Pedro Manuel Pereira Xavier

Licenciado em Engenharia Informática

Plataforma de Despesas ARTSOFT

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: Mestre Filipe de Almeida Grangeiro
Diretor, TI - Tecnologia Informática, S.A.

Coorientador: Joaquim Francisco Ferreira da Silva
Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

Júri:

Presidente: Doutor Miguel Carlos Pacheco Afonso Goulão
Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

Arguentes: Doutor Rui Pedro da Silva Nóbrega
Professor Auxiliar, Faculdade de Ciências e Tecnologia da Universidade NOVA de Lisboa

Mestre Filipe de Almeida Grangeiro
Diretor, TI - Tecnologia Informática, S.A.

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Outubro, 2021

1

Introdução

1.1 Motivação e Contexto

Registar e gerir despesas em serviço pode tornar-se uma tarefa cansativa quando um colaborador de uma empresa realiza várias despesas por dia e tem a necessidade de as declarar à sua empresa. Tudo se tornaria mais simples se existisse uma plataforma onde o colaborador pudesse registar as despesas em serviço.

Esta dissertação é realizada em colaboração com a ARTSOFT[1], uma empresa especializada no desenvolvimento e comercialização de soluções tecnológicas de apoio à gestão empresarial. Desde a sua fundação, em 1987, que o seu foco se encontra alinhado em grande parte com o mercado empresarial, passando pelo desenvolvimento de pesquisa para conseguir acompanhar as suas necessidades. Tendo um dos seus colaboradores da ARTSOFT apresentado o problema referido anteriormente, foi-me proposto a criação de uma plataforma *standalone* que efetue a gestão de despesas de uma empresa.

A empresa ARTSOFT deseja num futuro próximo reestruturar os seus serviços e dispô-los como um único servidor Web. Para se proceder a essa mudança é necessário um estudo das várias tecnologias e arquiteturas existentes, de forma a ser possível extrair as vantagens e desvantagens das mesmas e se proceder a uma escolha informada. Com o desenvolvimento desta dissertação a empresa ficará com o conhecimento de como funcionam determinadas tecnologias e onde as mesmas podem ser melhor aplicadas.

1.2 Definição do problema

Tal como referido anteriormente nesta dissertação, a definição do problema está direcionada pela implementação de uma plataforma de gestão de despesas em serviço. Desta forma é necessário que a plataforma criada permita a inserção de vários tipos de despesas por parte de um colaborador das diversas empresas.

Para que seja possível ao colaborador receber o estorno de cada despesa em serviço, é necessário que os gastos realizados pelo mesmo sejam posteriormente aprovados por um

[1] https://www.artsoft.pt/

utilizador com um grau hierárquico superior. Por consequência foi criado um sistema de aprovação de despesas.

Após a aprovação de uma despesa pode-se finalmente proceder ao pagamento da mesma, sendo necessário que o colaborador a quem pertence essa despesa tenha acesso ao estado da mesma durante todo este processo.

A plataforma deve possibilitar a criação de regras específicas para cada empresa em questão. Uma vez que as metodologias podem variar de empresa para empresa, é importante que estas possam alterar determinadas especificações para se sentirem confortáveis no seu modelo de trabalho.

1.3 Objetivos e solução proposta

O grande objetivo desta dissertação é a criação de uma plataforma para a gestão de despesas dos colaboradores de diversas empresas. No entanto, para que este propósito seja alcançado será necessário atingir certos requisitos que se encontram enumerados de seguida.

Será necessário a criação de um servidor Web para gerir os seus utilizadores e os registos das despesas, como por exemplo a sua consulta, a sua submissão e as suas atualizações. Esse servidor terá de ser capaz de receber os pedidos por parte de um ou mais clientes, tratando depois de toda a sua parte lógica e também de comunicar com a base de dados para efetuar as pesquisas ou inserções. Um dos objetivos para este sistema é que a sua comunicação com a base de dados seja independente do Sistemas de Gestão de Bases de Dados (SGBD) utilizado. Ou seja, deve ser relativamente simples alterar o SGBD, apenas modificando algum *driver*.

No lado do cliente pretende-se a criação de uma aplicação web que funcione como *backoffice*, permitindo tanto registar as despesas dos colaboradores, como um suporte à parte administrativa, gerindo os registos de despesas efetuados pelos colaboradores e toda a parte das regras da empresa.

1.4 Estrutura do documento

O documento será dividido nos seis capítulos seguintes:

- **Introdução** - No primeiro capítulo é descrito o contexto e a motivação da dissertação, apresentando também a definição do problema bem como os objetivos propostos para a sua concretização.

- **Enquadramento** - Neste capítulo serão apresentados os conceitos relacionados com o estudo realizado, que ajudaram na escolha das técnicas para o desenvolvimento da solução. São estes os conceitos ligados aos servidores Web e à criação da interface para o utilizador para aplicações web.

- **Trabalho Relacionado** - No terceiro capítulo será apresentado um estudo feito em conjunto com o Gestor de Produto da ARTSOFT onde foi realizada uma análise de domínio com a identificação de tecnologias e funcionalidades já existentes.
- **Conceção da Solução** - Neste capítulo será pormenorizada a solução proposta, evidenciando as técnicas e ferramentas utilizadas, tendo em conta o estudo apresentado no capítulo 2.
- **Implementação** - No quinto capítulo é especificado de modo mais detalhado a implementação das funcionalidades da plataforma.
- **Conclusões e Trabalho futuro** - No sexto e último capítulo serão expostas as conclusões da dissertação, assim como o trabalho que poderá ser desenvolvido futuramente de modo a melhorar o projeto criado.

2

Enquadramento e Background

2.1 Serviços Web

Serviço Web é uma tecnologia que permite a partilha de dados a partir de mensagens bem estruturadas, utilizando um protocolo padrão da internet, na maioria das vezes HTTP. Estas tecnologias permitem que diferentes clientes em várias partes do mundo possam aceder a esses dados através da internet sem terem qualquer conhecimento da estrutura interna do serviço Web.

Existem várias arquiteturas sobre as quais se podem projetar serviços Web, sendo as mais conhecidas o Simple Object Access Protocol, em português Protocolo Simples de Acesso a Objetos (SOAP) e REpresentational State Transfer, em português Transferência Representacional de Estado (REST). Relativamente ao SOAP será feita apenas uma breve abordagem, enquanto que será dado maior ênfase ao REST, tendo em conta as suas vantagens e o facto da sua arquitetura ser utilizada nesta dissertação.

2.1.1 SOAP

SOAP é um protocolo de comunicação para partilha de dados em serviços Web [14]. O seu desenvolvimento foi projetado para que a transmissão de mensagens fosse executada por vários protocolos alternativos, independente de qualquer modelo de programação.

Este protocolo utiliza a linguagem eXtensible Markup Language, em português Linguagem de Marcação Extensível (XML) para definir a estrutura das mensagens. Sendo esta composta por um *envelope* que contém um *cabeçalho* e um *corpo*. O *cabeçalho* é um elemento opcional que contém informações sobre a aplicação tal como a autenticação, endereçamento, confiabilidade e segurança. O *corpo* contém os detalhes da mensagem, ou seja, aquilo que queremos perguntar e receber como resposta [10].

Transparência e independência de protocolo de transporte são duas das vantagens do SOAP. Inicialmente o SOAP foi desenhado para operar com o protocolo HyperText Transfer Protocol, em português Protocolo de Transferência de Hipertexto (HTTP), mas podem existir cenários onde exista um benefício na utilização de qualquer outro protocolo e nesse caso o SOAP permite essa variedade.

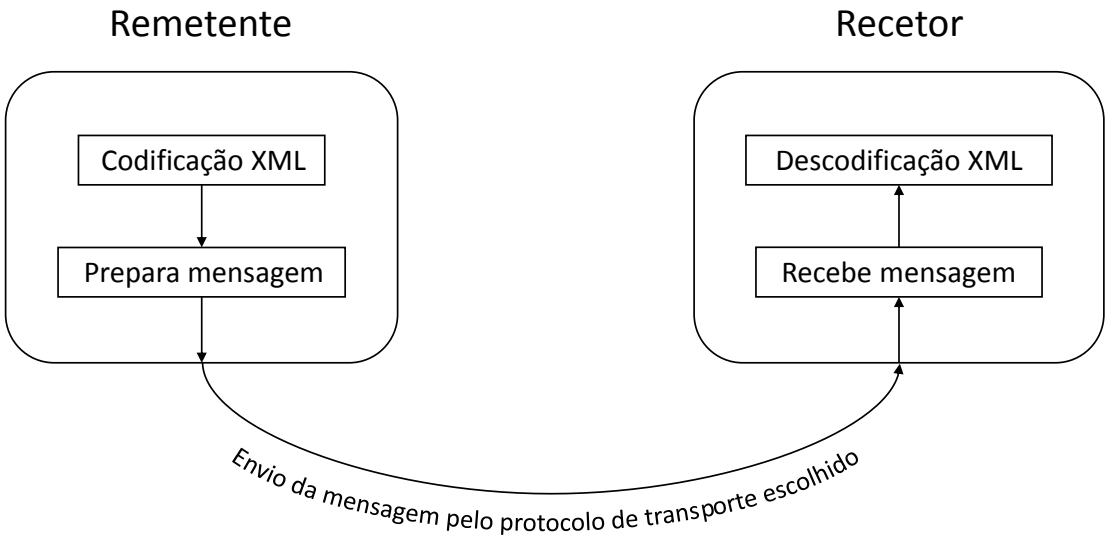

Figura 2.1: Arquitetura SOAP [17]

Uma vez que não existe um padrão sobre o conteúdo a inserir dentro dos *cabeçalhos* e do *corpo* da mensagem, é possível personalizar esses conteúdos de acordo com o desejo do utilizador. No entanto algumas aplicações web criaram alguns protocolos padrão para resolver determinados problemas. A vantagem desses protocolos está relacionada com o facto de ser possível escolher o que é mais adequado e com isso resulta outra vantagem do SOAP, que corresponde à extensibilidade [18].

Devido a ser uma arquitetura baseada em diferentes protocolos, a sua construção requer uma aprendizagem de todos os protocolos que podemos utilizar. Logo, pelo facto de ser demasiado complexo leva a que o nível de aprendizagem seja mais demorado [13, 18].

Como sabemos, as mensagens em SOAP são enviadas em ficheiros XML e por norma a sua estrutura é demasiado complexa, o que leva a que o envio da mensagem demore mais tempo, pois é necessário lidar com a complexidade do processamento das *tags* da mensagem XML [13].

2.1.2 REST

REST trata-se de uma abstração para projetar a construção de sistemas distribuídos baseados em comunicação via rede, cujos princípios utilizados se enquadram perfeitamente com a utilização do protocolo HTTP, que por sua vez, pode ser utilizado para a construção de serviços Web. O REST permite que um cliente comunique com um servidor, sem ter conhecimento prévio de onde esse servidor se encontra, bem como o que está guardado dentro dele.

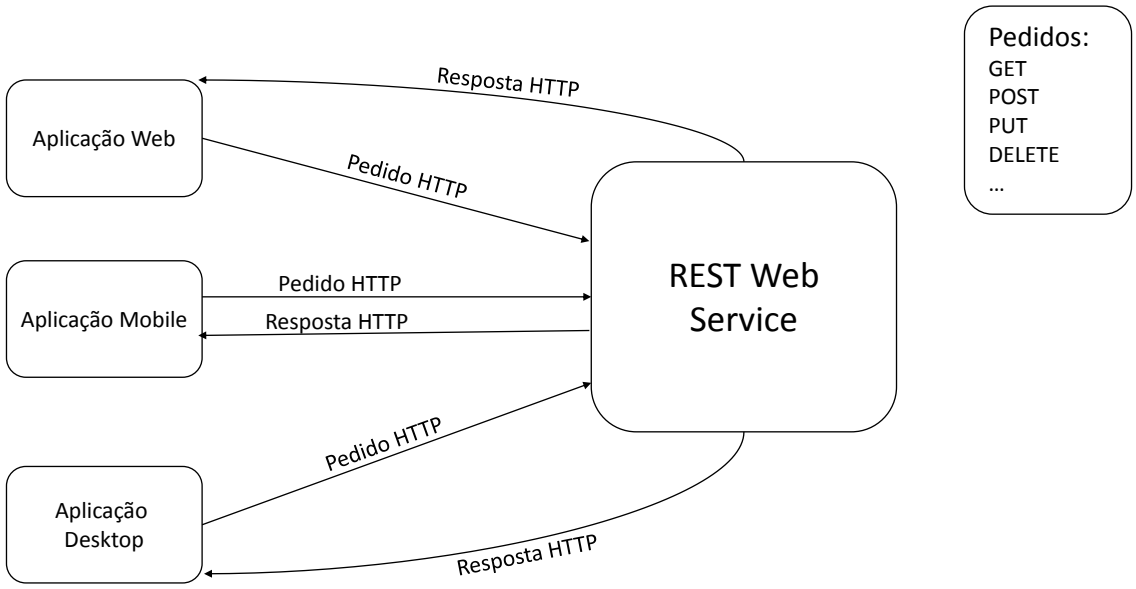

Figura 2.2: Arquitetura REST [3]

2.1.2.1 Elementos do REST [4]

Recursos. Todos os tipos de informação que podem ser consultados ou alterados podem ser considerados como recursos. Os recursos podem ser estáticos, caso estes nunca se alterem e possam apenas ser consultados, ou por sua vez podem ser não estáticos, se o seu valor poder ser alterado ao longo do tempo.

Identificador de recurso. São utilizamos identificadores de recursos para identificar um recurso específico durante uma interação entre componentes. Um identificador de recurso poderá referir-se a diferentes recursos, em momentos diferentes. Um cliente que faça uma pergunta a um serviço REST não precisa de saber com antecedência de que tipo é o recurso a qual o seu identificador está a especificar. A resposta deverá incluir metadados com os quais o cliente deverá conseguir interpretar os dados recebidos.

Representação de recursos. Um cliente e um servidor executam ações sobre um determinado recurso, mas não é o recurso em si que é transferido do servidor para o cliente, ou vice-versa. É sim uma representação desse recurso que lhes fornece o estado atual ou um estado projetado para esse recurso, sendo que essa representação é enviada nas mensagens.

Uma representação consiste em dados e em metadados que descrevem os dados. A metadata pode-se dividir em metadados de representação, que fornece informação sobre a representação que está a ser transmitida, como a data de criação, o número da versão. Ou podem ainda ser metadados sobre os recursos, que devolvem informações adicionais sobre um determinado recurso.

É considerado boa prática o servidor devolver sempre o mesmo tipo de representação, independentemente do cliente que lhe fez o pedido, sendo que faz parte do trabalho do cliente interpretar a resposta que o servidor deu. Isto porque para tornar o desempenho do servidor mais eficiente garantido escalabilidade, não convém estar a processar os pedidos de cada cliente. Num sistema com milhões de clientes essa opção seria insuportável.

Um recurso pode ser representado de diversas maneiras, sendo que as mais comuns são XML, JavaScript Object Notation, em português Notação de Objeto JavaScript (JSON) ou HyperText Markup Language, em português Linguagem de Marcação de Hipertexto (HTML).

2.1.2.2 Princípios do REST

O estilo arquitetónico REST é baseado em quatro princípios, sendo eles [10]:

Identificação de recursos através do Uniform Resource Identifier, em português Identificador Uniforme de Recurso (URI). Um serviço Web RESTful permite ao cliente uma interação com o mesmo, tendo este de identificar os recursos específicos a que se está a referir. Os recursos são assim identificados por URIs, que fornecem um espaço de endereçamento global para o seu reconhecimento.

Interface uniforme. Os clientes podem manipular esses recursos via requisições HTTP, sendo que as quatro mais famosas são GET, POST, PUT e DELETE.

- GET é uma solicitação que não altera o estado do recurso, apenas é utilizado para obter os dados do mesmo.
- POST cria um novo recurso com os dados fornecidos pelo cliente.
- PUT atualiza os dados de um determinado recurso, substituindo-os pelos novos apresentados pelo cliente.
- DELETE serve para eliminar um determinado recurso.

A utilização das solicitações POST e PUT varia um pouco de desenvolvedor para desenvolvedor, alguns utilizam as normas anteriormente apresentadas, enquanto outros preferem realizar essas solicitações de maneira oposta, o seja, o POST passa a ser uma solicitação de atualização e o PUT uma de criação.

O facto de considerar que o PUT é uma solicitação de atualização deve-se ao facto de o mesmo ser idempotente, ou seja, a sua repetida solicitação é equivalente ao caso em que apenas se solicita uma vez. Enquanto que a utilização repetida do POST cria tantos recursos como aqueles que foram solicitados pelo POST [11].

Mensagens auto-descritivas. As mensagens do cliente para o servidor devem conter uma descrição pormenorizada, de forma a que o servidor consiga processar a mensagem. Essas informações enviadas pelo cliente terão várias utilizações, por exemplo controlar a cache, detetar erros de transmissão, verificar a autenticação e controlo de acessos.

Interações com estado através de *hiperlinks*. As interações com um determinado recurso não têm estado, ou seja, não existe nenhum conhecimento prévio dos dados que se encontram no servidor, da sua estrutura, nem de como este está organizado. Com isto o estado pode ser integrado nas mensagens de resposta que o servidor envia ao cliente, informando o mesmo de possíveis maneiras de alterar esse estado através dos *hiperlinks*. Um exemplo desses *hiperlinks* são as URIs que o cliente pode utilizar em próximos pedidos, de modo a descobrir mais informações sobre determinado recurso.

2.1.2.3 Vantagens

Um serviço Web RESTful é considerado simples pois apresenta uso de padrões já conhecidos pela generalidade dos utilizadores, como HTTP, JSON, XML. O facto de as principais linguagens de programação suportarem clientes e servidores HTTP, traz mais uma vantagem para a criação de serviços Web RESTful, uma vez que este se baseia maioritariamente em HTTP.

Outro motivo que leva à sua adoção deve-se pela possibilidade de efectuar a sua construção com um número reduzido de ferramentas, tornando o sistema barato e com um menor esforço de implementação. Um desenvolvedor pode começar a criar um serviço utilizando ferramentas simples, como por exemplo o seu browser, não sendo necessário o desenvolvimento inicial de um software muito complexo.

O facto de um serviço Web RESTful ser *stateless* (sem estado), oferece a possibilidade de atender um grande número de pedidos do lado do cliente, devido ao suporte para armazenamento em cache e aos formatos de mensagens mais leves, como é o caso do JSON.

2.1.2.4 Desvantagens

Apesar de REST ser uma arquitetura com os seus padrões, existe alguma discordância entre os desenvolvedores sobre as práticas da sua implementação, tal como a que foi apresentada sobre a utilização do verbo POST ou PUT. Existe também quem não utilize todos os métodos de requisição HTTP e opte simplesmente por utilizar GET para solicitações idempotentes e POST para o resto das solicitações.

Todas essas diferentes alternativas levam a um esforço adicional pelos desenvolvedores Web no que toca à elaboração e testes das suas implementações, pois têm de estar cientes de como funciona o serviço a que estão a fazer um pedido.

Para solicitações com grande quantidade de dados de entrada, pode acontecer que o servidor rejeite as URIs do recurso por estarem mal formadas, ou na pior das hipóteses poderá dar-se o *crash* expondo o serviço a ataques do tipo *buffer overflow*.

2.1.3 Segurança

Quando são desenvolvidos serviços Web é necessária uma forma segura de autenticar os utilizadores que fazem pedidos ao serviço em questão, verificando se podem ou não aceder aos recursos que estão a solicitar. Para tal pode desenvolver-se uma solução que cuide do formato de autenticação onde recebe o utilizador e a sua palavra passe encriptada. No entanto existe um padrão estabelecido a nível mundial chamado OAuth.

O padrão OAuth 1 foi criado por Blaine Cook em 2006 numa solução de autenticação para o *Twitter* e um ano depois expandida à comunidade pela *Google*. Essa primeira versão era vista por muitos de bastante complexidade de implementação e com regras bastante fixas [15]. Devido às desvantagens apresentadas pela comunidade, foi desenvolvida uma segunda versão, o OAuth 2, que será abordado com mais detalhe.

O OAuth 2 suporta diferentes tipos de subvenções (*grants*), abordadas no ponto seguinte, sendo que estas são apenas conceitos que os desenvolvedores podem aplicar, dessa forma não é declarado como deve ser implementado, mas sim quais os conceitos que devem ser aplicados e é o desenvolvedor que decide qual deles é o mais adequando para a sua realidade.

Antes de entrar em detalhe sobre quais as possíveis *grants* a serem aplicadas, devemos abordar primeiro as terminologias aplicadas ao OAuth 2 [20].

- Proprietário do Recurso: Utilizador que quer ter acesso a um determinado recurso protegido.
- Cliente: Aplicação Cliente que solicita acesso a um recurso protegido em nome do Proprietário do Recurso.
- Servidor de Recursos: O servidor que gere os recursos protegidos. É a este serviço que é feito o pedido.
- Servidor de Autenticação: É este servidor que verifica se o Proprietário do Recurso pode aceder a determinado recurso e emite *Tokens* de Acesso após obter a as credenciais corretas.
- Agente do Utilizador: Agente usado pelo Proprietário do Recurso para interagir com o Cliente, por exemplo, um browser ou uma aplicação móvel onde corre a aplicação Cliente.

2.1.3.1 Authorization Code Grant (Grant Código de Autorização)

Este tipo de *grant* é utilizado para clientes confidenciais, pois a aplicação cliente não tem acesso a nenhuma credencial do proprietário do recurso. Ele é fundamentado em

redireccionamento, o cliente deve ser capaz de interagir com o proprietário do recurso, fazendo com que ele seja capaz de comunicar com um servidor de autenticação conhecido e no qual ele tenha confiança [5].

Um exemplo da sua utilização, é quando queremos entrar numa aplicação e ela nos pergunta se pretendemos fazer login com a nossa conta da *Google* ou do *Facebook*. Como conhecemos e confiamos nessas duas empresas faz com que aceitemos fazer o login através delas e não facultamos os nossos dados a essa outra aplicação que pretendemos aceder.

Pelo diagrama de sequências da figura 2.3 é possível observar o fluxo da sua implementação. Em primeiro lugar o proprietário do recurso pretende aceder à aplicação e pede pelo seu login. O cliente comunica com um servidor de autenticação externo (pelo exemplo anterior o da *Google* ou do *Facebook*). Esse servidor irá disponibilizar uma página à parte para o proprietário do recurso disponibilizar as suas credenciais. Depois de as mesmas serem preenchidas de forma correta, o servidor de autenticação devolve um código de autorização ao cliente e, com esse código juntamente com a sua *secret*, já podemos pedir o *token* de acesso ao servidor de autenticação. Quando o mesmo é devolvido para o lado do cliente é finalizado o login e já é possível fazer pedidos à Application Programming Interface, em português Interface de Programação de Aplicações (API) da nossa aplicação.

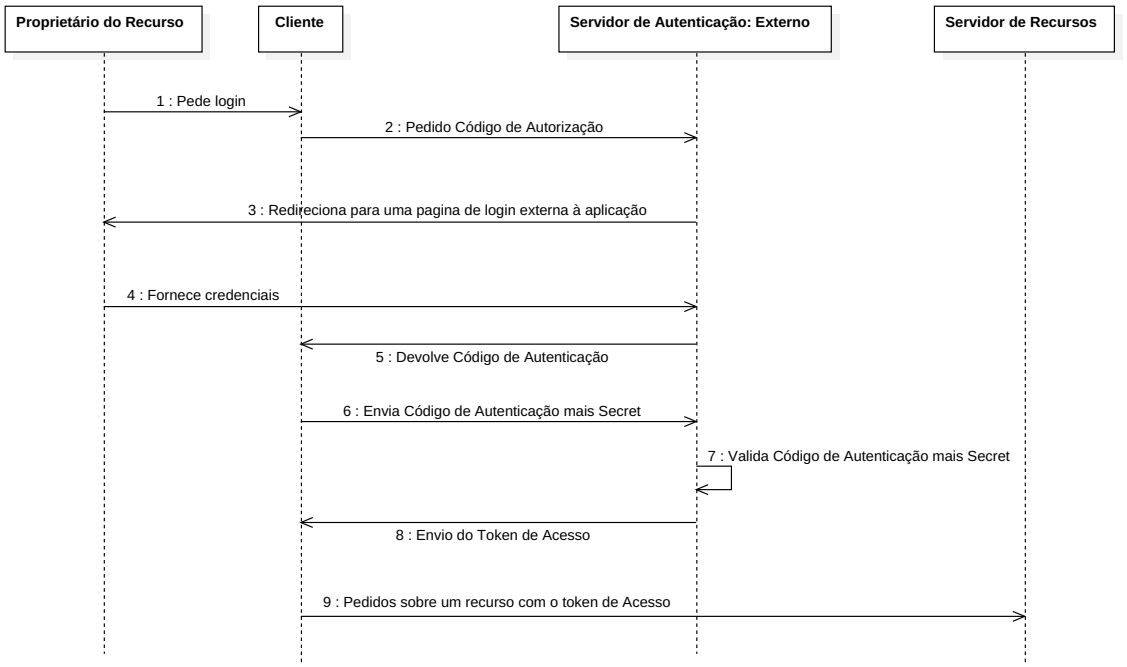

Figura 2.3: Diagrama de Sequências da abordagem Authorization Code Grant [1]

2.1.3.2 Implicit Grant (Grant Implícita)

Esta *grant* é semelhante à apresentada anteriormente (Código de Autorização), mas mais simples onde é reduzido o número de pedidos entre cliente e servidor de autenticação.

Nesta implementação, em vez de obtermos um código de autorização do servidor de autenticação, o cliente recebe o *token* de acesso assim que faz o login [5].

Esta abordagem é ideal para aplicações *single-page* (página única), uma vez que elas atuam fazendo apenas pedidos à API e não armazenam informações no lado do cliente, logo seria impossível guardar a *secret* do cliente neste tipo de aplicações [20].

Apesar de ser uma abordagem mais eficiente, devido ao menor número de troca de pedidos, a nível de segurança não é a melhor abordagem uma vez que o *token* de acesso é exposto do lado do cliente.

Podemos observar como se comporta esta abordagem pelo diagrama de sequências da figura 2.4 que, como já foi anteriormente referido, é bastante semelhante à *grant* Código de Autorização, retirando a receção do código de autorização e o envio do mesmo juntamente com a *secret* do Cliente para o servidor de autenticação.

