotals`. Nestes casos, é necessário alterar o nó considerado e adicionar o sufixo correto. Para o `TotalVatAmount` o ajuste é "../cac:TaxTotal", uma vez que o ".." permite-nos voltar atrás e reposicionar no nó "cac:TaxTotal".

No que diz respeito às relações entre diferentes elementos, a lógica é semelhante mas utiliza-se a operação "link create ...".

```
...
create element Delivery {
    field create ActualDeliveryDate tag "cbc:ActualDeliveryDate" date;
    ...
    field create DeliveryPartyName tag "cac:DeliveryParty/cac:PartyName/cbc:Name" string; // novo campo
    link create child InvoicingPeriod tag "../cac:InvoicePeriod" OneToOne;
    link create child Address tag "cac:DeliveryLocation/cac:Address" OneToOne;
};

create element DocumentTotals {
    field create SumLineNetAmount
        tag "cbc:LineExtensionAmount[@currencyID=/*[local-name()='Invoice']/cbc:DocumentCurrencyCode]" float NotNull;
    ...
    field create TotalVatAmount
        tag "../cac:TaxTotal/cbc:TaxAmount[@currencyID=/*[local-name()='Invoice']/cbc:DocumentCurrencyCode]" float;
    field create AccountingCurrencyTotalVatAmount
        tag "../cac:TaxTotal/cbc:TaxAmount[@currencyID=/*[local-name()='Invoice']/cbc:TaxCurrencyCode]" float;
};

create element Invoice {
    field create InvoiceNumber tag "cbc:ID" string NotNull;
    ...
    link create child Delivery tag "cac:Delivery" OneToOne;
    ...
    link create child Seller tag "cac:AccountingSupplierParty/cac:Party" ManyToOne;
    link create child Buyer tag "cac:AccountingCustomerParty/cac:Party" ManyToOne;
    ...
};
...
```

Listagem 61: Adição de um campo ao modelo de dados

Exemplo 2: Remover um campo

Para efeitos fiscais um país não acha relevante persistir a informação sobre o campo `LocationSchemeId` no elemento `Delivery` (listagem 62). Para proceder à sua remoção basta comentar ou eliminar o campo e voltar a fazer a geração automática das classes.

```
...
create element Delivery {
    field create ActualDeliveryDate tag "cbc:ActualDeliveryDate" date;
    field create LocationId tag "cac:DeliveryLocation/cbc:ID" string;
    // field create LocationSchemeId tag "cac:DeliveryLocation/cbc:ID/@schemeID" string;
    field create DeliveryPartyName tag "cac:DeliveryParty/cac:PartyName/cbc:Name" string;
    link create child InvoicingPeriod tag "../cac:InvoicePeriod" OneToOne;
    link create child Address tag "cac:DeliveryLocation/cac:Address" OneToOne;
};
...
```

Listagem 62: Remoção de um campo ao modelo de dados

Exemplo 3: Adicionar um novo elemento

Para criar um novo elemento basta utilizar a operação "create element $nome_do_elemento$" e declarar os campos que o constituem (listagem 63). O elemento `Invoice` corresponde à *root* e portanto o novo elemento será filho direto do `Invoice` ou terá como pai elementos filho do `Invoice`. Este último passo pode ser efetuado através da operação "link create child ..." no pai do novo elemento.

```
...
create element NewElement { ... };

create element Invoice {
    field create InvoiceNumber tag "cbc:ID" string NotNull;
    ...
    link create child Delivery tag "cac:Delivery" OneToOne;
    ...
    link create child NewElement tag "cac:NewElement" ManyToOne NotNull;
};
...
```

Listagem 63: Adição de um elemento ao modelo de dados

Exemplo 4: Remover um elemento

A remoção de um elemento é muito semelhante à remoção de um campo (listagem 64). No entanto, é também necessário ter atenção às relações onde o elemento aparece e proceder à sua remoção.

```
...
//create element OldElement { ... };

create element Invoice {
    field create InvoiceNumber tag "cbc:ID" string NotNull;
    ...
    link create child Delivery tag "cac:Delivery" OneToOne;
    ...
    //link create child OldElement tag "cac:OldElement" ManyToOne NotNull;
};
...
```

Listagem 64: Remoção de um elemento ao modelo de dados

XII

SAFTPRO Portais

XII.1 Concretização das páginas para consulta de faturas no *Backoffice*

Para que os funcionários de uma Autoridade Tributária possam consultar as faturas difundidas pelo módulo de *e-Invoicing* foram desenvolvidas à medida duas *portlets* na aplicação. Uma delas junta a lista de faturas submetidas na rede com um formulário de pesquisa para filtrar os documentos por critério e a outra limita-se a mostrar os detalhes da fatura selecionada.