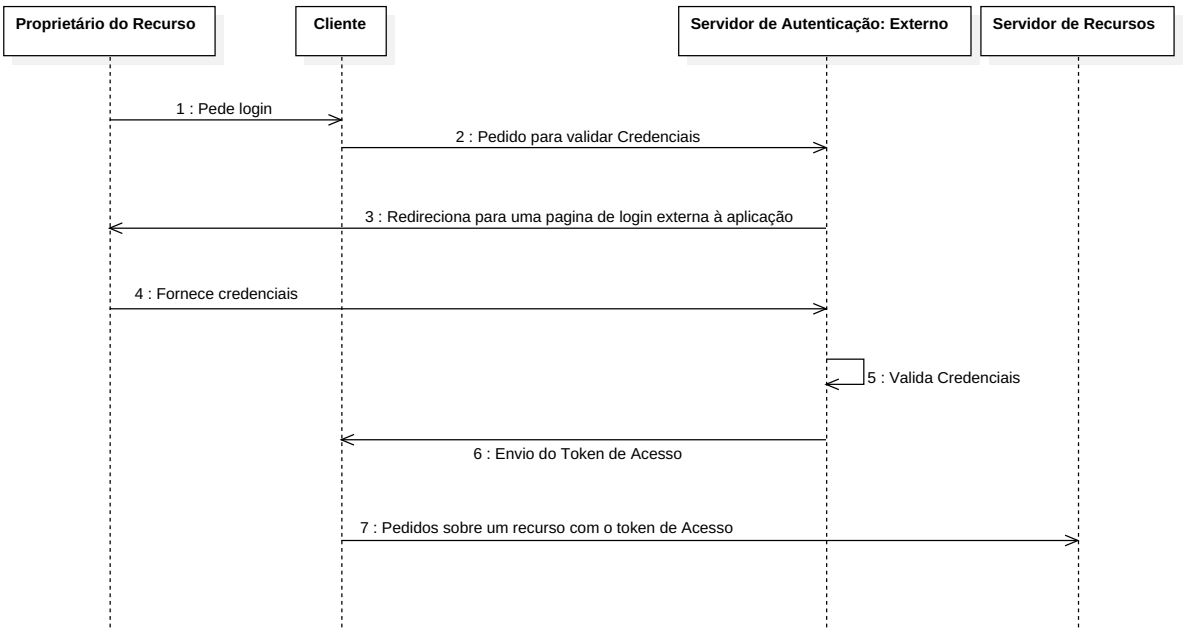

Figura 2.4: Diagrama de Sequências da abordagem Implicit Grant [6]

2.1.3.3 Resource Owner Password Credentials Grant (Grant Credenciais Proprietário do Recurso)

Esta abordagem apenas deve ser utilizada em casos onde o proprietário do recurso confia absolutamente no cliente, isto porque este vai ter acesso às suas credenciais como nome de utilizador e password [5].

Nesta *grant* o cliente cria o seu próprio formulário de login, onde o proprietário do recurso fornece as suas credenciais e as mesmas são enviadas para o servidor de autenticação. Geralmente este servidor de autenticação é gerido pela mesma entidade que gere

o servidor de recursos, não sendo neste caso um servidor de autenticação externo como o da *Google* ou do *Facebook* do exemplo anterior.

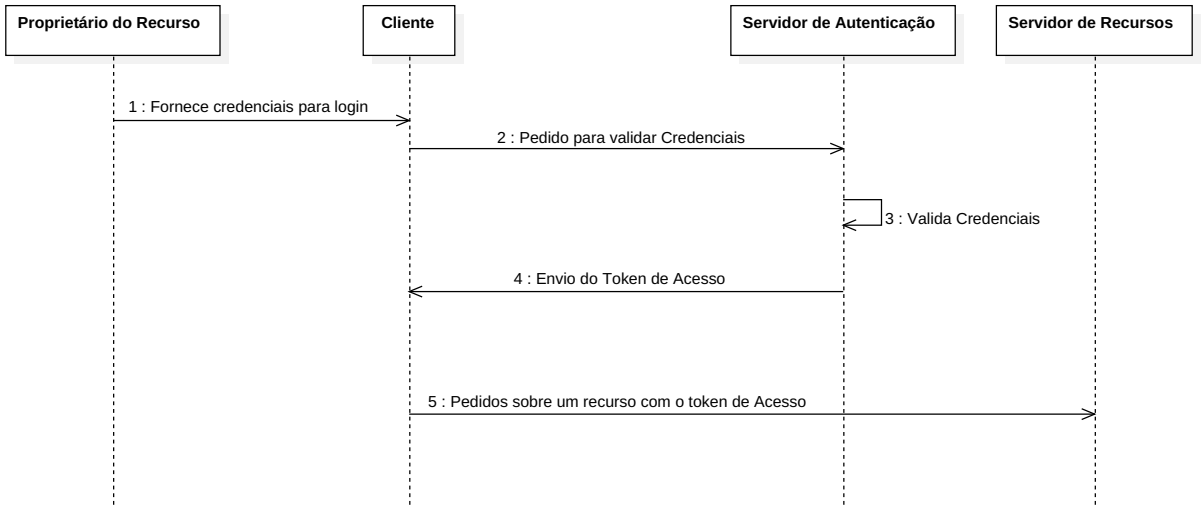

Figura 2.5: Diagrama de Sequências da abordagem Resource Owner Password Credentials Grant [12]

Depois do servidor de autenticação validar as credenciais enviadas pelo cliente o mesmo fornece o *token* de acesso permitindo agora o cliente de fazer pedidos ao serviço de recursos sobre determinados recursos. Podemos observar esse fluxo no diagrama de sequências da figura 2.5.

2.1.3.4 Client Credentials Grant (Grant Credenciais do Cliente)

Este tipo de *grant* é utilizado em comunicações entre servidores, onde não existe um proprietário do recurso específico, mas sim uma aplicação ou uma API que quer ter acesso a outra API [20].

Imaginemos que existe um servidor que executa um *cron job* onde todos os dias tem de aceder a uma API da *Google*. Como essa comunicação vai acontecer dentro desse servidor, ele é o proprietário do recurso responsável por efetuar o pedido à API da *Google*, não faz sentido ter credenciais como password e nome de utilizador, basta então enviar as credencias guardadas no servidor, como a sua *secret* de cliente da *Google* [5].

Podemos observar pelo diagrama de sequências da figura 2.6 o fluxo desta abordagem. Como foi referido, o Cliente (aqui visto como *backend*) envia a sua *secret* e o seu *id*. Recebe do servidor de autenticação o *token* de acesso se as suas credenciais estiverem corretas. E feito isso pode efetuar os pedidos necessários ao servidor de recursos.

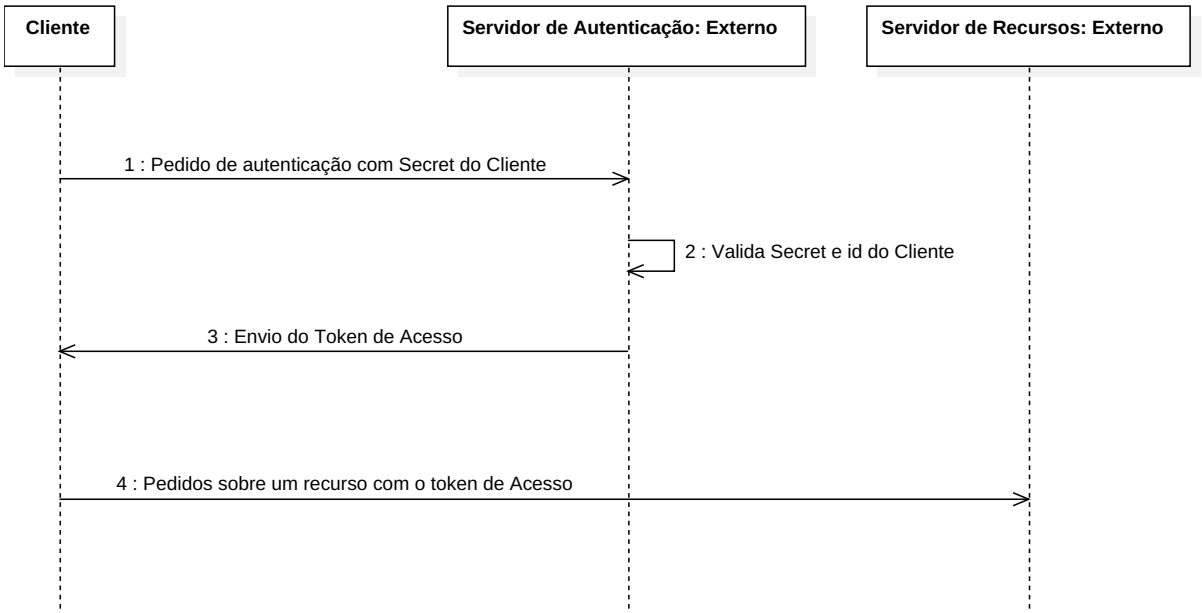

Figura 2.6: Diagrama de Sequências da abordagem Client Credentials Grant [2]

2.1.4 Tecnologias

Atualmente existem inúmeras frameworks que nos permitem acelerar o desenvolvimento e a implementação das nossas aplicações e, como tal, a sua utilização facilita em muito os desenvolvedores a atingirem os seus requisitos mais rapidamente. Com isto em mente, nesta secção serão exploradas algumas tecnologias para a implementação de um serviço Web.

2.1.4.1 Spring Boot

Spring Boot[1] é uma framework *open source* que visa facilitar o desenvolvimento de aplicações, isso porque não é necessária uma configuração complexa para se começar a desenvolver, bastando utilizar uma ferramenta disponível (*spring initializr*) e com uns poucos cliques é possível criar a base do projeto com todas as dependências que sejam necessárias. Sendo que se mais tarde for necessário adicionar uma nova dependência, este é bastante flexível, pois basta atualizar o ficheiro de gestão das dependências e não afeta de qualquer forma as dependências anteriormente instaladas.

Essas dependências englobam todas as funcionalidades necessárias para desenvolver um serviço Web e não só. Vão desde a segurança, acesso à base de dados, testes, cloud entre outras. É possível criar tanto um projeto Maven ou Gradle e a sua principal linguagem de programação é o Java.

[1] https://spring.io/

O Spring boot tem também padrões precisos sobre a forma como devemos organizar o nosso código dentro das pastas.

2.1.4.2 Node.js

Node.js[2] foi projetado para desenvolver aplicações que têm como requerimento uma necessidade de escalabilidade em rede, pois este é capaz de lidar com um grande número de conexões simultâneas em alto rendimento.

Comparado às técnicas tradicionais de serviços Web, nas quais cada pedido gera uma nova *thread*, ocupando a RAM do sistema e podendo atingir o limite máximo de RAM disponível, o Node.js opera como um ambiente de execução JavaScript assíncrono orientado a eventos, não bloqueando os pedidos que se sucedem, permitindo suportar dezenas de milhares de conexões simultâneas. Com isto a sua utilização não é recomendada se o servidor exigir operações que consumam muita RAM, uma vez que os pedidos efetuados de seguida serão bloqueados, pois o servidor estará ocupado a computar o pedido anterior.

Uma vez que o Node.js é implementado com JavaScript a sua utilização é muito mais vantajosa quando é utilizado um SGBD No-SQL, pelo simples facto de não ser necessário transformar os dados quando é comunicado com a base de dados, como é o caso de MongoDB, onde os dados podem ser armazenados em ficheiros JSON.

2.2 Base de Dados

2.2.1 Independência de Sistemas de Gestão de Bases de Dados

Num serviço Web é importante que exista um certo nível de abstração por cada fase de desenvolvimento, pois caso se altere alguma funcionalidade num determinado nível, os outros níveis abrangentes não têm também de sofrer alterações.

Neste segmento será discutida uma arquitetura que permite uma abstração de dados por parte do cliente e consequentemente a independência de SGBD.

A figura 2.7 mostra-nos uma arquitetura dividida em três camadas [19]:

- Nível do cliente
- Nível do serviço (*business logic*)
- Nível físico (interação com as bases de dados)

O **nível do cliente** é o nível com mais abstração sobre os dados, sendo responsável pelo desenvolvimento das aplicações que garantem a interação dos dados com o utilizador. Podem existir vários tipos de clientes responsáveis por extrair dados do sistema, que operam em diferentes linguagens de programação e dão suporte a diferentes tecnologias.

[2] https://nodejs.org/en/

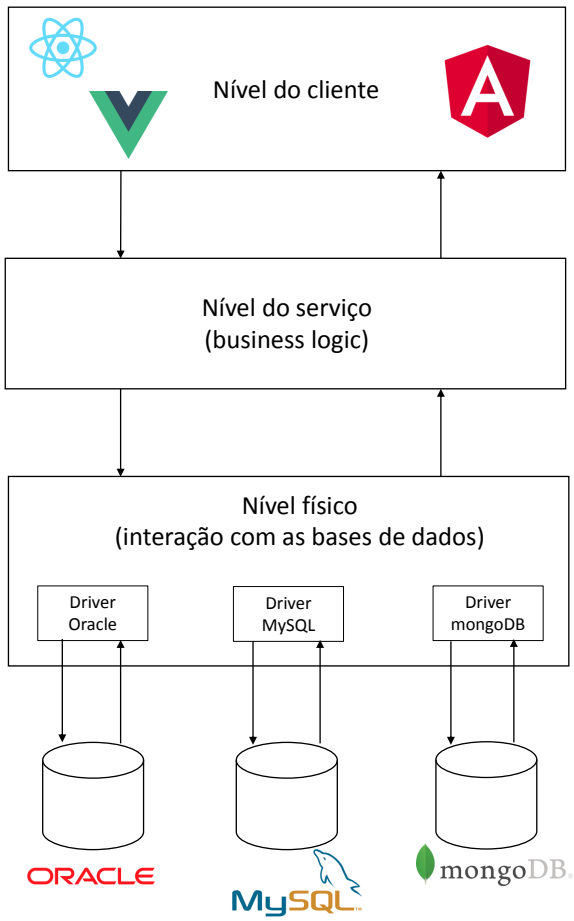

Figura 2.7: Arquitetura de independência SGBD

Estes fazem a interação com o nível mais abaixo através de um tipo de pedido específico (ex. HTTP).

No segundo nível, surge o **nível do serviço**, responsável por tratar de toda a parte lógica do serviço Web. Este recebe um pedido por parte do cliente, para consultar ou alterar algum tipo de dado e faz um pedido à próxima camada (geralmente chamando uma função) para efetuar determinada operação sobre um certo tipo de dados. É neste nível também que se desenvolve a parte de segurança do sistema, como quem pode ou não aceder a determinado tipo de dados.

Por fim existe o último nível, o **nível físico** que é o único a conhecer a especificidade de como os dados estão armazenados no SGBD. É neste nível que se fazem as *queries* ao SGBD de modo a obter ou alterar os dados. Caso seja necessário efetuar uma alteração ao esquema da base de dados, como por exemplo alterar ou adicionar uma nova entidade, apenas este nível irá sofrer alterações no modo como efetua as *queries*, pois a "função" chamada pelo nível anterior para efetuar uma operação sobre os dados será a mesma.

Com esta abordagem conseguimos também garantir que o servidor consegue operar com diferentes SGBD. Mas para isto será necessário criar diferentes módulos dentro deste

nível para poder operar com os diferentes SGBD, uma vez que o esquema e a linguagem de consulta são diferentes de SGBD para SGBD.

Supondo que o nosso serviço foi comprado por uma nova empresa e esta pretende utilizar o SGBD mongoDB, ao invés do que estávamos a utilizar anteriormente que era PostgreSQL. Com esta arquitetura não é necessário alterar o serviço todo devido ao novo formato de armazenamento de dados, mas apenas necessitamos de criar um novo módulo dentro deste nível para efetuar as consultas ao novo SGBD.

2.2.2 Sistemas de Gestão de Base de Dados

Apesar de nos requisitos desta dissertação ser proposta uma abordagem que permita a utilização de um qualquer Sistema de gestão de Base de dados, serão destacadas de seguida algumas opções que existem.

2.2.2.1 MySQL

MySQL[3] é um dos sistemas de base de dados relacional *open source* mais populares do mundo. É um sistema bastante confiável, simples de utilizar e compatível com quase todos os sistemas de hospedagem.

As transações em MySQL funcionam como uma única unidade, ou seja, a transação só será concluída com êxito assim que todas as operações individuais da mesma sejam concluídas. Portanto, se uma operação falhar, a transação inteira irá também falhar, fazendo com que todas as operações concluídas até então sejam retrocedidas. Esta é uma vantagem positiva para aplicações que lidam com transações de dinheiro. Por exemplo, o dinheiro de um cliente só será debitado se executar com sucesso todas as operações, caso contrário todo o processo é revertido [9].

O MySQL garante uma elevada disponibilidade, ao fornecer uma grande variedade de *clusters* e master-slave replication que garantem uma boa tolerância a falhas para acessos ininterruptos. Ele foi desenhado para processar milhares de consultas e de transações por segundo, fornecendo também memoria em cache e índices de *full-text* a uma elevada velocidade.

A segurança do sistema sempre foi uma das preocupações da empresa, e a mesma garante uma elevada segurança dos dados com recurso a uma poderosa criptografia, que impede a visualização dos mesmos por pessoas não autorizadas e suporte de certificados SSH e SSL que garantem conexões mais seguras [9].

O MySQL é bastante rápido, independentemente da plataforma utilizada. Possui recursos de auto-gestão como reinicialização automática e configurações automáticas para facilitar a sua gestão. Disponibiliza também uma interface gráfica que permite ao utilizador uma boa usabilidade e um monitoramento do desempenho em tempo real através de um gráfico simples para gerir problemas operacionais.

[3] https://www.mysql.com/

2.2.2.2 Oracle

Oracle[4] é um sistema de base de dados relacional, possuindo algumas características exclusivas que a tornam numa das melhores do mundo. É um serviço bastante dispendioso, mas para muitos compensa o dinheiro investido, pois apresenta características bastante boas.

Hoje em dia a Oracle é utilizada pelas principais empresas do setor bancário para gerirem os seus dados, incluindo as dez primeiras a nível mundial. Com esta constatação podemos ver que a nível de segurança este SGBD é um dos melhores.

É suportado por mais de 100 plataformas de hardware e 20 protocolos de rede, possuindo também os recursos para ser desenvolvido num sistema operacional à escolha, garantindo uma elevada portabilidade.

Bastante bom no que toca a recuperação de falhas, uma vez que possui um dos recursos de backup e recuperação online mais rápidos do mercado. Logo, caso ocorra uma falha, assim que o sistema voltar a estar online são recuperados os dados predefinidos como backup [16].

A Oracle possui ainda um sistema de desempenho complexo com controlo de bloqueio de transações garantindo uma elevada velocidade e segurança, independentemente do volume de dados [16].

O SGBD da Oracle serve-se das características Atomicity Consistency Isolation Durability, em português Atomicidade Consistência Isolamento e Durabilidade (ACID) no que toca às transações dos dados garantindo uma alta integridade do armazenamento dos mesmos.

Oracle pode ser difícil para quem começa a aprender a sua utilização e é necessário bastante tempo para o utilizador ficar familiarizado com as suas características e desta forma conseguir implementar um sistema em condições. Essa maior dificuldade de aprendizagem pode levar a uma pior performance por parte do sistema, uma vez que se for implementada alguma funcionalidade de forma descuidada originará uma pior performance [16].

2.3 Cliente

Quando é desenvolvida uma interface cliente, é também importante ter em conta as arquiteturas existentes para a sua implementação, uma vez que a sua escolha terá repercussões no futuro, podendo tornar a aplicação mais lenta e ser difícil efetuar a sua manutenção futura.

De seguida serão apresentadas alguns tipos de arquiteturas Graphical User Interfaces, em português Interfaces Gráficas do Utilizador (GUI) apresentadas por Martin Fowler [8].

[4] https://www.oracle.com/

2.3.1 Formulários e Controladores

Tal como o nome indica nesta arquitetura existe a distinção entre os formulários e os controladores. Os formulários são responsáveis por estruturar o posicionamento dos controladores no ecrã e também por definir a sua lógica de interação.

Este tipo de arquitetura é caracterizado na maioria das interfaces de desenvolvimento por permitir ao utilizador utilizar uma técnica de *drag and drop* que possibilita arrastar os controlos para dentro do formulário e estruturá-los da maneira desejada.

Os controladores gerem os dados, tanto aqueles que são exibidos na interface gráfica ao utilizador, como aqueles que são introduzidos pelo utilizador na aplicação. Como tal, na maioria dos casos existem três tipos de dados:

- **Dados registados**, são aqueles que se encontram guardados na base de dados.
- **Dados de sessão**, são aqueles que fazem a ligação entre a aplicação e a base de dados. Servem como versão temporária dos dados que serão alterados pelo utilizador, até serem guardados na base de dados.
- **Dados do ecrã**, são aqueles que são visíveis no ecrã da aplicação.

Com diversos tipos de dados é importante que os controladores consigam manter os dados sincronizados e para isso foi desenvolvida uma técnica, chamada *Data Binding*, que ajudou a tornar essa gestão mais simples. A sua funcionalidade é fazer com que sempre que um valor do ecrã é atualizado, atualize também o valor da sessão correspondente.

No caso de existirem controladores que dependam de outros controladores, será necessário utilizar uma lógica diferente, pois se o utilizador altera um dado do ecrã este pode ser atualizado ao nível da sessão através da *Data Binding*, mas não irá atualizar outro controlador que está a depender desses dados.

Para que essa atualização funcione é necessário que o formulário possa ser alertado sempre que um dado seja alterado e este chame algum género de função para atualizar os campos.

Um método para implementar esta funcionalidade é utilizar a noção de eventos. Cada controlador tem uma serie de eventos que pode gerar. O formulário fica à espera de que um desses eventos seja desencadeado e quando assim o verificar executa as operações que pretende, como por exemplo atualizar outros controladores.

2.3.2 MVC

O Model View Controller, em português Modelo Visão Controlador (MVC) é uma das arquiteturas de desenvolvimento de interfaces gráficas mais conhecido e mais citado pelos desenvolvedores.

No MVC pretende-se que exista uma abstração entre os objetos de domínio (aqueles que são uma representação real do objeto) e os objetos de visualização (aqueles que vemos no ecrã da nossa aplicação).

Chamamos de Modelo ao elemento que gere os objetos de domínio e não tem qualquer conhecimento da interface gráfica utilizada. Os objetos de visualização são tratados por dois elementos, a *View* e o *Controller*. A *View* é responsável por apresentar os dados ao utilizador e o *Controller* tem como função receber os dados inseridos por um determinado utilizador e saber o que fazer com os mesmos. Sendo assim existe um par *View Controller* por cada elemento apresentado na aplicação.

Para efetuar a *Data Binding*, a *View* e o *Controller* não comunicam diretamente um com o outro, ao invés disso observam o Modelo, sendo que este trata da lógica subjacente da aplicação e diz o que se deve alterar. Por exemplo, quando um utilizador altera uma caixa de texto o *Controller* desse *widget* informa o Modelo que o mesmo foi alterado. O Modelo por sua vez atualiza as *Views* correspondentes (por vezes mais do que uma) de acordo com a lógica da aplicação. O *Controller* desta forma desconhece o que irá alterar na aplicação. Podemos observar melhor o esquema desta arquitetura no diagrama de sequências na figura 2.8.

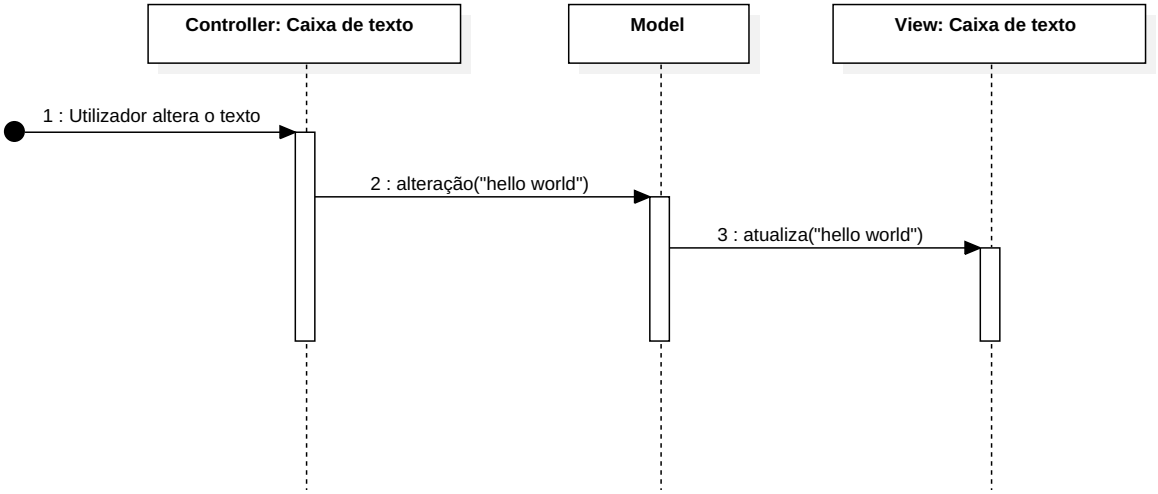

Figura 2.8: Diagrama de sequências do MVC [8]

2.3.3 Modelo de Aplicação

Este modelo é uma derivação do MVC, pois adquire algumas das suas principais características, o Modelo de Aplicação tem como principal diferença a criação de uma classe intermédia entre o Modelo e os *widgets* (*Controllers* e *Views*), essa nova classe é chamada de Modelo de Aplicação. Desta forma, os *widgets* não comunicam diretamente com o Modelo, mas sim com essa nova classe.

A utilização do Modelo de Aplicação permite o utilizador separar o estado da interface gráfica do estado real do Modelo do domínio. Ajuda-nos também a realizar as atualizações dos dados sem a necessidade de escrever muito código, pois a lógica encontra-se mais dividida.

Utilizando o exemplo do diagrama de sequências da figura 2.9, será explicado como funciona o modelo em questão.

Supondo que ao alterar o valor de uma caixa de texto, um outro *widget* tem também a necessidade de ser alterado. Neste modelo, quando o utilizador altera o valor da caixa de texto, o *Controller* comunica que o mesmo foi alterado ao Modelo de Aplicação correspondente a esse *widget*, alterando o seu valor guardado dentro da aplicação. De seguida, o Modelo de Aplicação irá comunicar com o Modelo do domínio e é este que altera o valor real desse *widget*. Como verificado no MVC, este Modelo é responsável pela lógica da aplicação e verifica que com a alteração deste é necessário a alteração de um outro *widget*. Este comunica com o Modelo de Aplicação desse novo *widget* que necessita de alteração e é o Modelo de Aplicação que vai alterar a *View* correspondente.