A configuração de uma nova *portlet* requer alguns passos, dos quais serão destacados os considerados mais importantes.

Tudo começa no ficheiro `web.xml` (listagem 65) com a declaração de uma instância do `ViewRendererServlet`, cujo o propósito é converter *Portlet Requests* em *Servlet Requests* e *Portlet Responses* em *Servlet Responses* para tirar proveito das funcionalidades do Spring *Web* MVC.

De seguida, é preciso indicar a diretoria onde se encontram os *templates* utilizados para construir as páginas HTML mostradas ao utilizador. Neste projeto em concreto, o *template engine* utilizado é o Thymeleaf[1] e como exemplo pode-se consultar, na listagem 66, o *template* utilizado pela *Portlet* que disponibiliza a lista de faturas.

Após estes passos, é necessário indicar no ficheiro `portlet.xml` (listagem 67) as classes responsáveis por controlar o ciclo de vida útil das *portlets* e onde se encontram os seus contextos de configuração.

[1] https://www.thymeleaf.org/

```xml
<?xml version="1.0"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <context-param>
        <param-name>contextClass</param-name>
        <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationContext</param-value>
    </context-param>
    ...
    <servlet>
        <servlet-name>ViewRendererServlet</servlet-name>
        <servlet-class>org.springframework.web.servlet.ViewRendererServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>ViewRendererServlet</servlet-name>
        <url-pattern>/WEB-INF/servlet/view</url-pattern>
    </servlet-mapping>
    ...
    <servlet>
        <servlet-name>Liferay Thymeleaf</servlet-name>
        <servlet-class>com.liferay.portal.kernel.servlet.PortletServlet</servlet-class>
        <init-param>
            <param-name>portlet-class</param-name>
            <param-value>org.springframework.web.portlet.DispatcherPortlet</param-value>
        </init-param>
        <load-on-startup>0</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Liferay Thymeleaf</servlet-name>
        <url-pattern>/WEB-INF/templates/*</url-pattern>
    </servlet-mapping>
</web-app>
```

Listagem 65: SAFTPRO Portais - /WEB-INF/web.xml

```xml
<div xmlns:th="http://www.w3.org/1999/xhtml">
    <div id="app-root-element"></div>
    <script type="text/javascript" th:inline="javascript">
            /*<![CDATA[*/
            document.addEventListener('DOMContentLoaded', () => {
            let portletModel = {
                language: /*[[${language}]]*/ '',
                locale: /*[[${locale}]]*/ '',

                currencyLocale: /*[[${currencyLocale}]]*/ '',
                currencyPattern: /*[[${currencyPattern}]]*/ '',

                dateFormat: /*[[${dateFormat}]]*/ '',

                invoices: /*[[${invoices}]]*/'',
                restUrl: /*[[${search}]]*/ '',
                statusCatalog: /*[[${status}]]*/'',
                typesCatalog: /*[[${types}]]*/''
            };

            let appRootElem = document.getElementById("app-root-element");
            startBackofficeInvoiceListApp(appRootElem, portletModel);
        });
        /*]]>*/
    </script>
</div>
```

Listagem 66: SAFTPRO Portais - WEB-INF/templates/backoffice-invoice-list.html

```xml
<?xml version="1.0"?>
<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
            http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd" version="2.0">
        <portlet>
                <portlet-name>backoffice-invoice-list</portlet-name>
                <display-name>backoffice-invoice-list</display-name>
                <portlet-class>org.springframework.web.portlet.DispatcherPortlet</portlet-class>
                <init-param>
                        <name>contextClass</name>
                        <value>modules.core.spring.web.context.AnnotationPortletApplicationContext</value>
                </init-param>
                <init-param>
                        <name>contextConfigLocation</name>
                        <value>pt.opensoft.saftproportais.backoffice.invoice.list.config.InvoiceListConfig</value>
                </init-param>
                <expiration-cache>0</expiration-cache>
                <supports>
                        <mime-type>text/html</mime-type>
                </supports>
                <resource-bundle>content.Language</resource-bundle>
                <portlet-info>
                        ...
                </portlet-info>
                ...
        </portlet>
        ...
</portlet-app>
```