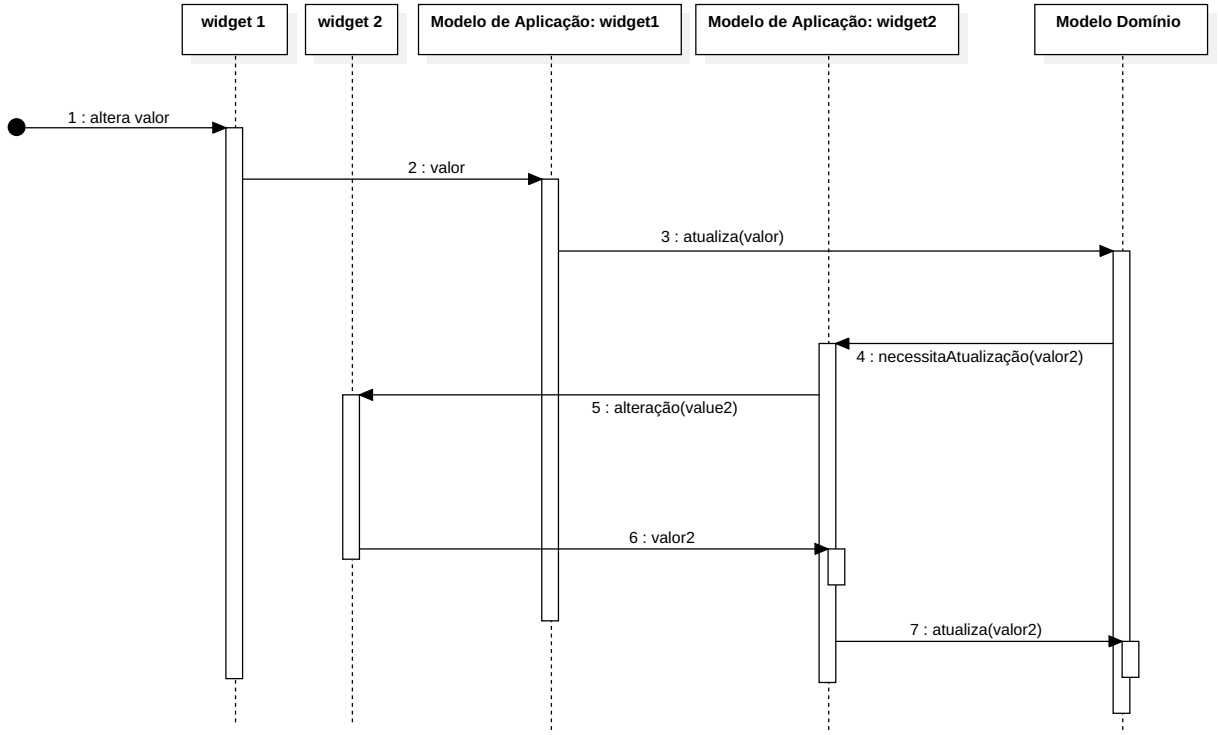

Figura 2.9: Diagrama de sequências do Modelo de Aplicação [8]

2.3.4 MVP

Desta forma o Model View Presenter, em português Modelo Visão Apresentação (MVP) pode ser abordado tendo em conta todas as outras arquiteturas apresentadas até ao momento, vendo as vantagens de cada uma e uni-las a este modelo. Por um lado, os Formulários e Controladores fornecem um design fácil de entender e faz uma boa separação entre os *widgets*. Vamos também associar as vantagens do MVC e das suas derivações, que nos apresenta uma separação bastante útil entre componentes.

o MVP é descrito em contraste com as outras arquiteturas da seguinte forma:

- Formulários e Controladores: o *Presenter* deve manipular o Modelo atualizando-o sempre que detetar um evento. Sendo que as atualizações de objetos devem passar sempre primeiro pelo *Presenter*.
- MVC: No MVP a *View* lida com a maior parte da lógica de visualização que pode ser descrita declarativamente e o *Presenter* lida com os casos mais complexos. Os *widgets* não são separados em *View* e *Controller*, mas o *Presenter* pode ser visto com um *Controller* do MVC, mas sem se preocupar com as ações do utilizador.
- Modelo de Aplicação: A *View* envia os eventos para o *Presenter*, assim como no Modelo de Aplicação. No entanto podem existir casos onde a *View* pode ser atualizada diretamente do Modelo de domínio não tendo a necessidade de passar pelo Modelo de Aplicação.

Tendo agora em conta o diagrama de Sequências da figura 2.10 pode ser observado como o MVP se comporta. Começa da mesma forma do Formulário e Controlador onde o *widget* gera um evento assim que este é alterado. O *Presenter* escuta esse evento e obtém o valor, sendo que de seguida atualiza o Modelo de domínio. Caso exista mais algum *widget* que tenha de ser alterado, o Modelo é o responsável por efetuar essa alteração. A última parte é a execução da lógica de visualização, que é feita pelo *Presenter*.

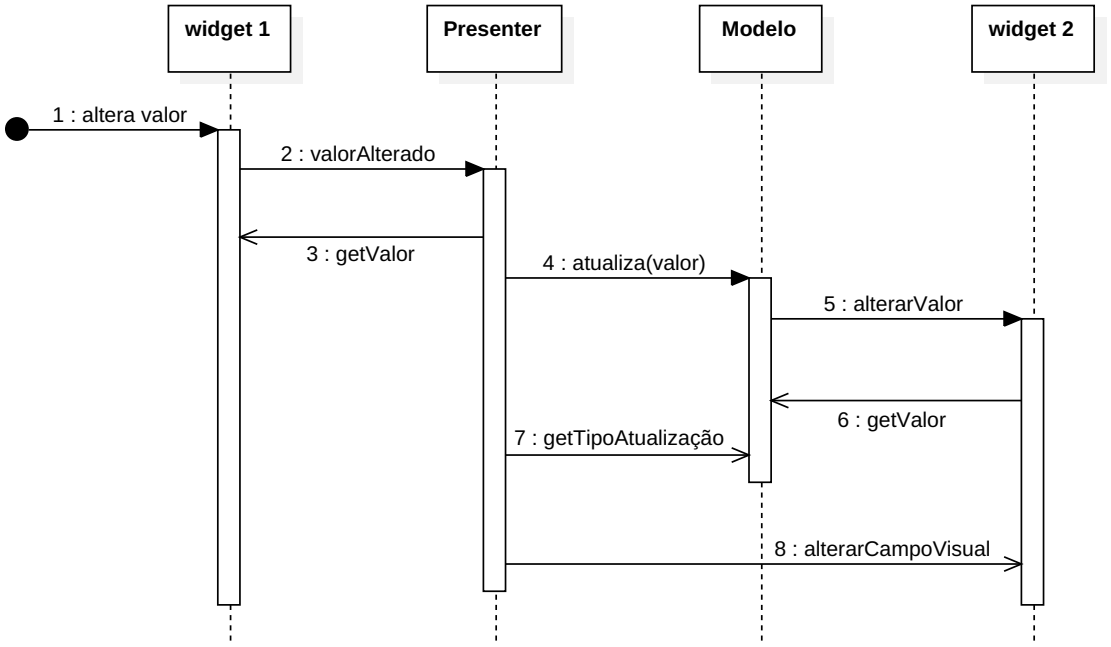

Figura 2.10: Diagrama de sequências do MVP [8]

Imaginemos que ao alterar um determinado valor numa caixa de texto esse faça com que se altere a cor desse mesmo texto. Por exemplo se for escrito "sim" o texto fica a verde e se escrevermos "não" o texto fica a vermelho. Neste caso é necessário que exista uma lógica na aplicação para lidar com a configuração da cor. Existem duas lógicas associadas

a este exemplo, a primeira é a lógica de domínio que corresponde a fazer a associação entre o texto "sim" (positivo), "não" (negativo) e o resto. Esta avaliação pode ser realizada pelo Modelo, pois este trata do domínio da aplicação. A outra lógica é a de visualização que cuida de mapear as cores para o tipo de texto correspondente e alterar o campo da *View*. O problema do MVC é saber onde colocar esta lógica de visualização. Com o MVP a lógica de domínio é realizada pelo Modelo e verificamos essa separação adicionando a lógica de visualização no *Presenter*.

2.3.5 Tecnologias

2.3.5.1 React

React[5] é uma biblioteca JavaScript desenvolvida pelo *Facebook* que tem como foco criar interfaces de utilizador *front-end* de *single page*. Apresenta uma curva de aprendizagem rápida, onde apenas com alguns dias, lendo a sua documentação e vendo alguns tutoriais de exemplo, é possível entender e aprender os princípios desta biblioteca. À medida que se começa a desenvolver uma aplicação os seus princípios vão-se tornando cada vez mais naturais e tornam-se simples.

A sua estrutura é baseada em componentes. As componentes são aquilo que dá forma à aplicação e podemos pensar nelas como *widgets*. O agrupar das várias componentes formam a aplicação. Cada componente tem a sua lógica interna e muitas vezes podem ser reutilizadas em vários locais da aplicação, reutilizando código e facilitando a sua manutenção.

O React fornece um elevado desempenho no que toca à renderização dos componentes. Hoje em dia os interpretadores de JavaScript são bastante poderosos e conseguem lidar bem com aplicações complexas, mas as atualizações do Document Object Model, em português Modelo de Documento por Objetos (DOM) não são assim tão eficientes e é nesse ponto que a maioria das aplicações *front-end* falha. O React resolve esse problema criando o chamado DOM virtual. Trata-se de um DOM mantido em memoria e sempre que existe uma atualização nos componentes elas são refletidas primeiro nesse DOM virtual. Só depois essas atualizações são enviadas para o DOM real e exibidas no ecrã, através de um algoritmo que calcula a melhor maneira de aplicar as alterações, comparando os estados anteriores e os atuais.

O React fornece uma abstração de arquitetura, isto é, um desenvolvedor é livre de implementar a arquitetura que achar mais apropriada.

Caso se pretenda desenvolver uma aplicação móvel para Android e iOS existe a possibilidade de o fazer utilizando React Native[6]. Suponhamos que começamos por desenvolver uma aplicação web utilizando React, é possível reutilizar partes de código para desenvolver agora a aplicação móvel e neste caso não é necessário estar a aprender uma

[5] https://reactjs.org/
[6] https://facebook.github.io/react-native/

nova tecnologia para desenvolver a aplicação móvel pois os conceitos já foram aprendidos a quando do desenvolvimento da aplicação web.

2.3.5.2 Angular

Angular[7] é uma framework desenvolvida pela *Google* construída em TypeScript que fornece aos desenvolvedores ferramentas para criar aplicações web.

A versão inicial do Angular, o conhecido AngularJS, apresentava bastantes problemas quando comparado com tecnologias mais recentes como o React, o que levou os desenvolvedores a criarem uma nova versão baseada em componentes, bastante diferente da primeira. Como a segunda versão do Angular não é compatível de todo com a primeira (AngularJS), é necessário reescrever todo o código de AngularJS para as versões mais recentes, apresentando assim uma grande desvantagem para quem já tinha aplicações web desenvolvidas em AngularJS. O que já não acontece hoje em dia pois os desenvolvedores comprometeram-se a fazer com que a mudança de versões futuras fosse feita de forma direta.

Angular apresenta uma documentação bastante detalhada onde os desenvolvedores podem encontrar todas as informações necessárias para poderem começar a desenvolver uma aplicação web de raiz e rapidamente resolverem problemas que possam ocorrer.

Existe uma grande comunidade a desenvolver aplicações com Angular e desta forma foram criadas bastantes ferramentas adicionais que ajudam no processo de desenvolvimento das aplicações.

Como referido na introdução a esta framework, a mesma é baseada em componentes e funciona de forma semelhante ao referido sobre React. A aplicação é dividida em componentes que são independentes umas das outras cada uma com a sua lógica associada, mas podem comunicar entre si. As componentes podem ser reutilizadas em vários pontos da aplicação e para alem disso facilita nos testes à aplicação, uma vez que se podem testar individualmente.

O compilador de Angular converte o TypeScript e HTML em JavaScript durante o seu processo de criação. Isto significa que o código é compilado antes de o browser carregar a aplicação, fazendo com que a mesma renderize muito mais rápido. Outra característica que possibilita níveis de carregamento mais rápidos é a *Ivy Renderer*. Esta converte as componentes da aplicação em JavaScript e remove o código que não está a ser utilizado durante a renderização, tornando-o mais compacto e por sua vez as aplicações carregam mais rapidamente.

Para a maioria dos desenvolvedores, Angular é bastante complexo e por isso pode ser difícil para quem o começa a aprender. Mesmo com uma arquitetura simples baseada em componentes que torna o código mais legível, é difícil gerir as dependências entre componentes. O facto de também ser desenvolvido em TypeScript faz com que se perca mais tempo no desenvolvimento pois é necessário identificar o tipo de todas as variáveis.

[7] https://angular.io/

Mas pode ser visto como um fator positivo pois estando o TypeScript bem implementado reduz o aparecimento de erros.

Tal como o React fornece uma ferramenta para criar aplicações moveis, o Angular não fornece uma ferramenta desenvolvida pelos seus criadores, mas também é possível fazer o mesmo que o React Native faz, utilizando o Ionic[8] onde é reaproveitado algum do código Angular para desenvolver uma aplicação móvel tanto para Android como iOS.

2.4 Reconhecimento ótico de caracteres

Optical Character Recognition, em português Reconhecimento Ótico de Caracteres (OCR) é uma tecnologia que reconhece texto por meio de uma imagem digital. Para efetuar essa tarefa o software OCR processa uma imagem com recurso à inteligência artificial e daí extrai as letras, números e símbolos reconhecidos nessa imagem.

Embora inicialmente a tecnologia OCR tenha sido idealizada para reconhecer texto impresso, hoje em dia também pode ser utilizada para reconhecer texto manuscrito. Por exemplo uma faculdade poderia utilizar um software OCR para realizar o processamento automático dos nomes dos alunos que realizaram um determinado exame.

De um modo geral existem duas maneiras diferentes de implementar o OCR: Reconhecendo os caracteres na sua totalidade (reconhecimento de padrões) ou detetando as linhas criadas pelos caracteres (deteção de componentes) [21].

2.4.1 Reconhecimento de padrões

Se todas as letras tivessem a mesma fonte seria fácil realizar o reconhecimento de padrões, uma vez que estariam armazenadas todas as letras que existem e assim que algum caractere correspondesse à letra armazenada, seria facilmente identificado [21].

Como esse caso é difícil de acontecer, uma alternativa é armazenar um conjunto de imagens de várias fontes diferentes. Assim o software de reconhecimento tem mais exemplos por onde se guiar e, por sua vez, analisará as várias alternativas que existem para a escrita de determinada letra e mais facilmente identificar os caracteres.

2.4.2 Deteção de componentes

Esta é uma maneira muito mais sofisticada de reconhecimento de caracteres. O OCR com esta abordagem ao invés de reconhecer um padrão completo de determinada letra, vai detetar regras gerais em todos os formatos de letra, independentemente do formato da sua escrita, ou da sua fonte [21]. Analisando o caso da letra "E" maiúscula: apresenta uma linha vertical e três linhas horizontais paralelas entre si. Se for verificada a existência desta regra é possível identificar facilmente a letra "E". Agora basta realizar regras para todas as letras, números e símbolos e tem-se um software de OCR a funcionar.

[8] https://ionicframework.com/

2.4.3 Cloud Vision

Na maioria das vezes a construção de algoritmos de inteligência artificial não é fácil de implementar, tratando-se de uma tarefa demorada e bastante especializada.

Com forma de combater essas dificuldades a Cloud Vision API[9] da *Google* permite que os desenvolvedores menos especializados nessa área tenham os recursos necessários para reconhecer e classificar imagens na sua aplicação, sendo apenas necessário efetuar um simples pedido a uma API REST.

Ao enviar uma imagem à API da Cloud Vision, os serviços da mesma ficam responsáveis por todo o processo de classificação da imagem com a aplicação de inteligência artificial e por fim será recebida uma resposta com os metadados associados à imagem em específico.

A Cloud Vision API fornece várias funcionalidades, entre elas a deteção facial indo até ao pormenor de nos indicar onde se encontram os vários órgãos da face como os olhos, a boca e o nariz. É igualmente possível a identificação de objetos dominantes numa imagem e até mesmo detetar algum logótipo de uma marca conhecida. A funcionalidade mais importante neste projeto é o OCR e esta suporta uma ampla variedade de idiomas.

Outro benefício da sua utilização é o facto de ser um serviço alojado na cloud fazendo com que dispositivos de baixa potência não tenham de suportar o algoritmo pesado de aprendizagem automática, bastando simplesmente efetuar um pedido à API para tirar proveito desse serviço.

2.5 Abordagens relacionadas

Nas primeiras semanas de trabalho na ARTSOFT, no sentido de me familiarizar mais com a proposta apresentada, realizei em conjunto com o gestor de produto da empresa uma análise de domínio onde exploramos outras aplicações que têm funcionalidades semelhantes às que irei desenvolver. Essa análise consistiu em fazer um estudo comparativo das tecnologias existentes no qual fiquei com uma noção do que já existe. As funcionalidades presentes nas plataformas são anunciadas de seguida e esquematizadas na tabela 2.1 com a correspondência para cada plataforma.

- Adicionar despesas manualmente
- Adicionar despesa com fotografia
- Adicionar despesa com fotografia e OCR
- Despesas por tempo - Ajudas de Custo
- Despesas por distância
- Despesas por distância com ligação ao Google Maps

[9] https://cloud.google.com/vision

- Despesas por distância onde faz o track por GPS
- Adicionar várias despesas em simultâneo
- *Merge* de duas ou mais despesas numa só, quando nos enganamos e apenas queríamos fazer uma
- Split de despesas, em várias
- Análise/Dashboard de despesas
- Introduzir e enviar várias despesas de um tipo específico
- Enviar despesas para aprovação
- Adicionar conta do banco para receber o valor das despesas via app
- Adicionar cartão de crédito (introduzir despesas a partir do extrato bancário) - Para Portugal não funciona
- Pagar as despesas recebidas
- Criar uma política de despesas para a empresa
- Sistema de Aprovação de despesas
- Gestão de acessos por utilizador/Perfil
- Integração com várias plataformas
- Gestão de adiantamentos (dinheiro ou transferência bancária ou cheque)
- Controlo dos movimentos dos cartões de crédito da empresa que o colaborador tem na sua posse (ligação direta ao banco)
- Sistema de notificações (notifica o colaborador do estado da despesa)
- Despesas por Centro de custo
- Integração das despesas na contabilidade
- Emitir um recibo da despesa (backoffice)
- Consulta do recibo da despesa na app

As plataformas Expensify[10], Rydoo[11] e Zoho[12] são plataformas estrangeiras para gestão de despesas. A plataforma mais conhecida será a Expensify por ser das mais completas

[10]https://www.expensify.com/
[11]https://www.rydoo.com/
[12]https://www.zoho.com/expense/

Tabela 2.1: Tabela de Análise de Domínio

Funcionalidades	*Expensify*	*Rydoo*	*Zoho*	*Expense Management Primavera*	*Portal Colaborador PHC*
Adicionar despesas manualmente	x	x	x	x	x
Adicionar despesa com fotografia	x	x	x	x	x
Adicionar despesa com fotografia e OCR	x	x	x		
Despesas por tempo - Ajudas de Custo	x				x
Despesas por distância	x	x	x		
Despesas por distância com ligação ao Google Maps	x		x		
Despesas por distância onde faz o track por gps	x		x		
Adicionar várias despesas em simultâneo	x	x	x		x
Merge de duas ou mais despesas numa só, quando nos enganamos e apenas queríamos fazer uma	x	x	x		x
Split de despesas, em várias	x				x
Análise/Dashboard de despesas	x	x	x	x	x
Introduzir e enviar várias despesas de um tipo específico	x	x	x	x	
Enviar despesas para aprovação	x	x	x	x	x
Adicionar conta do banco para receber o valor das despesas via app	x				
Adicionar cartão de crédito (introduzir despesas a partir do extrato bancário) - Para Portugal não funciona	x				
Pagar as despesas recebidas	x	x	x	x	x
Criar uma política de despesas para a empresa	x	x	x	x	x
Sistema de Aprovação de despesas	x	x	x	x	x
Gestão de acessos por utilizador/Perfil	x	x	x	x	x
Integração com várias plataformas	x	x	x		
Gestão de adiantamentos (dinheiro ou transferência bancária ou cheque)			x	x	
Controlo dos movimentos dos cartões de crédito da empresa que o colaborador tem na sua posse (ligação direta ao banco)				x	
Sistema de notificações (notifica o colaborador do estado da despesa)	x	x	x	x	x
Despesas por Centro de custo				x	x
Integração das despesas na contabilidade			x	x	x
Emitir um recibo da despesa (backoffice)	x	x	x	x	x
Consulta do recibo da despesa na app	x	x	x		x

do mercado, oferecendo quase na integridade todas as funcionalidades encontradas. Criada em 2008, a plataforma pode tanto ser utilizada para uso pessoal como comercial, sendo este último o mais importante pois assemelha-se com a plataforma que se pretende desenvolver. A Rydoo é uma plataforma bastante focada em empresas onde os seus funcionários realizam bastantes viagens de motivo comercial, fornecendo até um serviço de ajuda à reserva de hóteis e voos para colmatar as necessidades da viagem. Já a Zoho é um Customer Relationship Management, em português Gestão de Relacionamento com o Cliente (CRM) que fornece uma extensa quantidade de aplicações para administração das interações das empresas com os clientes; dentro da sua oferta de aplicações, existe uma que faz a gestão das despesas das empresas que adquirirem o serviço.

A Expense Management Primavera[13] e o Portal Colaborador PHC[14] são concorrentes portugueses da ARTSOFT, em que o seu foco é desenvolver software de apoio à gestão empresarial. Assim sendo, foi analisado quais as funcionalidades que as suas plataformas apresentam para a gestão das despesas de uma determinada empresa. Como observado na tabela o Portal Colaborador PHC é aquele que apresenta um número superior de funcionalidades.

[13] https://pt.primaverabss.com/pt/software/solucoes-especializadas/gestao-de-despesas/
[14] https://phcsoftware.com/

3

Conceção da Solução

3.1 Requisitos da plataforma

Depois de ter sido feito um estudo de algumas plataformas semelhantes à qual se pretende desenvolver foram definidos os requisitos que se pretendem alcançar nesta plataforma.

3.1.1 Requisitos funcionais

Entendem-se por requisitos funcionais o conjunto de funcionalidades a serem implementadas na plataforma. No trabalho desenvolvido durante a dissertação foram definidos os seguintes:

Criação de despesas Os utilizadores têm a capacidade de registam as suas despesas, preenchendo todos os campos necessários com as informações das mesmas. É possível registar despesas normais, ou despesas por distância.

Aprovação ou reprovação de despesas Relativamente aos utilizadores que sejam coordenadores de um grupo ou superiores hierárquicos de outros utilizadores, deve ser possível que estes possam pesquisar por despesas que têm para aprovar. Depois de verificarem os campos das despesas, têm o poder de as aprovar caso se encontre tudo corretamente preenchido. Por sua vez, caso encontrem algo que não considerem correto, têm a possibilidade de a reprovar.

Visualização das despesas O utilizador é capaz de visualizar o registo das suas despesas, com todos os seus detalhes bem como o estado das mesmas, sabendo se estas já estão aprovadas ou não. Existem utilizadores com permissões para visualizar as despesas efetuadas por outros, caso exista a necessidade de as aprovar.

Criação de empresas Esta plataforma suporta várias empresas e, consequentemente, um utilizador tem a possibilidade de colaborar em diversas. Desta forma, consegue criar várias empresas e colaborar em empresas criadas por outros utilizadores.

Convidar outros utilizadores para as empresas criadas Tendo um utilizador criado uma empresa, pode convidar outros utilizadores para colaborarem nessa mesma empresa criada. Os convites podem ser efetuados tanto a utilizadores já registados na plataforma, como a utilizadores não registados. Sendo que estes últimos quando receberem o convite têm de efetuar o registo na plataforma.

Registo de utilizadores A plataforma suporta também a gestão dos vários utilizadores, necessitando previamente do registo dos mesmos na plataforma.

Criação de grupos dentro das várias empresas Dentro de uma empresa existe a possibilidade de se criarem vários grupos onde se podem integrar os diversos recursos da empresa. Esses grupos terão coordenadores que serão os responsáveis dos mesmos e ocupar-se-ão de aprovar as despesas dos utilizadores que os integram.

Sistema hierárquico dentro das empresas Para ser possível efetuar o resgisto de aprovação ou reprovação de despesas é necessário que existam utilizadores que possam proceder ao mesmo, assim sendo, a plataforma suporta um sistema hierárquico dentro de cada empresa. O criador da empresa é o administrador e pode converter os diversos utilizadores da sua empresa a coordenadores. O administrador tem também a possibilidade de fazer com que certo utilizador possa aprovar as despesas de um outro sem que o primeiro seja coordenador.

3.1.2 Requisitos não funcionais

Os requisitos não funcionais atendem às características do sistema a ser desenvolvido e desta forma descrevem como o utilizador se relaciona com o mesmo. Foram então idealizados os seguintes requisitos não funcionais:

Portabilidade A plataforma permite uma independência de SGBD, sendo relativamente simples a mudança do mesmo apenas com a troca da classe que efetua as *queries* sobre os dados.

Reusabilidade Como referido na introdução desta dissertação o foco desta plataforma passa também por criar componentes e exemplos de arquiteturas que possam posteriormente ser reutilizáveis para uma nova reestruturação dos servidores da empresa ARTSOFT.

Segurança Devem ser garantidos os requisitos de segurança de autenticação dos utilizadores, bem como as permissões dos mesmos relativamente ao acesso ao sistema. Desta forma a plataforma só deve responder a utilizadores autenticados e autorizados que pretendam aceder a determinadas informações.

Escalabilidade A plataforma deve possibilitar que no futuro seja possível adicionarem-se mais funcionalidades à mesma, sem que para isso exista algum tipo de mudança nas funcionalidades anteriormente implementadas.

Persistência A persistência dos dados da plataforma são garantidos pelo SGBD que estiver a ser utilizada no sistema. Sendo esse o responsável por guardar todos os dados da plataforma necessários para o seu correto funcionamento.

3.2 Tecnologias

Como abordado na secção 2.1 desta dissertação, será implementado um serviço REST devido à sua simplicidade e ao facto de ser o modelo mais comum hoje em dia com uma grande diversidade de ferramentas que tornam a sua implementação mais simples.

3.2.1 Spring Boot

Para a sua implementação será utilizada a framework Spring Boot (2.1.4.1) pela facilidade que esta oferece no seu processo de desenvolvimento, sem que haja a necessidade de grandes configurações. Tendo em conta a implementação de um serviço seguro, o Spring é das melhores frameworks para a realização do mesmo. Possui também uma excelente documentação e uma grande comunidade de desenvolvedores.

3.2.2 MySql

Relativamente ao SGBD a utilizar foi definido no início do projeto que este deveria suportar uma abstração nesse aspeto e permitir a integração de diversos SGBDs. Para a implementação inicial será utilizado MySQL (2.2.2.1) por ser um sistema *open source* com o qual eu já trabalhei e estou acostumado.

3.2.3 Angular

A respeito da interface gráfica do utilizador será utilizado Angular (2.3.5.2) para a sua implementação. A sua escolha deve-se ao facto de ser uma framework sólida que facilita bastante a implementação do *front-end*, e destaca-se por impor uma estrutura específica ao projeto obrigando todos os desenvolvedores a segui-la. Apesar da sua aprendizagem demorar algum tempo, a longo prazo essa abordagem é benéfica aos desenvolvedores pois é uma estrutura reconhecida por todos e a sua manutenção torna-se fácil. Outro fator que me leva à sua escolha deve-se a já ter desenvolvido projetos onde utilizei a primeira versão do Angular o AngularJS. Apesar de esta divergir das versões mais atuais, a sua permuta não é algo muito complexo.

3.3 Modelações da plataforma

Como referido na secção anterior para a implementação do servidor será utilizado o Spring. Tendo em conta a estrutura recomendada pela documentação, na figura 3.1 está apresentada a abordagem pensada para a implementação desse serviço.

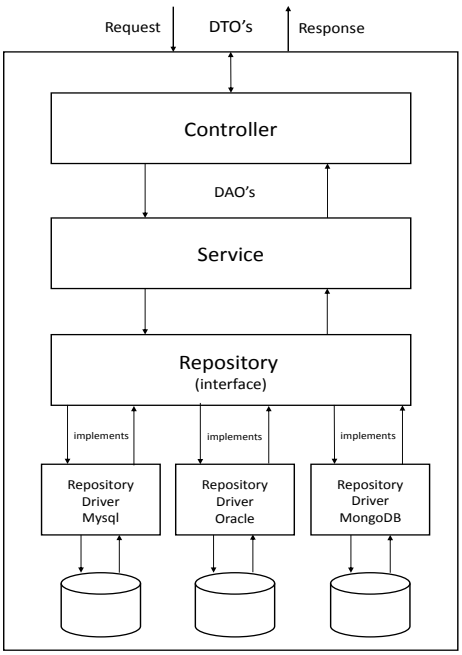

Figura 3.1: Modelação do Servidor de Recursos

Começando a analisar o esquema do topo para baixo, sendo este um servidor REST, a comunicação com o exterior é efetuada por mensagens HTTP e os recursos de entrada e saída são representados através de Data Transfer Object, em português Objeto de Transferência de Dados (DTO)s. Com a utilização dos DTOs o cliente não tem conhecimento de como está organizado o serviço e a única forma de comunicação com o mesmo é através de pedidos HTTP onde pode enviar e receber DTOs.

Enquanto o cliente gere os recursos do serviço através de DTOs dentro do serviço, é necessário a existência de um objeto que apresente mais dados do que aqueles que são apresentados ao cliente. Por exemplo, suponhamos que a nossa aplicação quer apresentar todos os utilizadores da plataforma, quando estes são exibidos ao cliente apenas podem conter alguns detalhes do mesmo, o serviço não poderá devolver, por exemplo, as palavras passe dos utilizadores.

Para efetuar essa filtragem os recursos são guardados dentro da base de dados como Data Access Object, em português Objeto de Acesso de Dados (DAO) e toda a lógica efetuada dentro do serviço é feita com DAOs. Mas para a exibição do recurso ao cliente é enviado um DTO.

O Controller é o elemento responsável por comunicar com o cliente. É o único que conhece tanto os DTOs e os DAOs e faz o *mapping* entre ambos para os enviar para o

Service ou para o lado do cliente.

Service é a componente responsável por lidar com a parte lógica do serviço. Faz todas as verificações necessárias e devolve erros, caso estes existam.

A interface Repository apresenta todas as operações de gestão dos recursos na base de dados (inserção, eliminação, atualização) e contem os métodos que o Service pode utilizar para realizar operações sobre a base de dados. Devido ao facto de o nosso serviço apresentar uma independência do SGBD é necessária a criação desta interface para evitar repetição de código sempre que adicionássemos um *driver* novo, assim todos os métodos estão aqui definidos e cabe a cada *driver* implementá-los.

Esses *drivers* são aqueles que comunicam com os SGBDs correspondentes. Nesse caso terão de implementar os métodos definidos na interface Repository na sua linguagem específica. Por exemplo para o MySQL a escrita das *queries* será diferente da utilização de MongoDB, devido a esse facto é necessário que cada SGBD tenha o seu *driver* específico.

Fica agora a faltar abordar as questões de segurança, uma vez que a modelação explicada anteriormente ainda não preenche esses requisitos. Para isso foi projetada uma modelação que consiga lidar com a autenticação dos utilizadores.

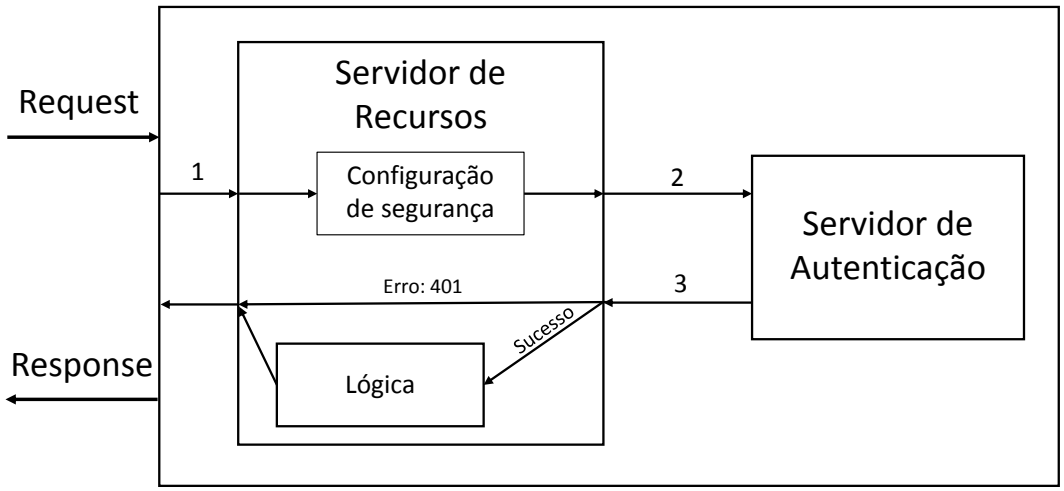

Figura 3.2: Modelação do Serviço de Autenticação

Na figura 3.2 é apresentada a arquitetura que visa solucionar a questão da segurança. Esta abordagem vem de encontro ao que é apresentado na secção 2.1.3.3. Optei pela utilização da Grant Credenciais Proprietário do Recurso pois o nosso serviço apenas comunicará com aplicações cliente desenvolvidas por mim, não havendo, desta forma, problema que o proprietário de recurso forneça as suas credenciais ao cliente.

Nesta modelação quando se efetua um pedido ao nosso servidor o mesmo é direcionado para o servidor de recursos, mas antes deste efetuar qualquer operação do foro

lógico irá enviar um pedido ao servidor de autenticação para validar o utilizador. Suponhamos que o cliente efetua um pedido ao servidor, esse pedido tem de conter um *header* com o *token* de acesso. O servidor de recursos recebe esse pedido (representado na figura 3.2 com o número 1) e envia o *token* ao servidor de autenticação que será responsável por verificar se o mesmo se encontra válido (número 2). O servidor de autenticação devolve uma resposta de sucesso ou de erro novamente ao servidor de recursos (número 3). Caso a resposta devolvida seja de sucesso, então o servidor de recurso cuidará de toda a parte lógica do pedido. Caso contrário, devolve uma mensagem de erro (401 Unauthorized) ao cliente.

Para efetuar o login dos utilizadores utilizaremos uma técnica do OAuth 2 explicada no capítulo 2 em 2.1.3.3.

A arquitetura do servidor de recursos corresponde à apresentada anteriormente, observada na figura 3.1.

Separando, desta forma, o servidor de autenticação do servidor de recursos, podemos colocá-los em máquinas diferentes e fornecer mais poder computacional a algum dos servidores que observemos que está a ficar sobrecarregado. Outra vantagem é o facto de serem independentes um do outro e com isso, caso se faça uma alteração no servidor de recursos, apenas será necessário fazer *deploy* nesse, permanecendo o outro igual.

4

Implementação

4.1 Base de Dados

Cada um dos servidores terá a sua base de dados correspondente. É possível a existência de algumas abordagens onde todos os servidores vão consultar a mesma base de dados, mas isso não é uma solução aconselhada, pois a maioria dos servidores será hospedado em máquinas diferentes.

Podemos observar pela figura 4.1 o modelo Entity Relationship, em português Entidade Relação (ER) desenhado para a base de dados do servidor de recursos. Esta modelação corresponde ao servidor de recursos e como observamos as tabelas são responsáveis por gerir as informações dos colaboradores, as suas hierarquias, as despesas associadas aos recursos.

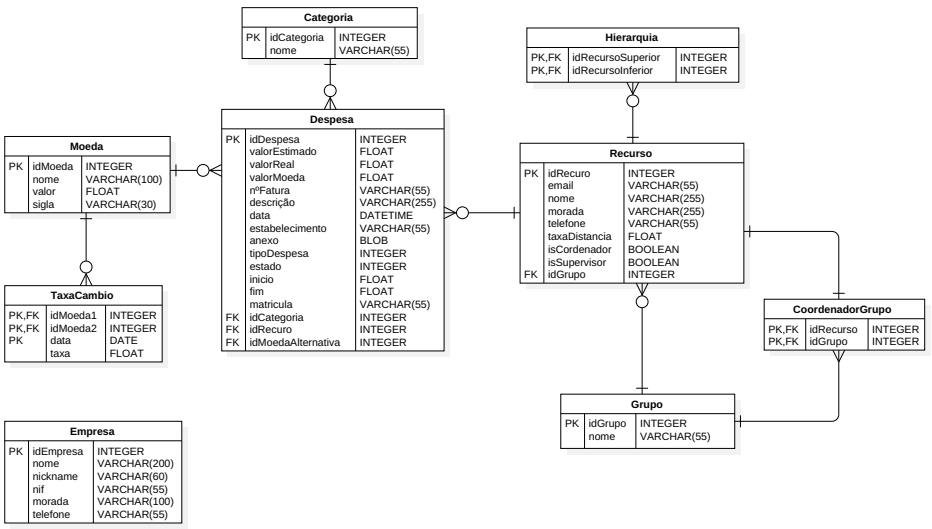

Figura 4.1: Modelo ER do Servidor de Recursos

Como mencionado no capítulo anterior, é determinado que o servidor de recursos permita a independência do SGBD, mas numa primeira experiência será utilizado o MySQL. Para que a aplicação Spring possa aceder à base de dados é necessário proceder a algumas

configurações nas propriedades da aplicação, adicionando as referências para a localização e autenticação da base de dados. De modo a ser possível configurar a aplicação de uma forma abstrata ao local onde se encontra implementada, as informações referentes à base de dados são guardadas em variáveis de ambiente que serão guardadas no sistema que estiver a executar o servidor. Deste modo as credenciais ficam escondidas e, caso o servidor esteja a ser executado em duas máquinas diferentes, a localização da base de dados poderá ser diferente, assim como as credenciais de acesso às mesmas. É necessário também adicionar as dependências Maven necessárias para se poder operar com as bases de dados MySQL.

Analisando de forma detalhada a modelação do modelo ER do servidor de recursos, será de seguida explicada cada uma das suas tabelas e as suas relações.

A tabela "Empresa" guarda toda a informação referente à empresa a que esta base de dados se refere. Estes dados são apenas informativos não sendo utilizados nenhum outro motivo a não ser a exposição.

A tabela "Moeda" armazena a informação sobre as várias moedas existentes, enquanto que a tabela "TaxaCambio" diz respeito ao valor de uma dessas moedas apresentadas na primeira tabela medido em relação a uma outra. Esta tabela tem de apresentar a relação entre todas as moedas existentes, e visto que os câmbios entre moedas têm uma variabilidade bastante elevada ao longo do tempo, tem de ser guardada a data desta relação. Apesar de serem apresentas estas tabelas, no trabalho desenvolvido não estão a ser utilizadas, uma vez que se optou primeiramente por uma abordagem mais simples, onde apenas se faz o registo de despesas com uma única moeda. Ainda assim encontram-se já modeladas no sistema para no futuro ser possível a sua utilização.

No que diz respeito às tabelas relacionadas com os recursos, é na tabela "Recurso" que estão guardados todos os detalhes de informações referentes aos mesmos, como email, nome, morada e telefone. O campo "taxaDistancia" é referente ao valor que o recurso especifico irá receber por cada quilometro efetuado em despesas por distância e o mesmo pode ser alterado pelo administrador da empresa. Já o campo "isCordenador" representa a *flag booleana* que indica se este recurso é ou não um coordenador na dada empresa. O campo "idGrupo" é um chave estrangeira da relação da tabela "Recurso" com a tabela "Grupo", onde um recurso apenas pode pertencer a um grupo e um grupo pode conter vários recursos. Ainda sobre a relação entre os recursos e os grupos, estes podem conter vários coordenadores e deste modo a tabela "CoordenadorGrupo" guarda os recursos que são coordenadores de determinado grupo.

De forma a guardar as informações sobre as ordens hierárquicas entre os recursos, existe a tabela "Hierarquia" com duas chaves estrangeiras dos *id*'s dos recursos correspondentes ao superior e inferior na hierarquia.

Por fim, a tabela "Despesa" contém todos os dados referentes às varias despesas efetuadas na respetiva empresa. São apresentados três tipos de valores - "valorEstimado", "valorReal" e "valorMoeda" no caso de serem apresentadas despesas com moedas diferentes e os mesmos são calculados pelas taxas de câmbio, mas como nesta primeira fase não

foi implementada essa funcionalidade será apenas dada importância ao "valorEstimado". Como foi anteriormente identificado, existem vários tipos de despesas, consequentemente esta tabela tem um campo ("tipoDespesa") com a *flag* correspondente ao tipo da despesa em questão: valor "1" se for uma despesa simples e "2" se for uma despesa de distância. Se for uma despesa simples os campos "início", "fim" e "matrícula" estarão como nulos na tabela, uma vez que não fazem sentido para a despesa em questão. Caso seja uma despesa por distância os campos "estabelecimento" e "nºFatura" serão desnecessários. A distância efetuada nessa despesa será calculada subtraindo o valor de "fim" ao valor de "início" e é calculado o valor da despesa multiplicando o número de quilómetros efetuados com o campo "taxaDistancia" do recurso que realizou a despesa.

Caso se pretenda anexar uma fotografia da fatura da despesa, o campo "anexo" guarda uma *string* em base64 referente a essa imagem. Relativamente às chaves estrangeiras o campo "idCategoria" diz respeito à categoria escolhida pelo recurso para representar a sua despesa, selecionando uma das que já se encontram enumeradas na tabela "Categoria", o "idRecurso" corresponde ao recurso que efetuou a despesa e por fim o "idMoedaAlternativa" não está a ser utilizado neste momento pois não foi implementado o sistema de introdução de despesas com diferentes tipos de moedas.

Na figura 4.2 temos representado o modelo ER da base de dados do servidor de autenticação que guarda informações de autenticação e as permissões dos utilizadores do sistema.

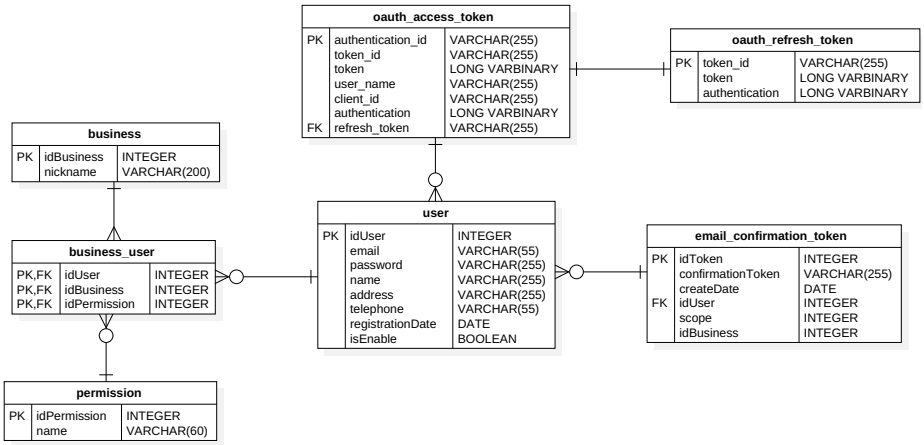

Figura 4.2: Modelo ER do Servidor de Autenticação

Como será explicado na secção 4.3 o servidor de autenticação foi criado com auxílio de funcionalidades do Spring que geram de forma automática *endpoints* para facilitar na implementação do mesmo. Como tal é necessário que a base de dados inclua tabelas obrigatórias ao seu funcionamento. São essas tabelas a "oauth_access_token", responsável por armazenar os *tokens* de acesso dos vários utilizadores do sistema, como o seu tempo de vida, e a "oauth_refresh_token", referente aos *tokens* de atualização, quando é necessário atualizar algum *token* de acesso devido ao seu tempo de vida já ter expirado.

A tabela "user" guarda todas as informações dos utilizadores registados no sistema tal como o seu email, password devidamente encriptada e detalhes do mesmo. De realçar que o campo "isEnable" é um booleano que indica se o utilizador está ativo ou não, e caso tenha o valor falso o utilizador não poderá efetuar o login.

As tabelas responsáveis pelas permissões dos utilizadores são as "business" que contêm todas as empresas criadas no sistema e o seu nickname, a "business_user" que estabelece a relação entre os utilizadores registados e as empresas criadas, juntamente com qual a sua permissão dentro da empresa respetiva, estando estas permissões guardadas na tabela "permission". Neste momento apenas existem na plataforma dois tipos de permissões, o **ADMIN**, utilizador responsável pela administração da empresa, ou o **USER** que se refere a um utilizador simples dentro da empresa.

Por último a tabela "email_confirmation_token" é responsável por guardar os *tokens* enviados para o email com um *link* de confirmação de acordo com um determinado "scope". O campo "scope" é um inteiro que abrange três hipóteses distintas, tem o valor 1 caso o *token* dessa linha da tabela seja referente à criação de um utilizador servindo, desta forma, para confirmação do registo do utilizador. Se tiver o valor 2 corresponde a um *token* de convite de um utilizador para a determinada empresa com o id contido em "idBusiness". Finalmente, pode ainda ter o valor 3 referente a um convite de um utilizador para uma empresa, mas este não se encontra registado no sistema, portanto caso aceite o convite para se juntar à empresa tem também de proceder ao seu registo.

4.2 Servidor de Recursos

Nesta secção é apresentado com o auxílio de diagramas de sequências todo o funcionamento do servidor de recursos. Para todos os *endpoints* da API é descrita a sua implementação revelando os seus passos e as classes que os executam. De realçar que as verificações relacionadas com a autorização e autenticação não estão directamente explícitas nos diagramas de sequências, mas serão abordadas posteriormente nas secções 4.2.7 e 4.2.8.

4.2.1 Gestão do Utilizador

Neste ponto estão presentes os *endpoints* relacionados com a gestão de autenticação dos utilizadores.

4.2.1.1 Login

Endpoint responsável pelo login dos utilizadores. Observando a figura 4.3 o servidor recebe as credenciais do utilizador (email e password) e envia as mesmas para o servidor de autenticação de modo a que este as possa validar. Caso estejam corretas, será enviada uma resposta com os *tokens* de acesso de modo a que o utilizador os possa utilizar para

poder realizar mais pedidos. Se as credenciais enviadas para o servidor de autenticação estiverem incorrectas, o mesmo devolve uma resposta a enunciar o erro, que será devolvida também como resposta final (400 "Email or Password are incorrect").

Quando recebidos os *tokens* de acesso do servidor de autenticação, procedemos à pesquisa das permissões do utilizador que pretende executar o login que serão enviadas juntamente com esses mesmos *tokens*.

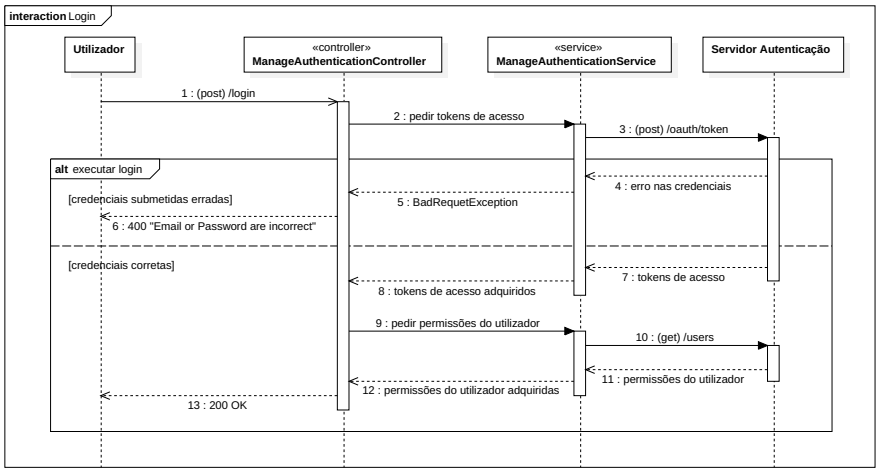

Figura 4.3: Login

4.2.1.2 Logout

O *endpoint* de logout faz exatamente o contrário do login, o qual foi anteriormente explicado. Desta vez como visualizado na figura 4.4 é feito também um pedido ao servidor de autenticação, mas agora é enviado o *token* de acesso do utilizador para o mesmo ser anulado, de modo a que se for feito mais algum pedido com esse *token* o servidor de recursos devolve uma mensagem de erro 401 "Unauthorized" pois o mesmo já não é válido.

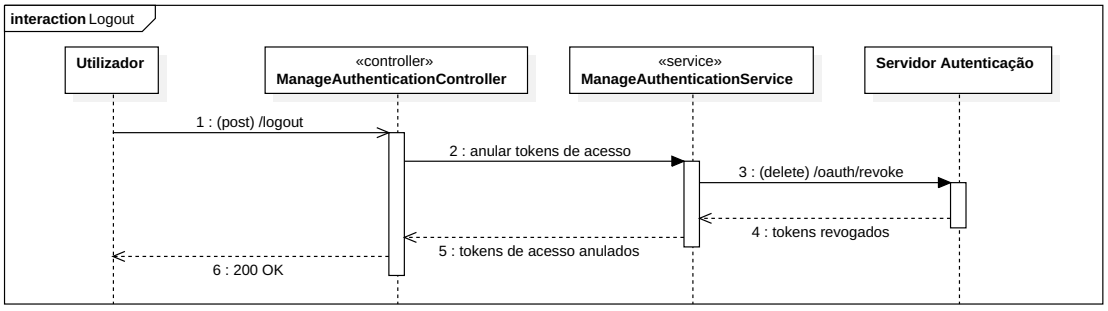

Figura 4.4: Logout

4.2.1.3 Refresh token

O *endpoint* de refresh token observado na figura 4.5 é utilizado quando o *token* de acesso já expirou e se pretende receber um novo para o utilizador poder continuar a realizar pedidos. Este envia o seu refresh token que é posteriormente enviado para o servidor de autenticação, que depois de devidamente verificado devolve como resposta um novo *token* de acesso.

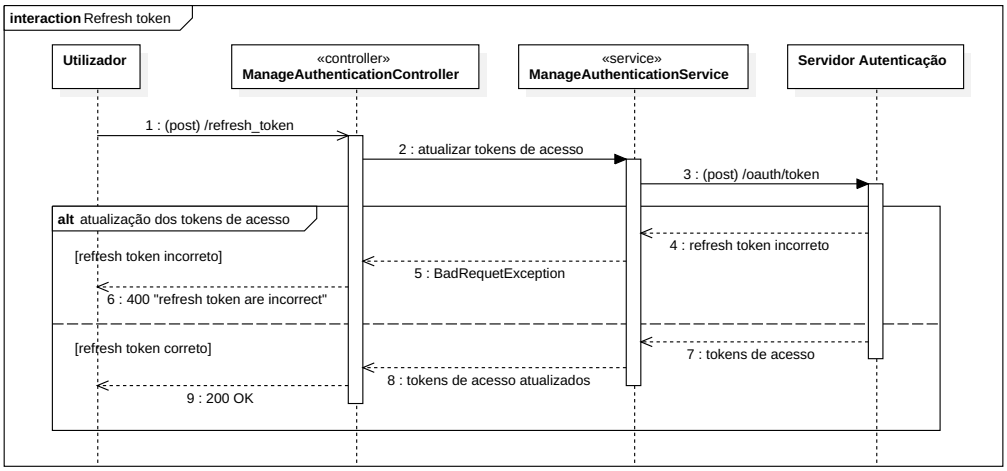

Figura 4.5: Refresh token

4.2.1.4 Signup

Quando se pretende registar um novo utilizador na plataforma utiliza-se o *endpoint* signup, representado na figura 4.6. O utilizador preenche devidamente os campos com as suas informações e envia para o servidor de recursos. Este envia os campos preenchidos anteriormente para o servidor de autenticação que passará a verificar os mesmos.

Caso se verifique algum problema na inserção dos campos o servidor de autenticação envia-nos uma mensagem de erro 400 "Bad Request" com as devidas informações. Outra mensagem de erro que poderá surgir, verifica-se quando o utilizador insere como email algum já existente na plataforma e neste caso será devolvida uma mensagem com o erro 409 "User with email already exists".

Se não for verificado nenhum erro por parte do servidor de autenticação, o mesmo enviará um email para o que foi indicado pelo utilizador, para este proceder à confirmação do registo. Só depois de confirmado o seu registo é que poderá efetuar pedidos à plataforma.

4.2.1.5 Atualizar informações do utilizador

Sempre que um utilizador necessitar de efetuar alterações às suas informações gerais na plataforma e não específicas de determinada empresa, como por exemplo alterar a morada, é utilizado este *endpoint* 4.7.

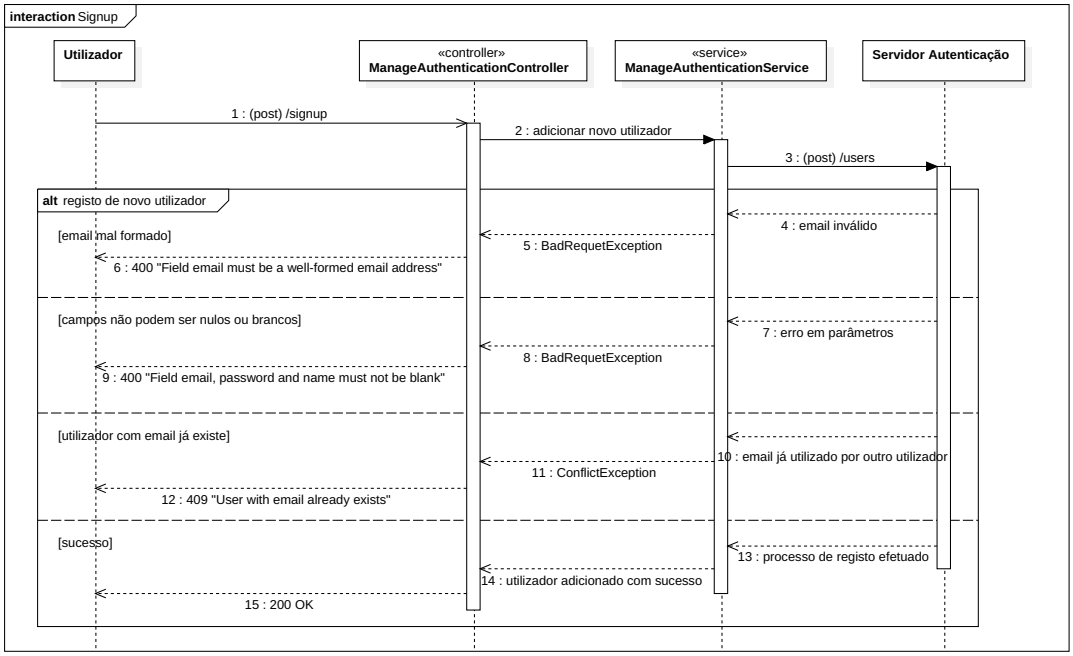

Figura 4.6: Signup

Mais uma vez o servidor de recursos envia um pedido ao servidor de autenticação com as mudanças a efetuar, este verifica se as mesmas são possíveis e executa as alterações pedidas. Depois de confirmar que as alterações ao utilizador foram realizadas e visto que no servidor de recursos também são guardadas algumas informações do utilizado,r procedemos à alteração das mesmas, desta vez na base de dados correspondente ao servidor de recursos.