Listagem 67: SAFTPRO Portais - /WEB-INF/portlet.xml

No SAFTPRO Portais, as configurações são maioritariamente expressas em classes Java, através das anotações específicas disponibilizadas pelas bibliotecas e *frameworks*.

```java
package pt.opensoft.saftproportais.backoffice.invoice.list.config;

import ...

@Configuration
@Import({SaftproPortaisMvcPortletConfig.class, SaftproRestApiConfig.class})
@ComponentScan({"pt.opensoft.saftproportais.backoffice.invoice.list.controller",
        "pt.opensoft.saftproportais.backoffice.invoice.service"})
public class InvoiceListConfig {

}
```

Listagem 68: SAFTPRO Portais - InvoiceListConfig.java

À semelhança do `DispatcherServlet`, o `DispatcherPortlet` atua como *Front Controller* e regista os *handlers* responsáveis por processar uma determinada *Portlet Request*, pelo que o próximo passo é definir os *Controllers* responsáveis por tratar um determinado pedido.

No Spring MVC, os diferentes tipos de *Controller* podem ser separados através da anotação `@RequestMapping` juntamente com o valor do modo da *Portlet* (`VIEW`, `EDIT` ou `HELP`).

```java
package pt.opensoft.saftproportais.backoffice.invoice.list.controller;

import ...

@Controller
@RequestMapping("VIEW")
public class InvoiceListController {
    ...
    @RenderMapping
    public String view(Model model, RenderRequest renderRequest, RenderResponse renderResponse) throws Exception {
        ...

        return "backoffice-invoice-list";
    }

    @ResponseBody
    @ResourceMapping("search")
    public ModelAndView search(ResourceRequest resourceRequest) throws Exception { ... }
}
```

Listagem 69: SAFTPRO Portais - InvoiceListController.java

Neste projeto em concreto, é necessário fazer um passo adicional numa classe de configuração do *site Backoffice*, para acrescentar os novos conjuntos de páginas e em relação ao código React, este é compilado à parte e disponibilizado como um *bundle*, pelo que é necessário declarar no ficheiro `liferay-portlet.xml` a sua localização.

```xml
<?xml version="1.0"?>
<!DOCTYPE liferay-portlet-app PUBLIC "-//Liferay//DTD Portlet Application 7.2.0//EN"
        "http://www.liferay.com/dtd/liferay-portlet-app_7_2_0.dtd">
<liferay-portlet-app>
        <portlet>
                <portlet-name>backoffice-invoice-list</portlet-name>
                <requires-namespaced-parameters>false</requires-namespaced-parameters>
                <header-portal-css>
                    /o/saftpro-theme-internet/css/saftpro/bootstrap-datepicker/bootstrap-datepicker.min.css
                </header-portal-css>
                <header-portal-css>
                    /o/saftpro-theme-internet/css/saftpro/bootstrap-datepicker/bootstrap-datepicker3.min.css
                </header-portal-css>
                <header-portal-javascript>
                    /o/saftpro-theme-internet/js/common/bootstrap-datepicker/bootstrap-datepicker.min.js
                </header-portal-javascript>
                <footer-portlet-javascript>/static/bundles/backoffice-invoice-list.bundle.js</footer-portlet-javascript>
        </portlet>

        <portlet>
                <portlet-name>backoffice-invoice-details</portlet-name>
                ...
        </portlet>
        ...
</liferay-portlet-app>
```

Listagem 70: SAFTPRO Portais - /WEB-INF/liferay-portlet.xml

Para terminar, no código abaixo encontra-se exemplificado a exportação da função `startBackofficeInvoiceListApp()` que depois é utilizada no *template* HTML listado acima.

```
import React from 'react';
import ReactDOM from 'react-dom';
import ReactIntlProvider from "../utils/react-intl-provider";
import {reactIntlProvider} from '../utils/react-intl-provider-hook';

export const BackofficeInvoiceListApp = ({invoices, searchUrl, invoiceTypeCatalog, invoiceStatusCatalog}) => {
    return ( /* Código respetivo à webpage que permite consultar as faturas e fazer pesquisas*/ )
}

export function startBackofficeInvoiceListApp(rootElements, portletModel) { ... };

global.startBackofficeInvoiceListApp = startBackofficeInvoiceListApp;
```

Listagem 71: SAFTPRO Portais - backoffice-invoice-list-app.js