Este *endpoint* tem como pré-requisito à sua autorização a validação explícita em 4.2.8.5.

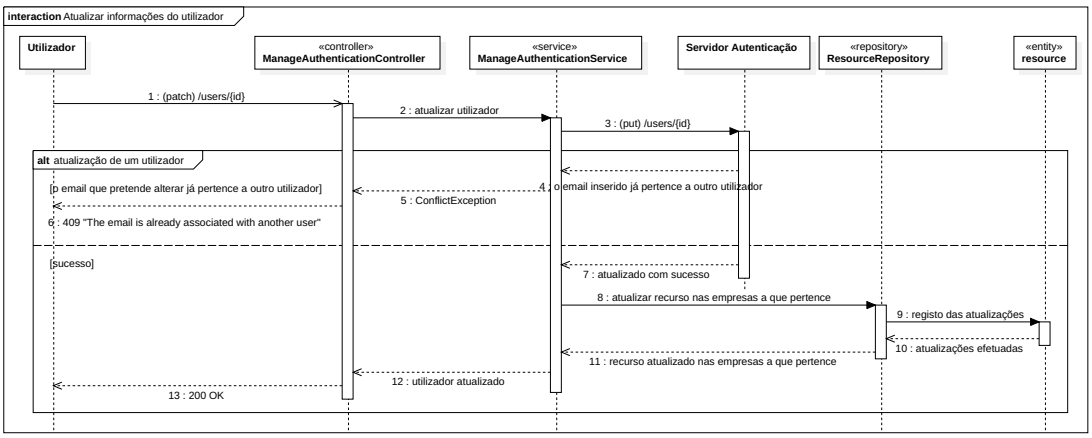

Figura 4.7: Atualizar informações do utilizador

4.2.1.6 Detalhes das permissões do utilizador logado

O *endpoint* observado na figura 4.8 é executado quando pretendemos saber quais as permissões do utilizador logado. Como demonstrado na figura, o utilizador indentifica no *header* o seu *token* de acesso que seguidamente é enviado ao servidor de autenticação que devolve as suas permissões.

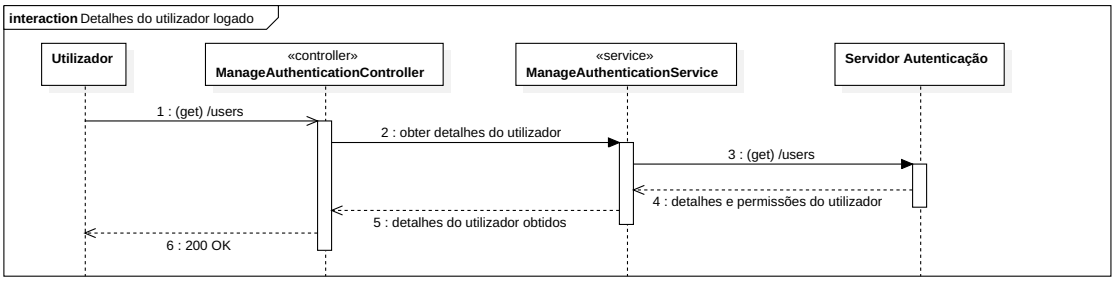

Figura 4.8: Detalhes das permissões do utilizador logado

4.2.2 Empresas

4.2.2.1 Criar empresa

Este é o *endpoint* responsável pela criação de empresas na plataforma (figura 4.9). Visto que cada empresa possui a sua base de dados, a primeira ação a ser realizada é a criação dessa base de dados correspondente a essa empresa. Antes da criação da base de dados, o servidor irá verificar se já existe uma empresa criada com o mesmo *nickname* fornecido e se o mesmo possui espaços em branco ou é nulo. Caso se verifique algum desses erros, o servidor responde com uma das seguintes mensagens respetivamente: 409 "Business with nickname already exists" ou 400 "The business nickname cannot be null or have whitespaces".

Estando criada a base de dados é agora necessário registar a empresa no servidor de autenticação de modo a atribuir as permissões aos utilizadores que futuramente serão inseridos na respetiva empresa. Concluído este passo, serão adicionadas na nova base de dados criada as informações fornecidas correspondentes à empresa e ao recurso que a criou. Terminando assim este *endpoint* e com a possibilidade de o utilizador poder neste momento realizar operações com a nova empresa criada.

4.2.3 Recursos

4.2.3.1 Listar todos os recursos

Endpoint da figura 4.10 que devolve as informações de todos os recursos de uma determinada empresa, os mesmos são pesquisados na empresa indicada.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

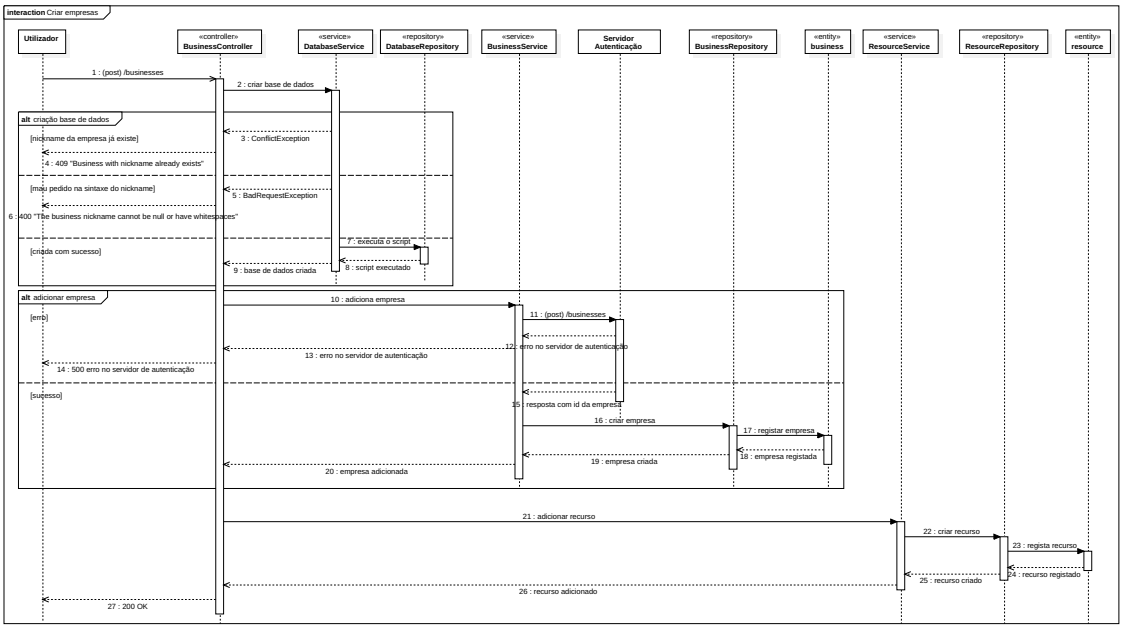

Figura 4.9: Criar empresa

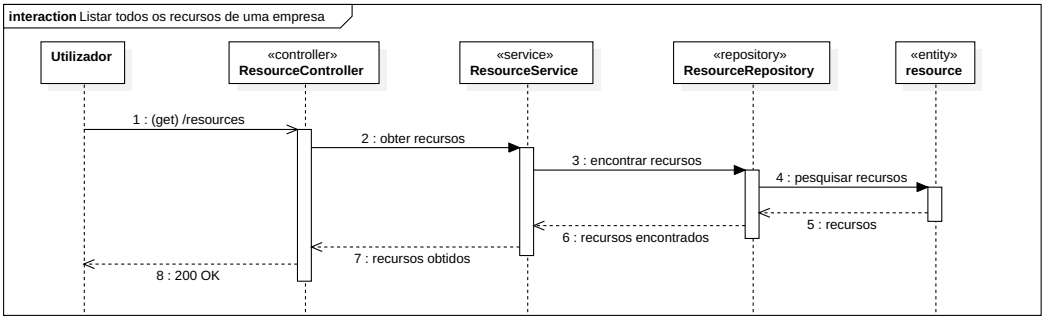

Figura 4.10: Listar todos os recursos

4.2.3.2 Obter recurso

Para se encontrar as informações de um utilizador numa determinada empresa, utiliza-se o *endpoint* descrito na figura 4.11 em que dado o *id* de um respetivo recurso é devolvida a sua informação. Caso a base de dados não possua nenhum recurso com o *id* fornecido é enviada a mensagem de erro 404 "Resource not found".

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.5, ou 4.2.8.2.

4.2.3.3 Convidar recurso para a empresa

Quando se pretende adicionar um recurso a uma empresa efetua-se o cenário descrito na figura 4.12. É enviado um pedido ao servidor de recursos com o email do recurso que se pretende adicionar à empresa e este mesmo email é reencaminhado juntamente com o *id* da respetiva empresa em forma de pedido ao servidor de autenticação. Caso o recurso

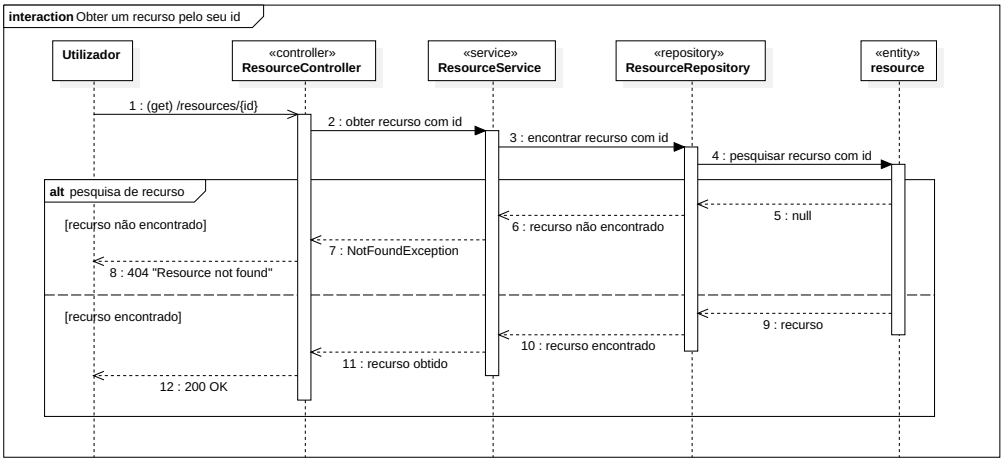

Figura 4.11: Obter recurso

indicado já se encontre inserido na respetiva empresa, será enviada uma mensagem de erro 409 "The resource is already in the business". Em caso de sucesso o servidor de autenticação enviará um email com o convite para ingressar na empresa, o mesmo é explicado em 4.3.2

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

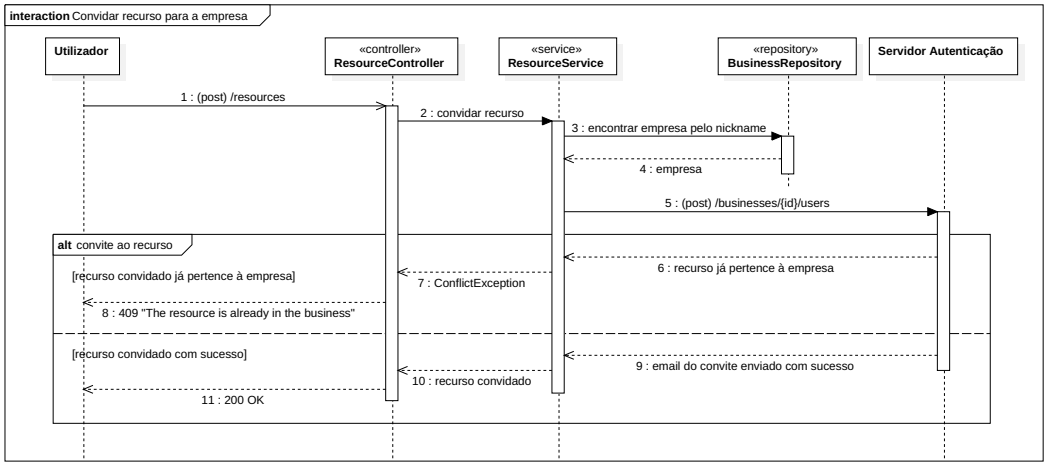

Figura 4.12: Convidar recurso para a empresa

4.2.3.4 Aceitar convite

Depois de ser efetuado o convite para um recurso ingressar numa determinada empresa, o mesmo irá receber um email com um *link* de confirmação para o seu ingresso. Esse mesmo *link* corresponde ao *endpoint* observado na figura 4.13.

Quando é efetuado este pedido é enviado também um *token* e o *nickname* da empresa

para ser possível ao servidor de autenticação associar o convite ao respetivo recurso. Associados a esses *tokens* o servidor de autenticação poderá devolver as seguintes mensagens de erro: 400 "The business nickname does not match to the invite token", caso o *nickname* da empresa não corresponda ao *token* enviado; 404 "The invitation token was not found", caso o *token* enviado não se encontre presente na base de dados. Outra mensagem de erro ocorre quando o respetivo convite já foi aceite, ou seja, o recurso já se encontra na empresa, devolvendo 409 "The user already is in the business".

Por fim, assim que o servidor de autenticação valida o *token* de convite, o recurso correspondente pode ser adicionado à base de dados da empresa.

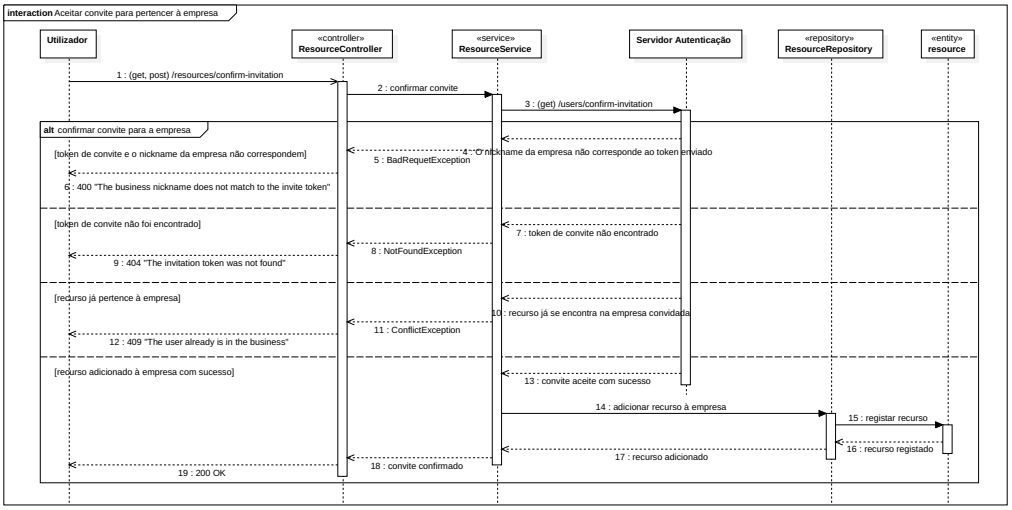

Figura 4.13: Aceitar convite

4.2.3.5 Adicionar recurso a um grupo

O *endpoint* responsável por adicionar recursos a um grupo é descrito pelo diagrama de sequências da figura 4.14.

Primeiramente é verificado se tanto os *id*'s do recurso como do grupo indicados existem na base de dados da respectiva empresa, caso não existam são enviadas as mensagens de erro: 404 "Resource not found"; 404 "Group not found" respetivamente. Como regra desta plataforma um recurso apenas pode estar incluído num grupo, como tal existe uma verificação atestar essa regra e caso o recurso indicado já pertença a um grupo é enviada a mensagem de erro: 409 "Resource already has group". Se não existir erros nessas verificações o recurso é adicionado ao grupo com sucesso.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.7, ou 4.2.8.2.

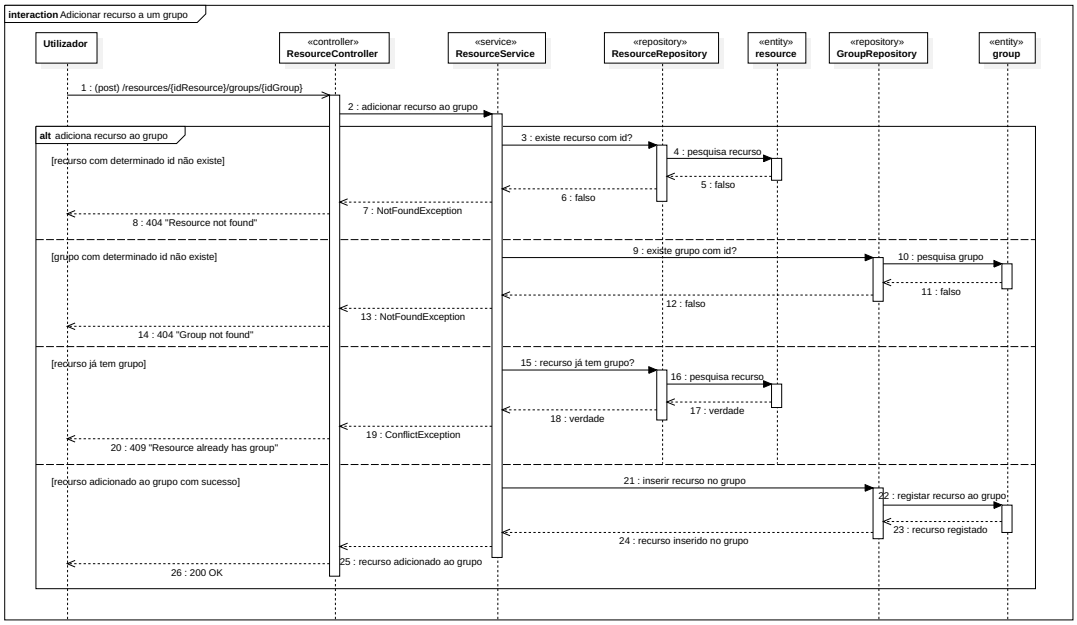

Figura 4.14: Adicionar recurso a um grupo

4.2.3.6 Tornar recurso como coordenador

O administrador da empresa tem a opção de eleger determinados recursos como coordenadores, para poderem passar a coordenar um grupo. O *endpoint* responsável por tornar um recurso como coordenador é representado na figura 4.15. Aqui é verificado se o recurso existe na empresa e se o mesmo já é ou não coordenador. Sendo feitas essas verificações, o recurso é então registado como coordenador.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

4.2.3.7 Listar todos os coordenadores

O *Endpoint* da figura 4.16 devolve os detalhes de todos os recursos que são coordenadores numa determinada empresa.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

4.2.3.8 Adicionar uma hierarquia a um recurso

Neste *endpoint* exposto na figura 4.17 são enviados os *id*'s correspondentes aos recursos a que se pretende estabelecer a hierarquia, os mesmos não podem ser iguais, pois um recurso não pode ser superior hierárquico dele próprio e caso isso seja verificado será enviada a mensagem de erro 400 "The resource superior cannot be the same as resource inferior".

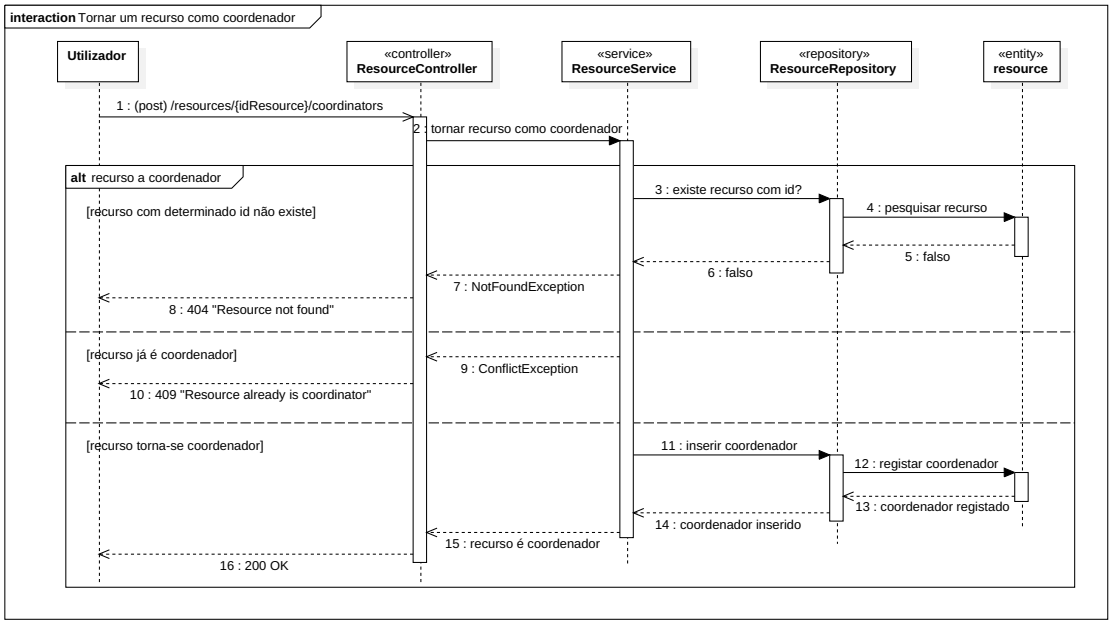

Figura 4.15: Tornar recurso como coordenador

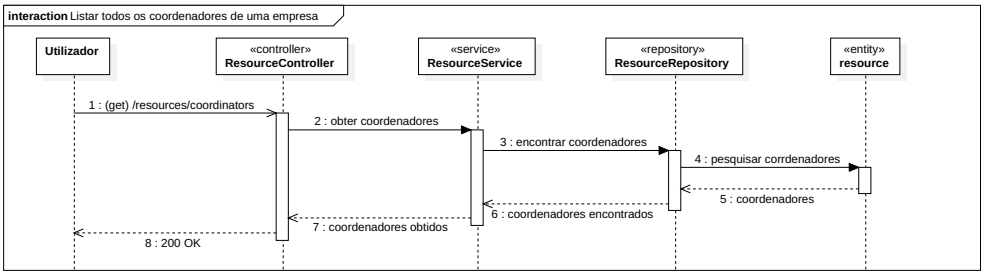

Figura 4.16: Listar todos os coordenadores

Encontrando-se essas verificações todas devidamente acertadas poderá registar-se na base de dados correspondente à empresa em causa a ordem hierárquica indicada.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

4.2.3.9 Lista dos recursos superiores

Para se listar os recursos superiores de um outro recurso, ou seja, os que estão num grau hierárquico superior ao recurso indicado, utiliza-se o *endpoint* exposto na figura 4.18.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

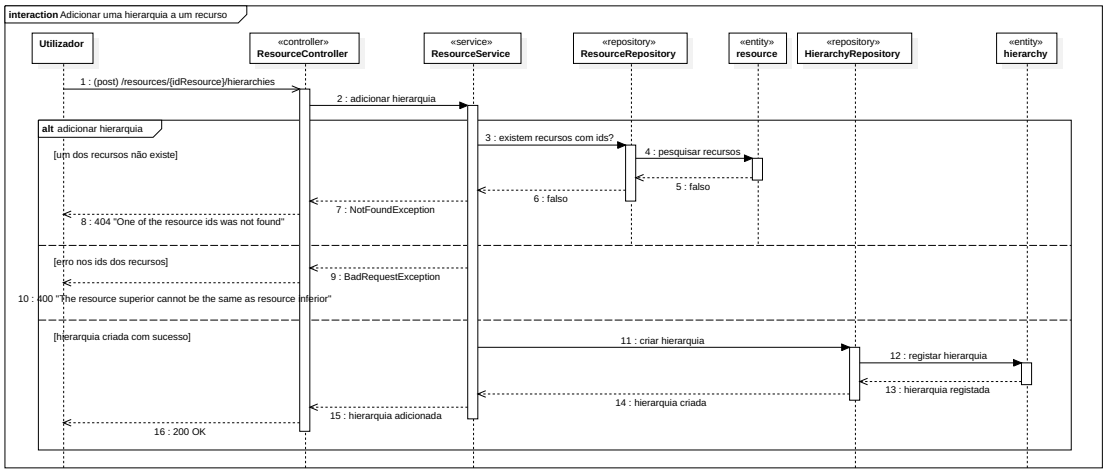

Figura 4.17: Adicionar uma hierarquia a um recurso

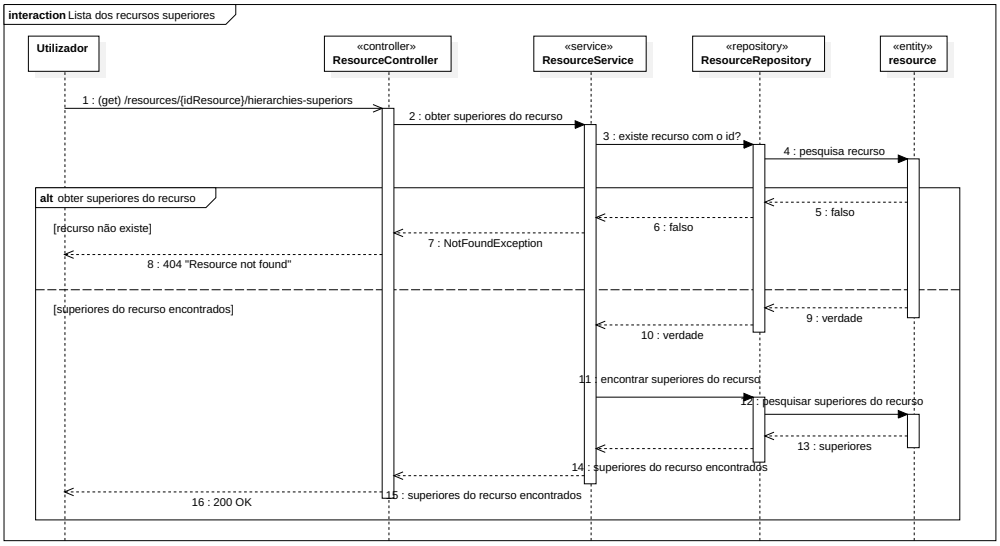

Figura 4.18: Lista dos recursos superiores

4.2.3.10 Lista dos recursos subordinados

Este *endpoint* que executa a operação contraria do exposto no ponto anterior. Neste exemplo são listados os detalhes dos recursos subordinados ao recurso indicado, ou seja, aqueles a quem o recurso indicado subordina, como apresentado na figura 4.19.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.5, ou 4.2.8.2.

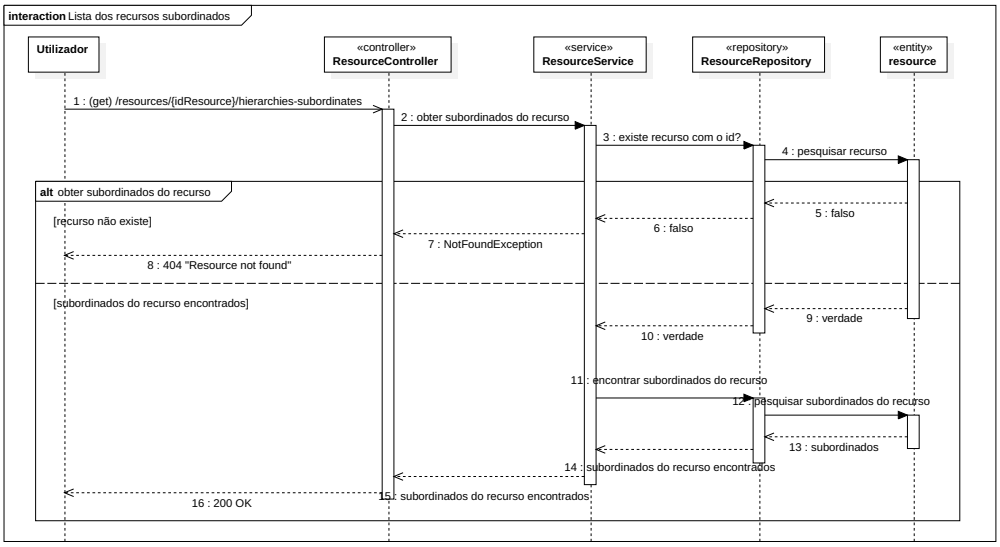

Figura 4.19: Lista dos recursos subordinados

4.2.4 Despesas

4.2.4.1 Obter despesa pelo id

Para se obter os detalhes de uma despesa pelo seu *id* é executado o *endpoint* descrito na figura 4.20. É feita a pesquisa da despesa com o *id* indicado e, caso o objecto não exista, é enviado o erro 404 "Expense not found"; no caso da não ocorrência de erros a despesa é enviada com sucesso.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.3, ou 4.2.8.2.

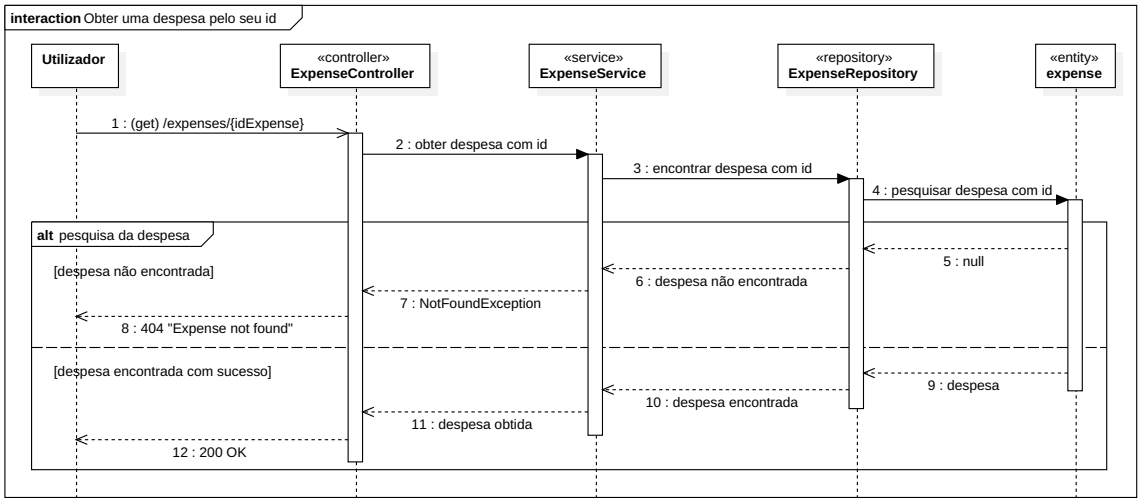

Figura 4.20: Obter despesa pelo id

4.2.4.2 Apagar uma despesa

Quando se pretende eliminar uma despesa da plataforma é efetuado o *endpoint* representado pelo diagrama de sequências da figura 4.21. Depois de se verificar que a despesa com o *id* indicado se encontra na base de dados, é pedido para a mesma ser removida do sistema.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.3, ou 4.2.8.2.

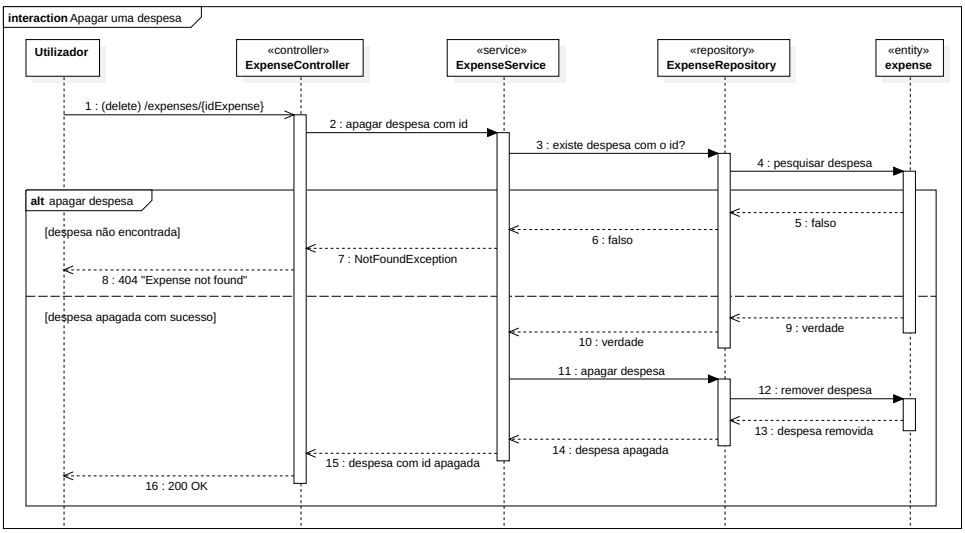

Figura 4.21: Apagar uma despesa

4.2.4.3 Atualizar uma despesa

Se for necessário atualizar algum campo de determinada despesa, por se verificar alguma inconformidade, é utilizado o *endpoint* representado na figura 4.22.

Como esta plataforma suporta vários tipos de despesas (despesas normais e despesas por distância), o servidor tem de efetuar algumas verificações para não existir incoerências na base de dados. Suponhamos que a despesa indicada é uma despesa normal e o utilizador pretende alterar campos das despesas por distância, O servidor não permite que tal coisa aconteça e devolve uma mensagem de erro 400 "Bad Request". O servidor faz então as seguintes verificações:

- **404 "Expense not found"** Caso não seja encontrada a despesa com o *id* indicado.

- **400 "You cannot make changes to the expense, as it has already been validated"** A despesa já foi validada e como tal não se pode proceder a alterações da mesma.

- **400 "You cannot change the cost of a distance expense, as the value is calculated automatically"** Uma despesa por distância tem o seu custo calculado automaticamente segundo os quilómetros efetuados e o valor dado a cada recurso por cada quilómetro feito, logo não se pode alterar o valor da despesa manualmente.
- **400 "You cannot change the value start or end of, as the expense is not a distance type"** Os valores de início e fim de uma despesa são referentes a despesas por distância, logo caso a despesa seja de outro tipo que não o indicado não podem ser preenchidos.
- **400 "You cannot change the number plate, as the expense is not a distance type"** Tal e qual como os campos de início e fim de uma despesa serem referentes a despesas por distância, a matrícula também é referente a esse tipo de despesas, logo não pode ser modificada caso não seja uma despesa de distância.
- **400 "Distance expense doesn't have the correct start and end values"** Uma despesa por distância tem de ter os valores de início e fim devidamente preenchidos, uma vez que o valor de início tem de ser inferior ao do fim. Caso não se verifique devolve o erro indicado.
- **404 "Category not found"** A categoria escolhida que se pretende alterar não existe

Depois de todas estas verificações serem efetuadas podemos proceder à atualização dos campos da despesa indicada.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.3, ou 4.2.8.2.

4.2.4.4 Aprovar despesa

Um superior hierárquico ou um coordenador podem aprovar uma despesa utilizando o *endpoind* observado na figura 4.23. É conferido se a despesa indicada se encontra na base de dados e, caso não se verifique a condição, será enviada a mensagem de erro 404 "Expense not found" e seguidamente verifica-se qual o estado da mesma, ou seja, se esta já foi aprovada é devolvida a mensagem de erro 409 "The expense is already approved", se não se verificar tal situação a despesa é aprovada com sucesso.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.4, ou 4.2.8.2.

4.2.4.5 Reprovar despesa

As despesas tanto podem ser aprovadas como reprovadas; para se proceder à reprovação de uma despesa é executado o *endpoint* evidente na figura 4.24. Depois de uma despesa ser reprovada esta pode voltar a ser reaprovada e quando isso acontece fica aprovada permanentemente, não se podendo reprovar novamente. Ou seja, caso uma despesa se

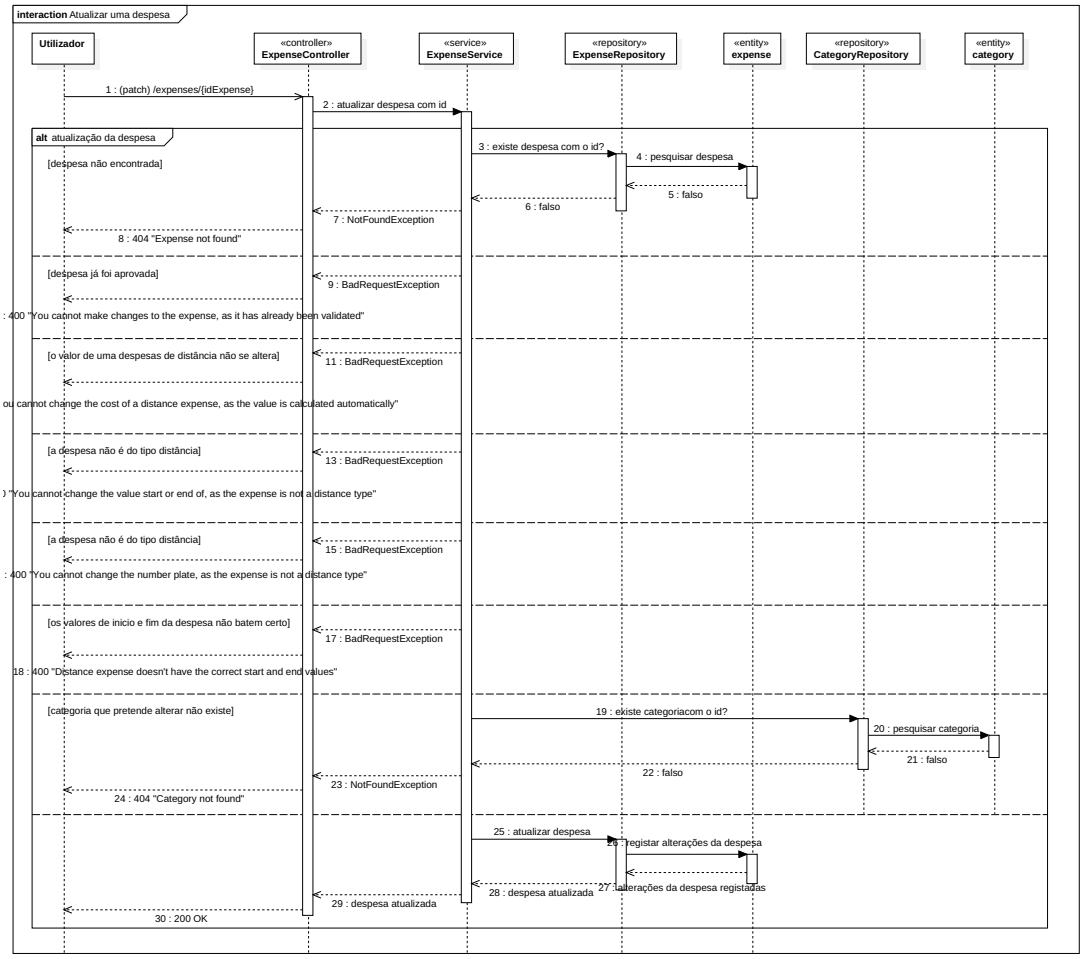

Figura 4.22: Atualizar uma despesa

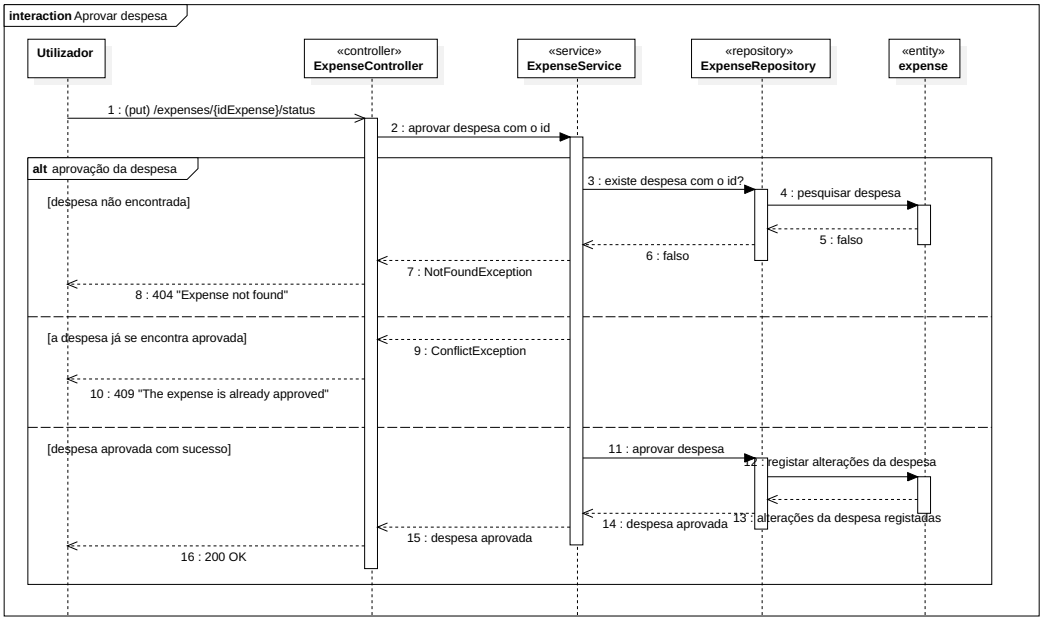

Figura 4.23: Aprovar despesa

encontre permanentemente aprovada não poderá ser novamente reprovada e o servidor devolverá a seguinte mensagem de erro 409 “The expense is permanently approved”.

Tal como acontece com a aprovação das despesas, neste cenário também não é possível reprovar uma despesa se a mesma já se encontrar reprovada. Caso se verifique esta situação, será devolvida a mensagem de erro 409 “The expense is already reproved”. Se nenhum erro se verificar a reprovação da despesa é efetuada com sucesso e as alterações registadas na base de dados da empresa onde essa despesa se encontra.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.4, ou 4.2.8.2.

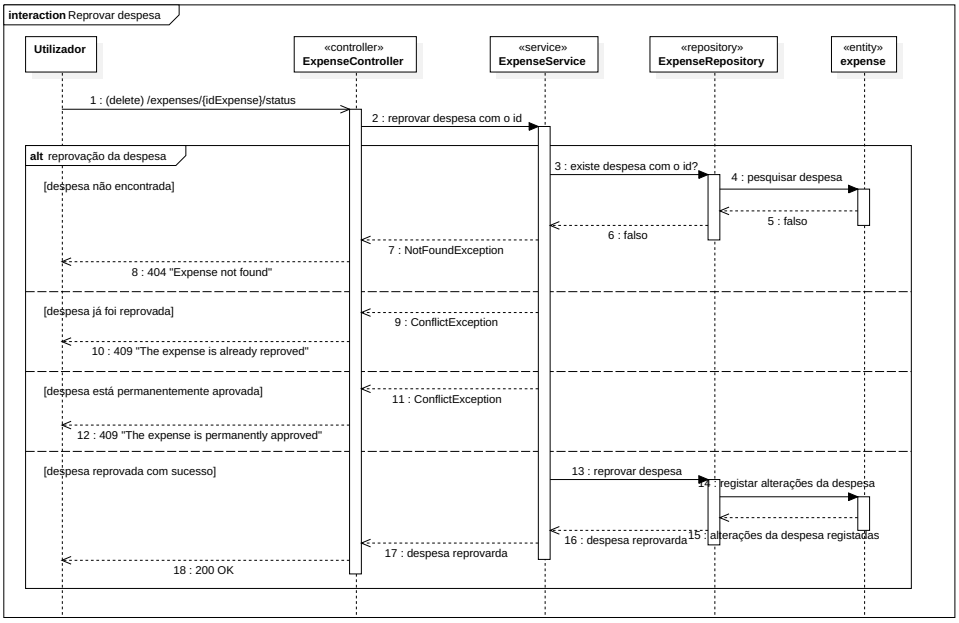

Figura 4.24: Reprovar despesa

4.2.4.6 Criar despesa

Para se proceder à criação de despesas na plataforma é efetuado o *endpoint* descrito na figura 4.25. Tal como na atualização da despesa, neste *endpoint* também é necessário a verificação de determinadas regras de acordo com os diversos campos da despesa, podendo o não cumprimento das regras resultar nos seguintes erros:

- **404 “Category not found”** Quando o *id* da categoria fornecido não se encontra no sistema.
- **404 “Resource not found”** Caso o recurso a que se pretende associar a despesa não exista.
- **400 “Expense type doesn’t exist”** Se o tipo de despesa selecionado não existe.

- **400 "Simple expense doesn't have the estimated value of it"** Nas despesas simples é necessário indicar qual o valor da mesma. Caso contrário o servidor devolve a mensagem de erro indicada.
- **400 "Distance expense doesn't have the correct start and end values"** Uma despesa por distância tem de ter os valores de início e fim devidamente preenchidos, uma vez que o valor de início tem de ser inferior ao do fim. Caso não se verifique será devolvido o erro indicado.
- **400 "Field date, expenseType and idCategory must not be blank"** Os parâmetros indicados na mensagem de erro (tipo da despesa e *id* da categoria) não podem ser brancos ou nulos.

Sendo todas as regras válidas pode-se proceder ao registo da despesa na base de dados.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.5.

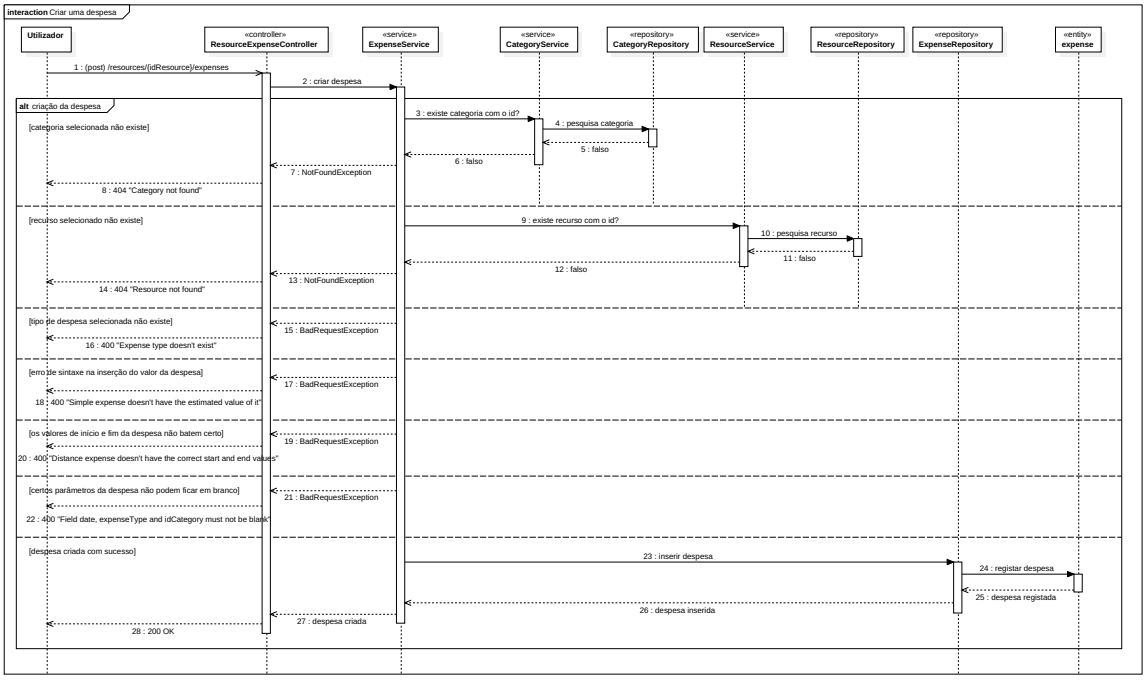

Figura 4.25: Criar despesa

4.2.4.7 Obter despesas do recurso

De modo a se obter as despesas de um determinado recurso numa empresa é utilizado o *endpoint* destacado na figura 4.26. Este *endpoint* permite também a pesquisa das despesas pelo seu estado, por exemplo, se apenas é necessário apresentar as despesas do recurso indicado que se encontram por aprovar, podemos indicar nos parâmetros o número referente ao estado da despesa pretendido e apenas é devolvida a lista desse tipo de despesas.

Se o número indicado não corresponder a nenhum estado correto de despesas, o servidor devolve a mensagem de erro 400 "Status doesn't exist. Must be a value between 1 and 4".

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das três seguintes, ou 4.2.8.5, ou 4.2.8.9, ou 4.2.8.2.

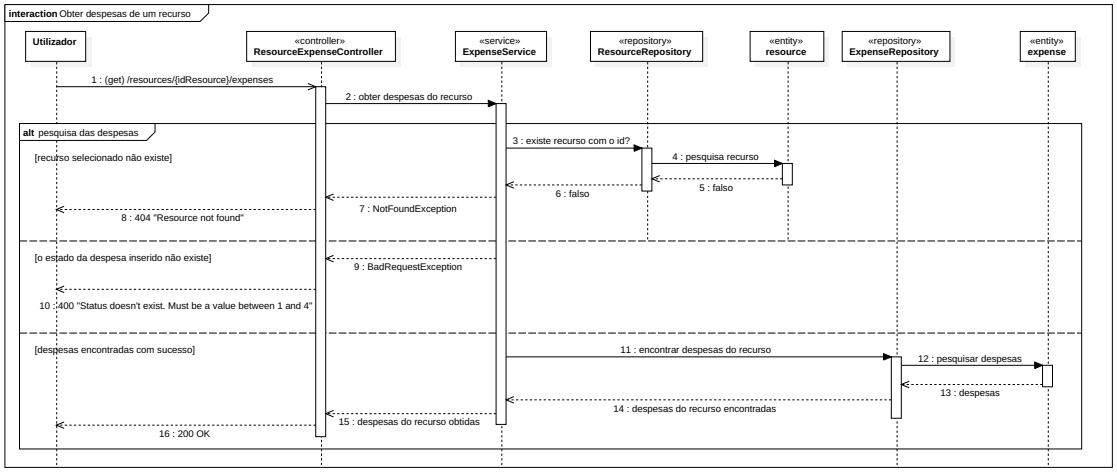

Figura 4.26: Obter despesas do recurso

4.2.4.8 Obter despesas para aprovação

Um recurso que tem a possibilidade de aprovar despesas utiliza o *endpoint* da figura 4.27 para descobrir a lista de despesas que o mesmo tem para aprovação. A única mensagem de erro que este *endpoint* pode devolver é caso o *id* do recurso indicado não existir no sistema.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.5, ou 4.2.8.2.

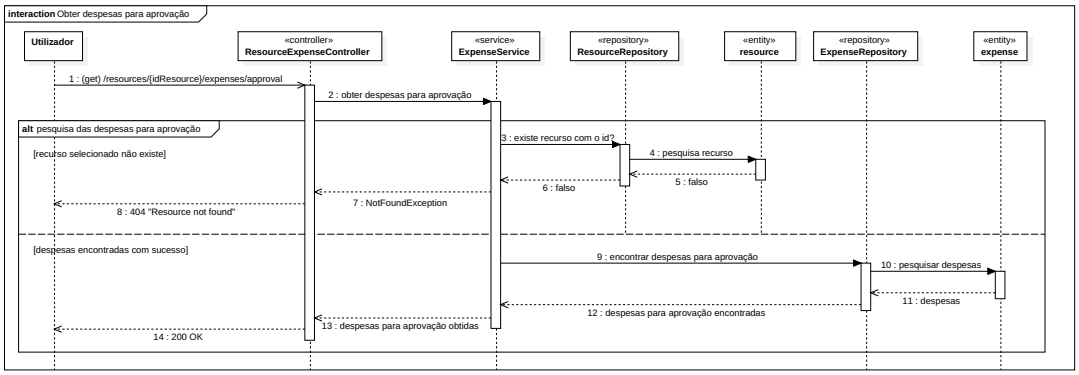

Figura 4.27: Obter despesas para aprovação

4.2.5 Grupos

4.2.5.1 Listar grupos

A listagem dos detalhes de todos os grupos presentes numa determinada empresa é efetuada segundo o *endpoint* representado na figura 4.28.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

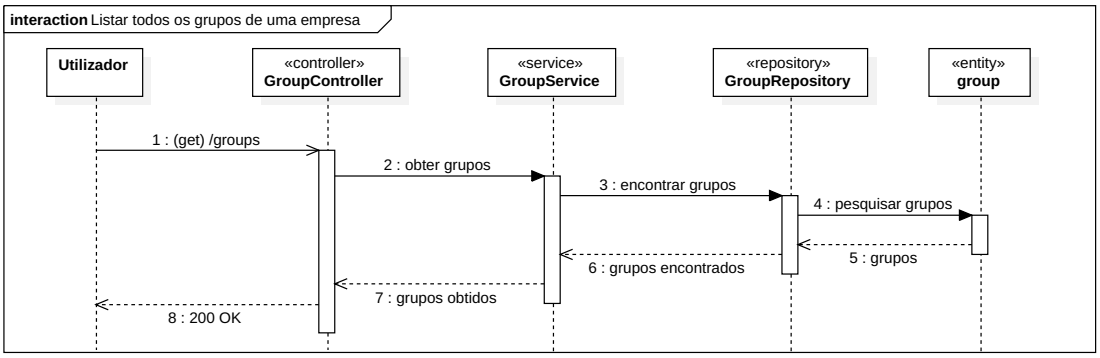

Figura 4.28: Listar grupos

4.2.5.2 Obter grupo

A fim de se obter os detalhes de um determinado grupo com o *id* indicado, numa dada empresa utilizamos o *endpoint* visualizado na figura 4.29, e em caso do *id* apresentado não se encontrar no sistema, o servidor devolve a seguinte mensagem de erro 404 "Group not found".

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.8, ou 4.2.8.2.

4.2.5.3 Obter recursos de um grupo

Quando se pretende exibir os detalhes de todos os recursos presentes num determinado grupo é executado o *endpoint* exposto na figura 4.30.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e uma das duas seguintes, ou 4.2.8.8, ou 4.2.8.2.

4.2.5.4 Criar grupo

Para se proceder à criação de um grupo observamos o *endpoint* da figura 4.31. O recurso responsável pela criação do grupo tem de ter a categoria de coordenador, pois só estes podem criar grupos. Como referido anteriormente um recurso só pode pertencer a um grupo, logo se o coordenador que proceder à criação do novo grupo já pertence a outro, o servidor devolve uma mensagem de erro de conflito 409 "Resource already has group".

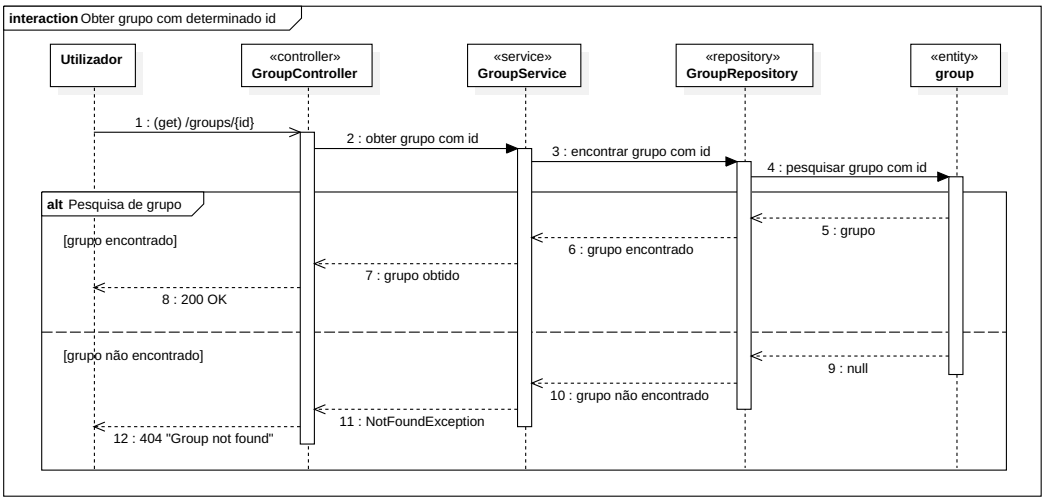

Figura 4.29: Obter grupo

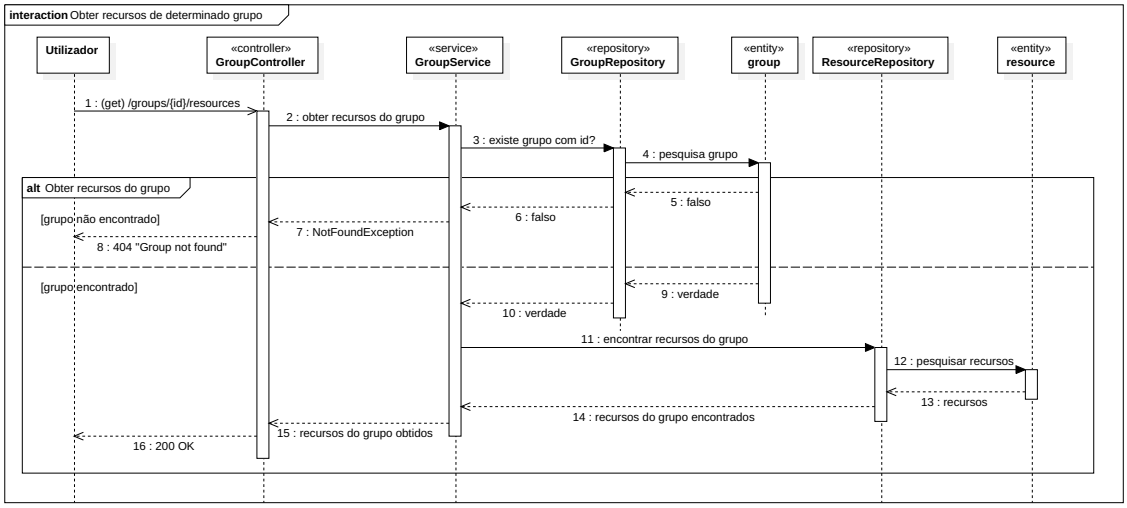

Figura 4.30: Obter recursos de grupo

Dentro de uma empresa não podem existir grupos com o mesmo nome, caso contrário durante a sua criação será devolvida a mensagem de erro 409 "Group already exist", e o nome não pode ser branco ou nulo (400 "Group name cannot be blank").

Estando todas as regras anteriores ultrapassadas pode-se proceder à criação do grupo com o recurso que o criou como seu coordenador.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.6.

4.2.5.5 Adicionar coordenador ao grupo

Como referido anteriormente, um grupo poderá ter vários coordenadores e para se proceder à adição dos mesmos no grupo recorremos ao *endpoint* descrito na figura 4.32.

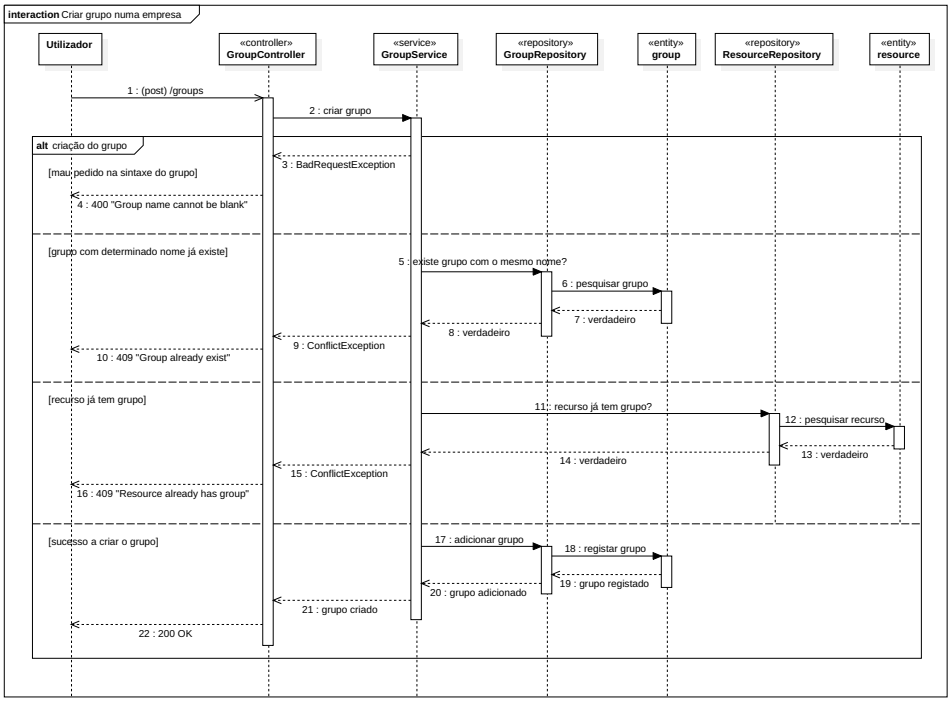

Figura 4.31: Criar grupo

Primeiramente verificamos se o recurso e o grupo com os *id*'s indicado existem, caso contrário o servidor devolve as mensagens de erro 404 "Resource not found" e 404 "Group not found" respetivamente. Seguidamente é verificado a existência dos seguintes conflitos:

- **409 "Resource is not coordinator"** Para se adicionar o recurso como coordenador de um grupo é necessário que o mesmo esteja eleito como coordenador, como explicado no ponto 4.2.3.6, caso contrário será devolvido o erro indicado.
- **409 "Resource already is coordinator of group"** Se o recurso selecionado já é coordenador do grupo, será enviada a mensagem de erro em questão.
- **409 "Resource already belongs to another group"** Como regra do sistema um recurso só pode pertencer a um grupo, caso se pretenda adicionar um coordenador que já pertence a outro grupo é devolvida a mensagem de erro indicada.

Concluídas as verificações existem duas possibilidades para o desfecho deste *endpoint*. No primeiro caso o recurso selecionado não pertence a nenhum grupo, como tal ele será adicionado ao grupo indicado e será registado como coordenador desse mesmo grupo. A outra hipótese verifica-se quando o recurso indicado já pertence ao grupo em questão, mas não é coordenador do mesmo, nesse caso, apensas passa a estar registado como coordenador desse grupo.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

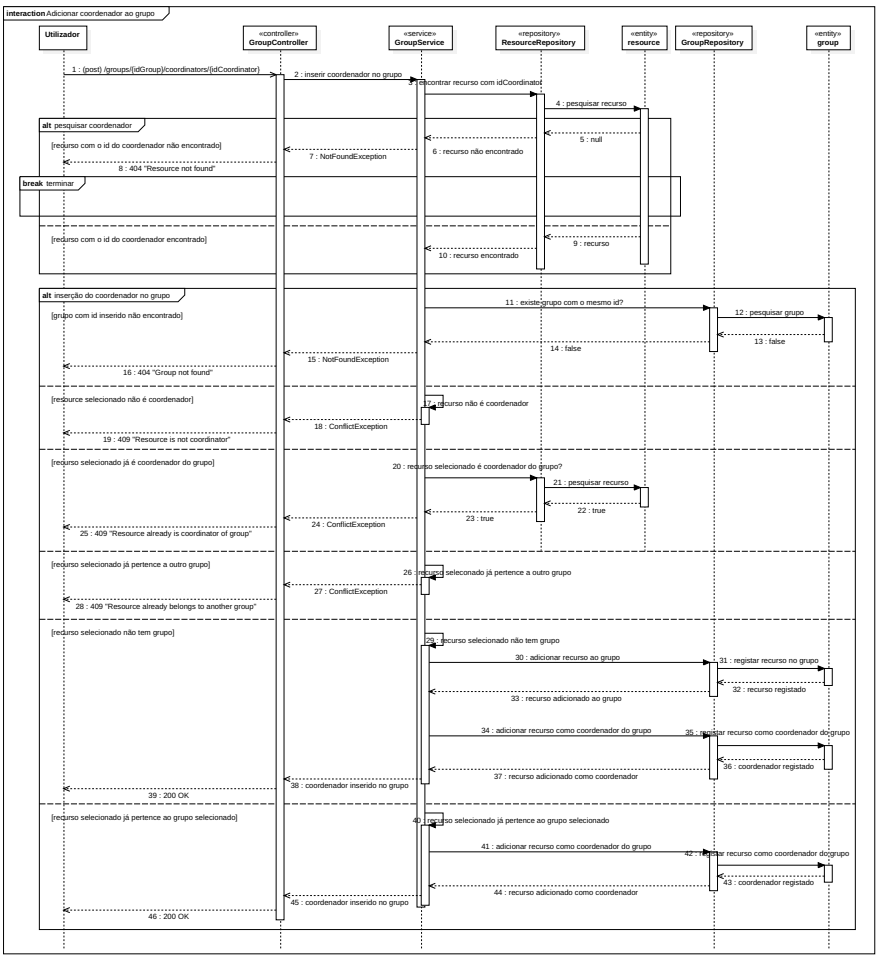

Figura 4.32: Adicionar coordenador ao grupo

4.2.6 Categorias

4.2.6.1 Criar categoria

De modo a ser possível ao administrador de uma empresa adicionar mais categorias representativas das despesas é utilizado o *endpoint* representado na figura 4.33. Evidentemente não podem ser criadas categorias com um nome já existente, caso contrário é devolvida a mensagem de erro 409 "Category already exist" e o nome da categoria que se pretende criar não pode ser nulo ou branco. Sendo estas directrizes cumpridas é possível proceder ao registo da nova categoria na base de dados correspondente à empresa em questão.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

4.2.6.2 Listar categorias

A listagem das categorias das despesas presentes no sistema é efetuada pelo *endpoint* observado na figura 4.34.

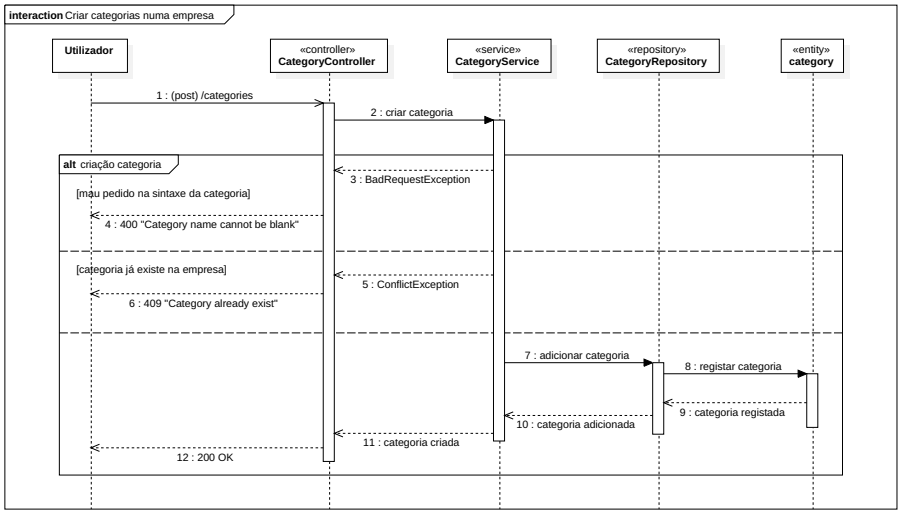

Figura 4.33: Criar categoria

Este *endpoint* tem como pré-requisito à sua autorização a validação explícita em 4.2.8.5.

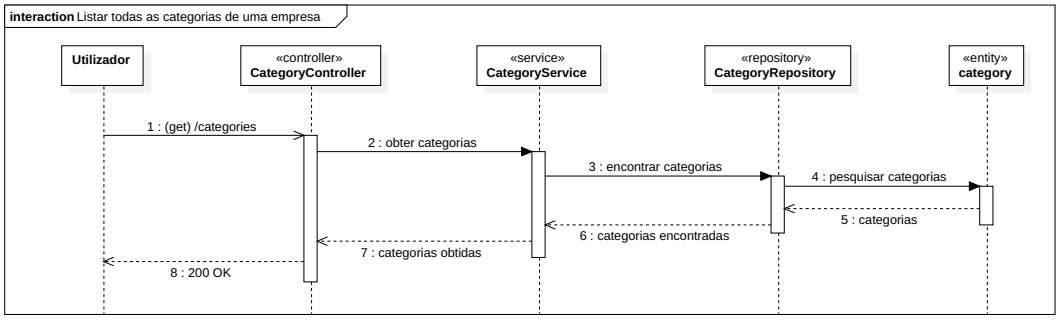

Figura 4.34: Listar categorias

4.2.6.3 Apagar uma categoria

Se for necessário apagar uma determinada categoria do sistema é utilizado o *endpoint* representado na figura 4.35. Onde é pesquisada a categoria por um *id* indicado, que caso não se encontre no sistema é devolvida a mensagem de erro 404 "Category not found". Se a determinada categoria se encontrar no sistema, a mesma é removida da base de dados correspondente à empresa indicada.

Este *endpoint* tem como pré-requisitos à sua autorização a validação explícita em 4.2.8.1 e 4.2.8.2.

4.2.7 Autenticação

O processo de autenticação dos utilizadores sempre que é efetuado um pedido ao servidor de recursos é realizado de forma automática pelo Spring através de uma configuração nas

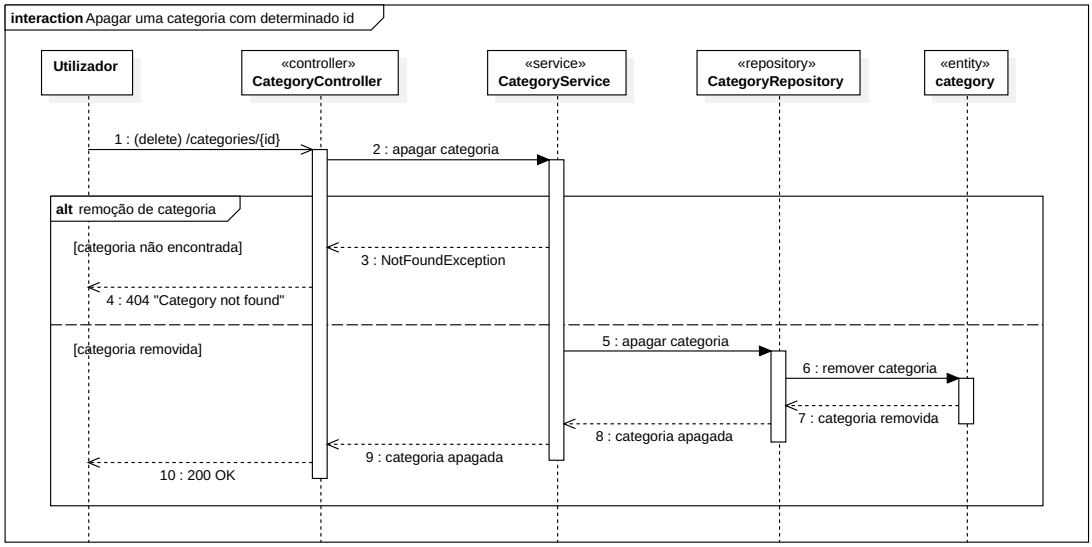

Figura 4.35: Apagar uma categoria

propriedades da aplicação. Desta forma, sempre que o servidor recebe um pedido que necessita de autenticação, a primeira coisa que ele faz é efetuar um pedido com o *token* de acesso enviado para a URI indicada, neste caso a URI escolhida representa um *endpoint* do servidor de autenticação (/oauth/check_token). Este *endpoint* verifica se o *token* de acesso enviado é válido, caso já tenha expirado, ou tenha sido revogado, é enviada uma mensagem de erro 401 "Unauthorized".

Os *endpoints* que não necessitam de autenticação são: login (4.2.1.1), refresh token (4.2.1.3), signup (4.2.1.4) e aceitar convite (4.2.3.4)

4.2.8 Autorização

Neste ponto serão abordadas as verificações efetuadas pelo servidor sobre as permissões do utilizador em cada *endpoint*. O não cumprimento destas regras por parte do utilizador que efetua o pedido faz com que o servidor de recursos devolva uma mensagem de erro 403 "Forbidden".

4.2.8.1 Tem acesso à empresa?

Esta condição, observada pelo diagrama de sequências da figura 4.36 é uma das principais e utilizada em quase todos os *endpoints*, pois faz a verificação se o utilizador autenticado tem acesso à empresa selecionada para poder fazer operações sobre a mesma.

4.2.8.2 É administrador da empresa?

Esta verificação é utilizada nos *endpoints* onde o administrador da empresa pode fazer operações. Existem casos de *endpoints* onde apenas o administrador os pode executar, ou outras ocorrências em que apesar de não ser uma operação direta do administrador do

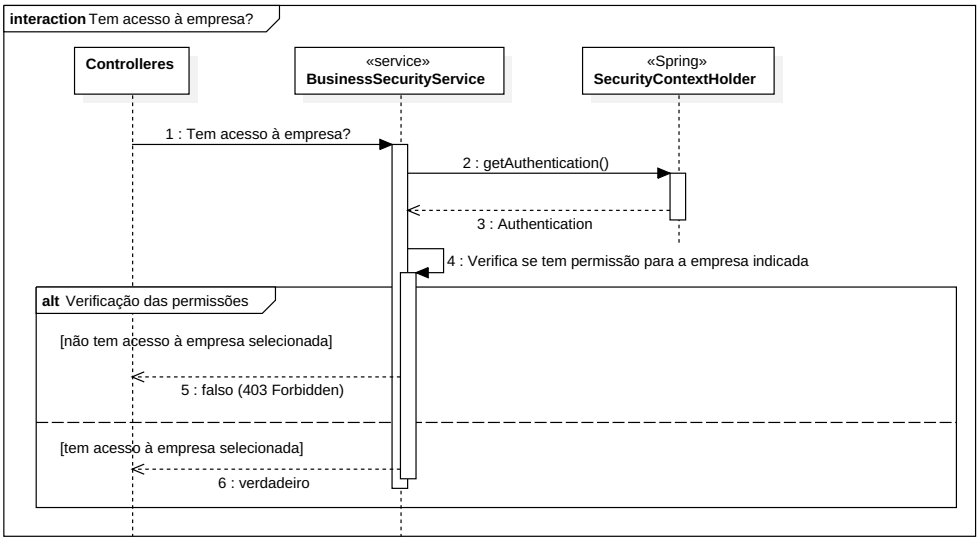

Figura 4.36: Tem acesso à empresa?

sistema, ele também tem a permissão para as executar. Podemos vizualizar essa função através da figura 4.37.

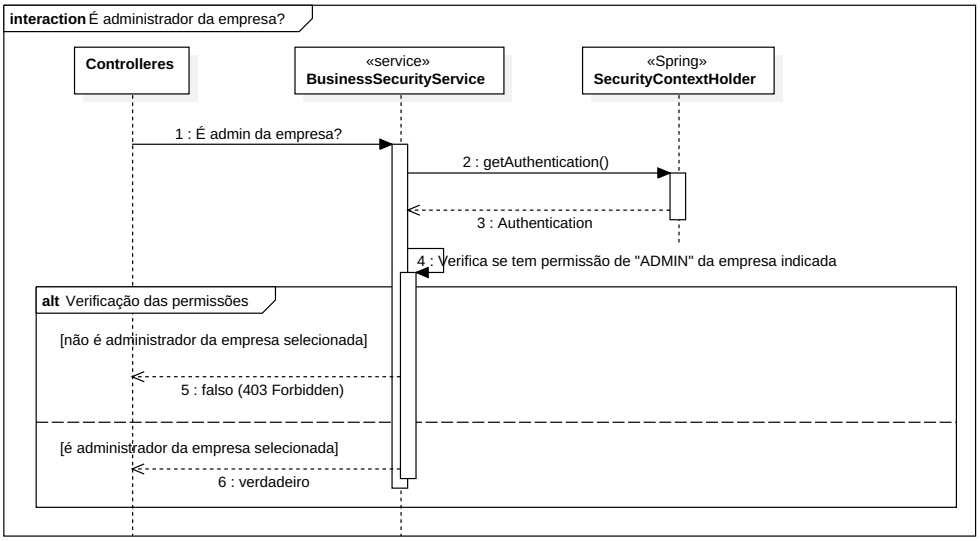

Figura 4.37: É administrador da empresa?

4.2.8.3 Recurso tem acesso à despesa?

A regra para se verificar se o recurso autenticado tem acesso à despesa apresentada pode ser visualizado na figura 4.38. Nesta verificação pesquisa-se pela despesa selecionada e é verificado se a mesma pertence ao recurso autenticado.

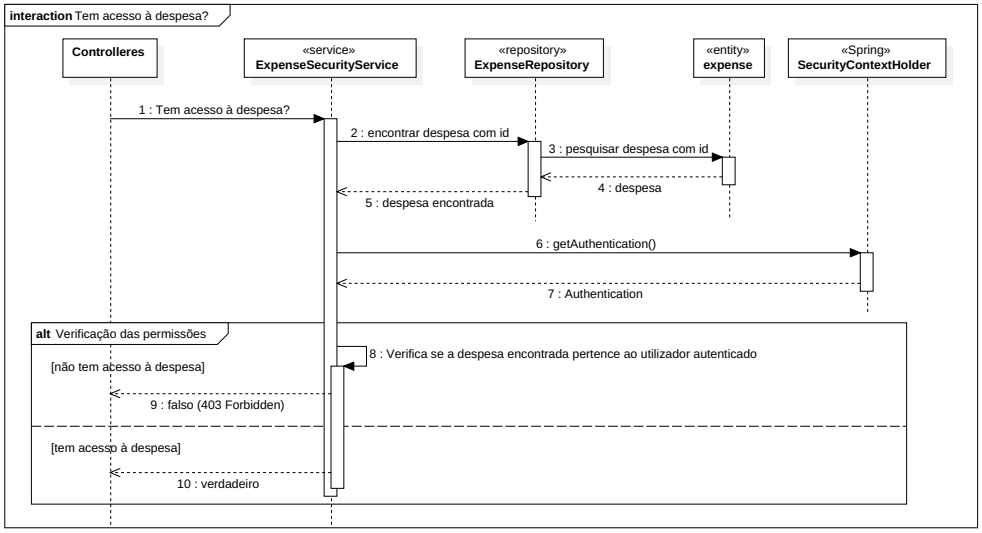

Figura 4.38: Tem acesso à despesa?

4.2.8.4 Recurso pode alterar o estado da despesa?

Quem pode alterar o estado de uma despesa são os coordenadores do grupo a qual o recurso pertença ou um superior hierárquico seu. Como observado na figura 4.39, primeiramente é feita uma pesquisa pela despesa indicada e será posteriormente verificado com auxílio da função explicada em 4.2.8.9 se o recurso autenticado é coordenador ou superior hierárquico do recurso a quem pertence a despesa selecionada.

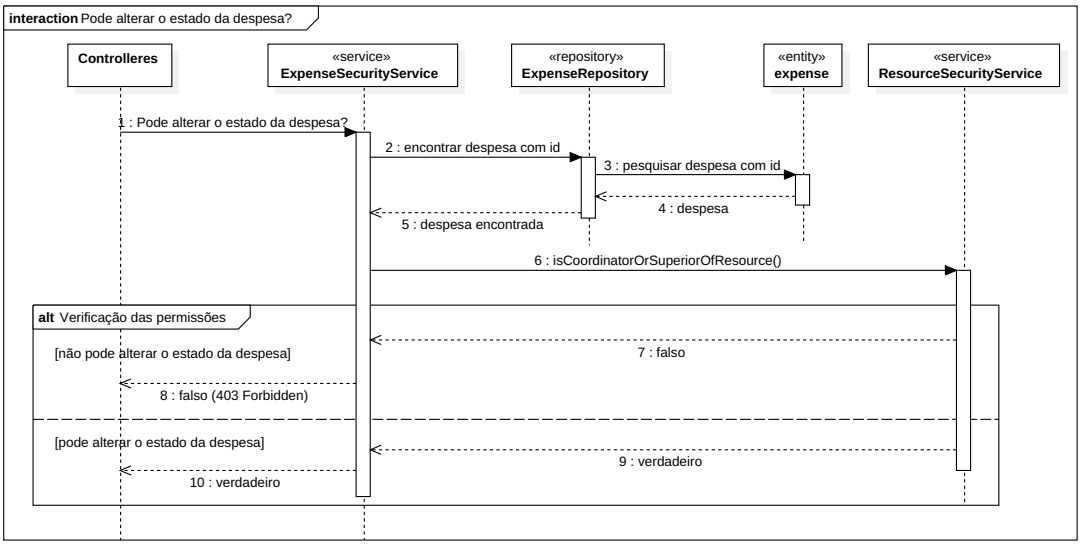

Figura 4.39: Pode alterar o estado da despesa?

4.2.8.5 Tem acesso ao recurso?

Esta verificação serve para confirmar se o utilizador autenticado tem o mesmo *id* do recurso ao qual se pretende fazer uma operação. Está explícita a sua implementação na figura 4.40.

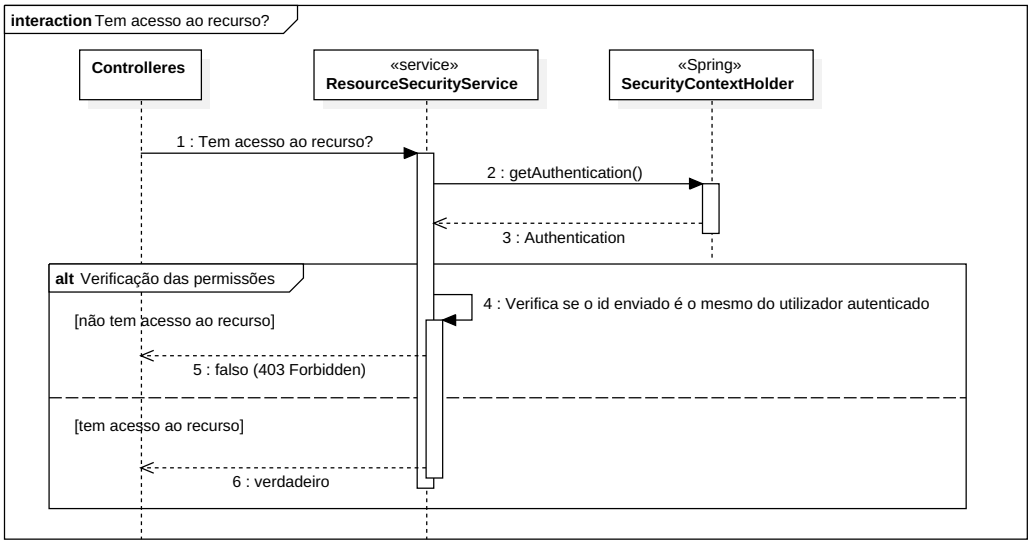

Figura 4.40: Tem acesso ao recurso?

4.2.8.6 Recurso é coordenador?

A fim de se verificar se um recurso autenticado é coordenador numa determinada empresa utilizamos a função explícita na figura 4.41.

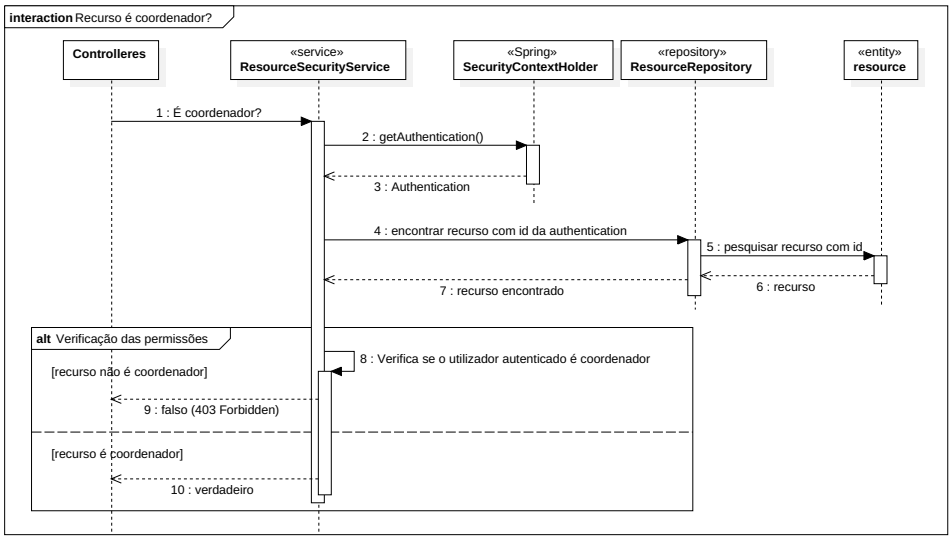

Figura 4.41: Recurso é coordenador?

4.2.8.7 Recurso é coordenador do grupo?

O exemplo da função da figura 4.42 diz respeito à verificação cujo recurso autenticado é ou não coordenador do grupo selecionado.

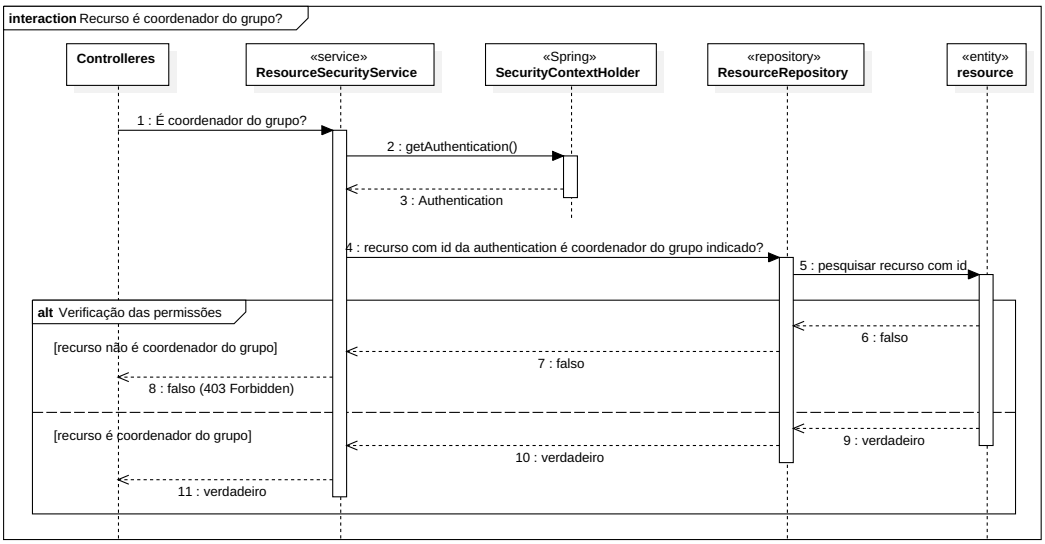

Figura 4.42: Recurso é coordenador do grupo?

4.2.8.8 Recurso pertence ao grupo?

Na figura 4.43 temos uma verificação semelhante à anterior, mas agora diz respeito à confirmação de que o recurso autenticado pertence ao grupo selecionado.

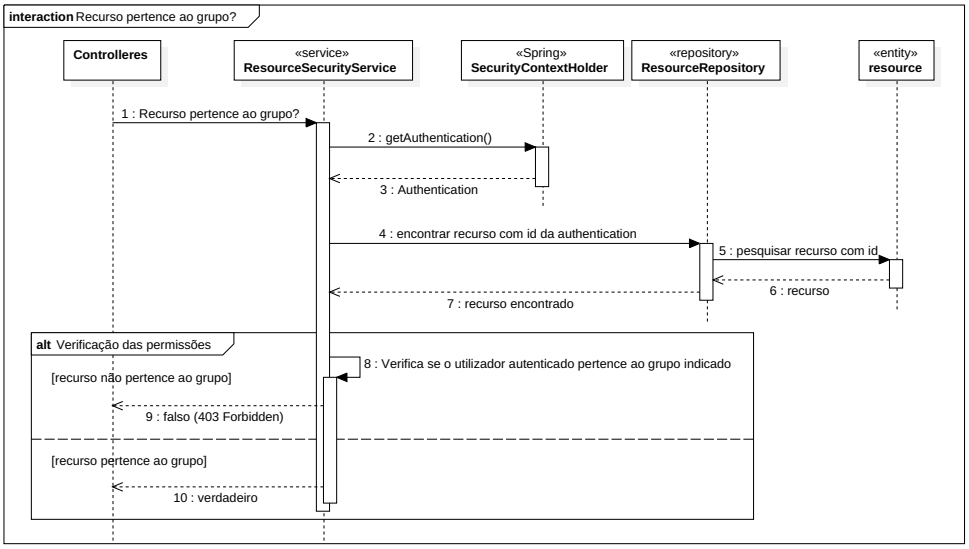

Figura 4.43: Recurso pertence ao grupo?

4.2.8.9 Recurso é coordenador ou superior de outro recurso?

A função presente na figura 4.44 executa a validação se o recurso autenticado é coordenador ou superior hierárquico de um outro recurso indicado. Depois de encontrados os detalhes de ambos os recursos, é validado se se verifica pelo menos uma das condições, ou seja, o recurso autenticado tem de ser coordenador do grupo onde o outro recurso se encontra, ou tem um grau hierárquico superior ao recurso em questão. Não sendo necessário cumprir as duas condições.

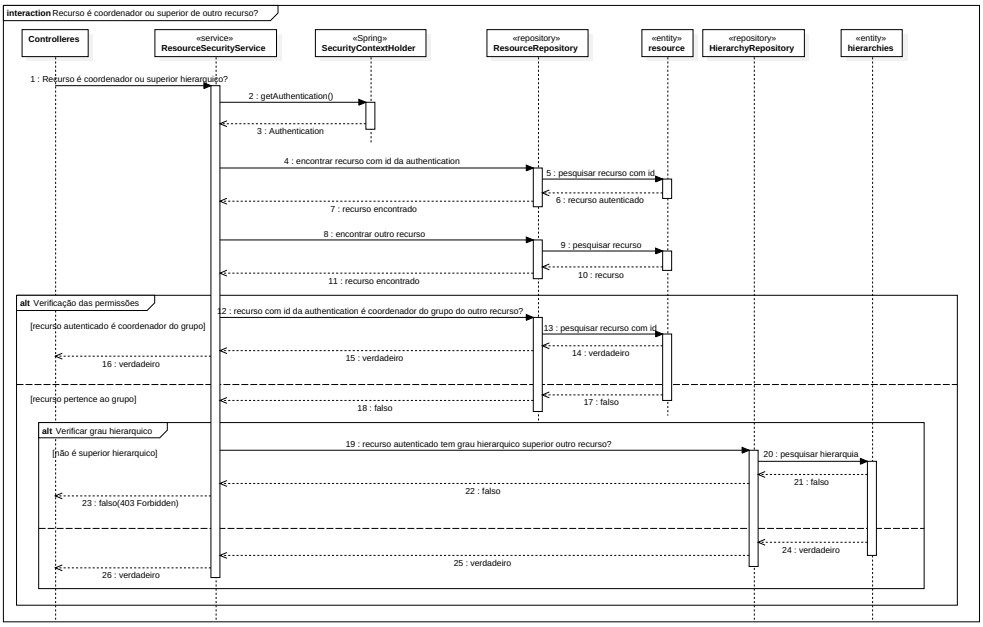

Figura 4.44: Recurso é coordenador ou superior de outro recurso?

4.3 Servidor de Autenticação

Para a implementação do servidor de autenticação é utilizada uma funcionalidade do Spring que facilita muito na criação destes servidores, onde apenas com a implementação de algumas configurações são gerados de forma automática *endpoints* que possibilitam a gestão da autenticação. São gerados então os seguintes *endpoints*:

- **/oauth/token** *Endpoint* responsável por efetuar os pedidos dos *tokens* de acesso ao sistema. Dadas as credenciais do utilizador, este efetua a autenticação do mesmo devolvendo os *token* de acesso. Caso se pretenda efetuar a atualização do *token* de acesso é também utilizado este *endpoint*, mas desta vez, ao invés de se enviarem as credencias do utilizador é enviado o refresh token.
- **/oauth/check_token** Para se proceder à confirmação de que o *token* de acesso se encontra válido, é utilizado este *endpoint* onde é enviado como parâmetro o respetivo *token* a confirmar.

Para além dos *endpoints* indicados foram também criados outros de modo a se conceber um servidor que correspondesse às necessidades do projeto, os quais serão enunciados de seguida com auxílio a diagramas de sequência que ajudam a compreender o seu funcionamento.

4.3.1 Adicionar empresa

É utilizado o *endpoint* representado na figura 4.45 para adicionar uma nova empresa com as permissões de um determinado utilizador. O nickname desta empresa não pode ser nulo ou ter espaços em branco, caso contrário é devolvida a mensagem de erro 400 "The business nickname cannot be null or have whitespaces". O utilizador que procede à criação da empresa terá a permissão de ADMIN nessa mesma empresa sendo essa permissão registada na base de dados simultaneamente com o registo da empresa.

Por fim utilizando uma classe específica do Spring que guarda as permissões do utilizador autenticado é procedida a atualização do mesmo com as novas permissões dentro da empresa criada.

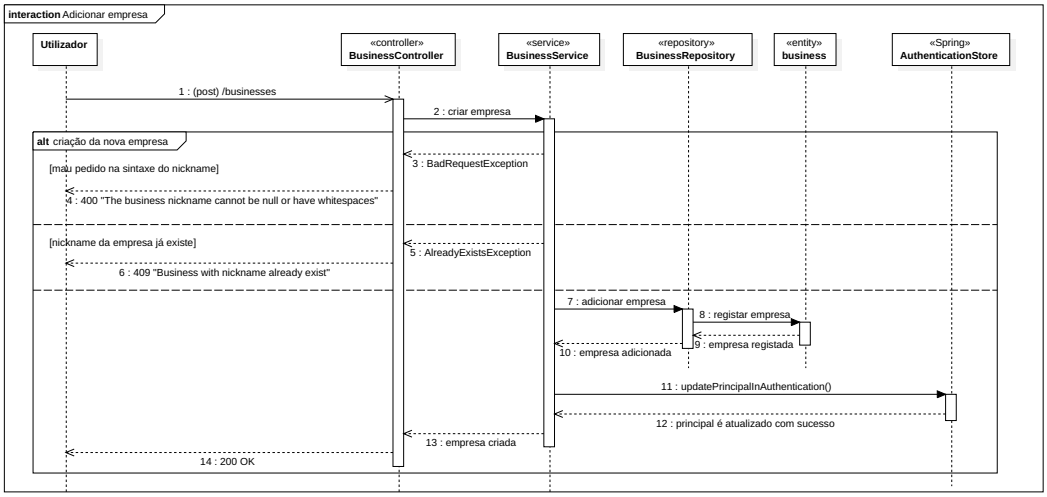

Figura 4.45: Adicionar empresa

4.3.2 Convidar utilizador para uma empresa

É possível convidar para ingressar numa empresa tanto utilizadores que já se encontrem registados na plataforma, como utilizadores ainda não registados (figura 4.46). Para o caso de utilizadores que não se encontrem registados no sistema quando se processa o convite para ingressar numa dada empresa, é fornecido o seu email e enviado um convite para o ingresso na empresa como para o seu registo na plataforma. Para isso cria-se um utilizador temporário, apenas com o campo do email preenchido e desactivado e quando esse utilizador proceder ao seu registo é automaticamente adicionado à empresa. No

outro cenário em que o utilizador já se encontra registado no sistema é apenas enviado um email para com um *link* e um *token* de confirmação para o mesmo aceitar o convite.

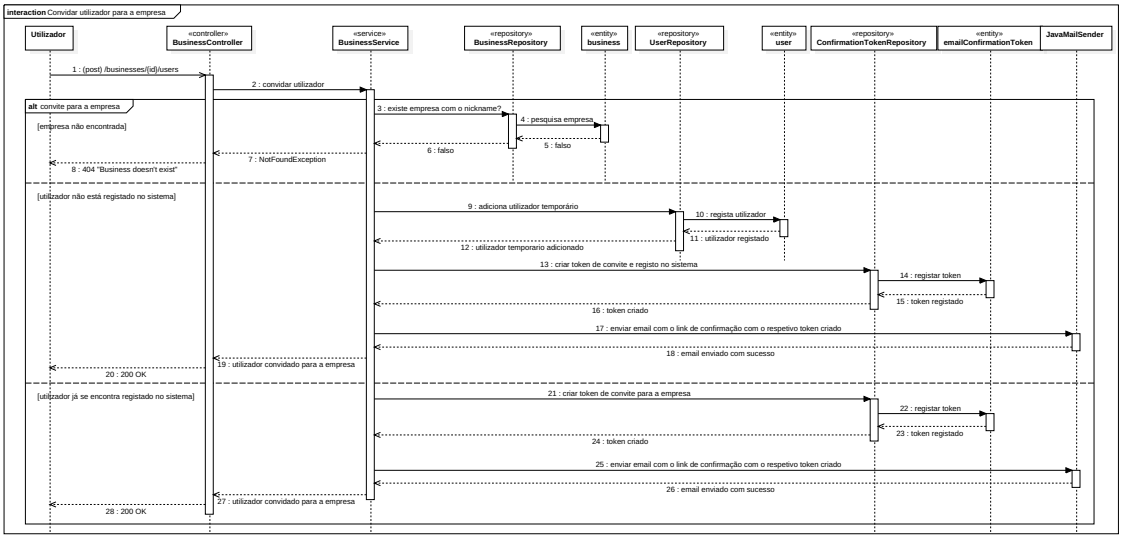

Figura 4.46: Convidar utilizador para uma empresa

4.3.3 Revogar tokens de acesso

Visto que dentro dos *endpoint* fornecidos automaticamente pelo Spring não existe nenhum para anular os *token* de acesso dos utilizadores autenticados, foi necessário proceder à criação de um, sendo possível observar o seu funcionamento através da figura 4.47. Neste utilizamos a função "revokeToken()" da classe DefaultTokenServices fornecida pelo Spring para efetuar operações com os *tokens* onde é indicado o *token* de acesso correspondente.

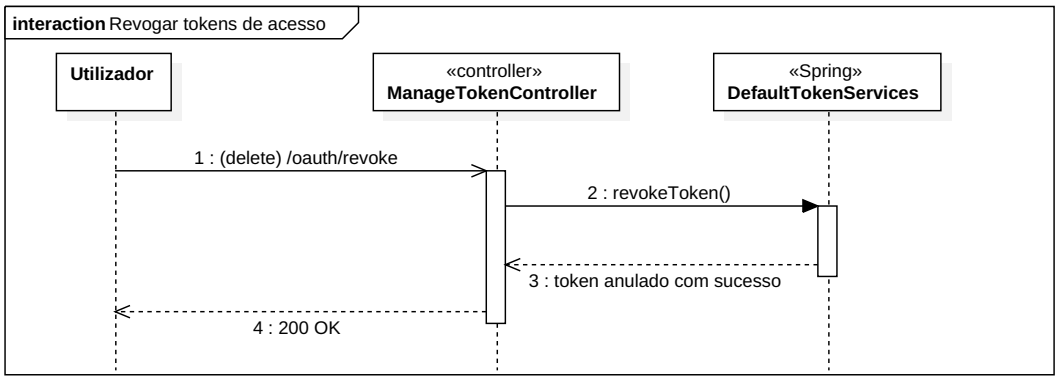

Figura 4.47: Revogar tokens de acesso

4.3.4 Criar utilizador

Com o objetivo de se registarem utilizadores no sistema, é apresentado o *endpoint* observado na figura 4.48 no qual se processa o registo dos mesmos. Caso já exista um utilizador com o email indicado o servidor, será apresentada a seguinte mensagem de erro 409 "User already exists". Não existindo nenhum erro o utilizador é registado na base de dados com as suas informações, mas com a *flag* "isEnable" a falso. É então criado um token de confirmação do resgisto que é enviado para o email do utilizador com um *link* para se proceder à sua confirmação.

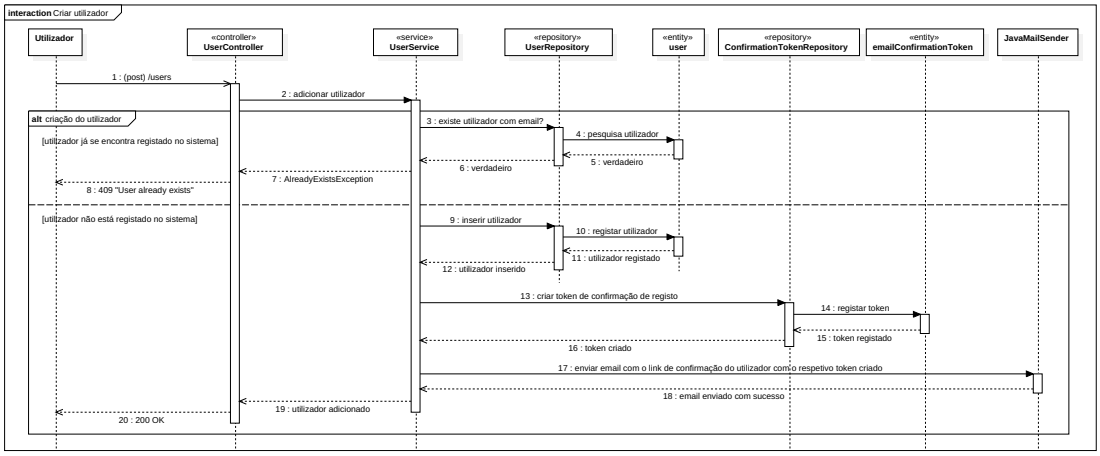

Figura 4.48: Criar utilizador

4.3.5 Obter utilizador pelo token de acesso

Para se obter os detalhes com as permissões do utilizador autenticado, é utilizado este *endpoint* exemplificado na figura 4.49.

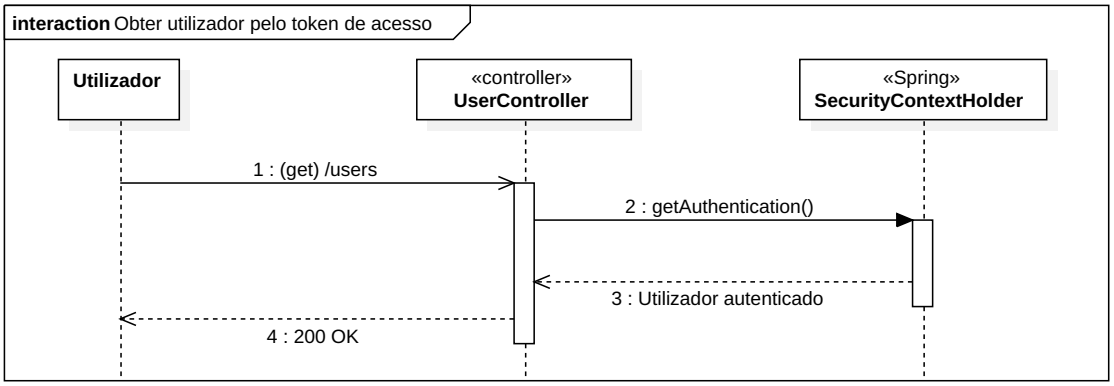

Figura 4.49: Obter utilizador pelo token de acesso

4.3.6 Atualizar utilizador

Caso seja necessário efetuar alguma alteração nas informações dos utilizadores o servidor irá deter o *endpoint* observado na figura 4.50. O mesmo atualiza os detalhes do utilizador não permitindo que este altere o email para um já presente no sistema, respondendo desta forma com uma mensagem de conflito 14 : 409 "User already exists". Sem nenhuma ocorrência de erros o servidor procede à alteração dos dados e à atualização desses novos dados na AuthenticationStore.

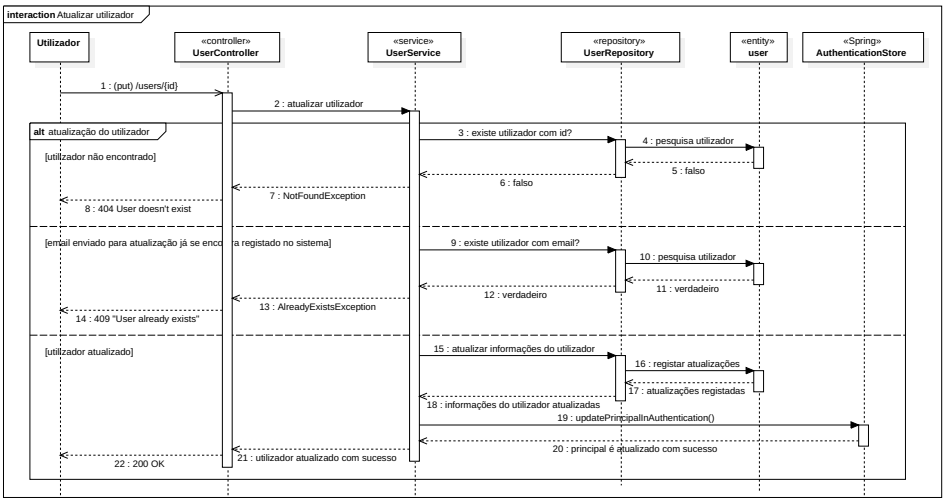

Figura 4.50: Atualizar utilizador

4.3.7 Confirmar registo do utilizador

Para se proceder à confirmação do registo do utilizador é enviado um pedido para o *endpoint* descrito na figura 4.51 onde é indicado o *token* fornecido no *link* enviado para o email do utilizador. Caso esse token esteja incorreto o servidor devolve uma mensagem de erro 404 "The email confirmation token was not found", caso contrário o *token* é encontrado correspondendo a um determinado utilizador e o mesmo é confirmado passando a estar ativo na base de dados e já sendo possível efetuar operações com ele.

4.3.8 Confirmar convite para a empresa

Depois do utilizador ter recebido no email o convite para ingressar numa empresa é redirecionado para um *link* que corresponde ao *endpoint* observado na figura 4.52. Caso não existam erros com o *token* enviado e este tenha uma relação com a empresa indicada, é então adicionado o utilizador com a permissão USER à empresa a qual foi feito o convite.

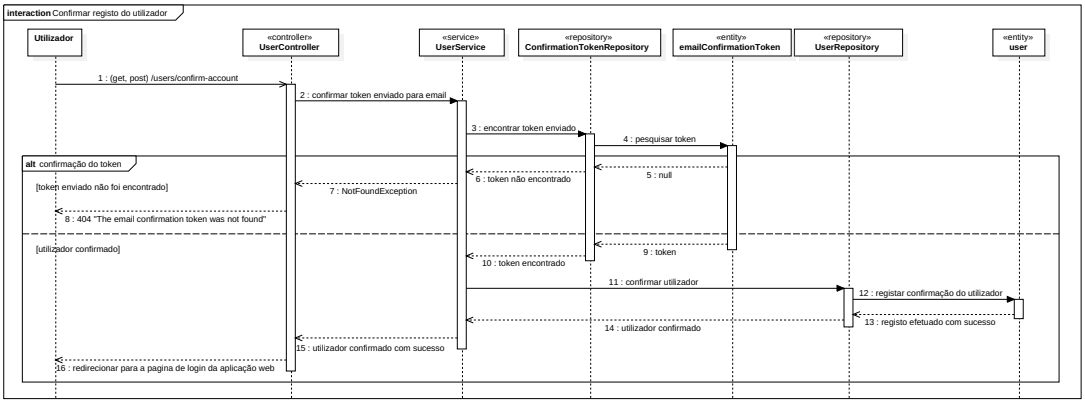

Figura 4.51: Confirmar registo do utilizador

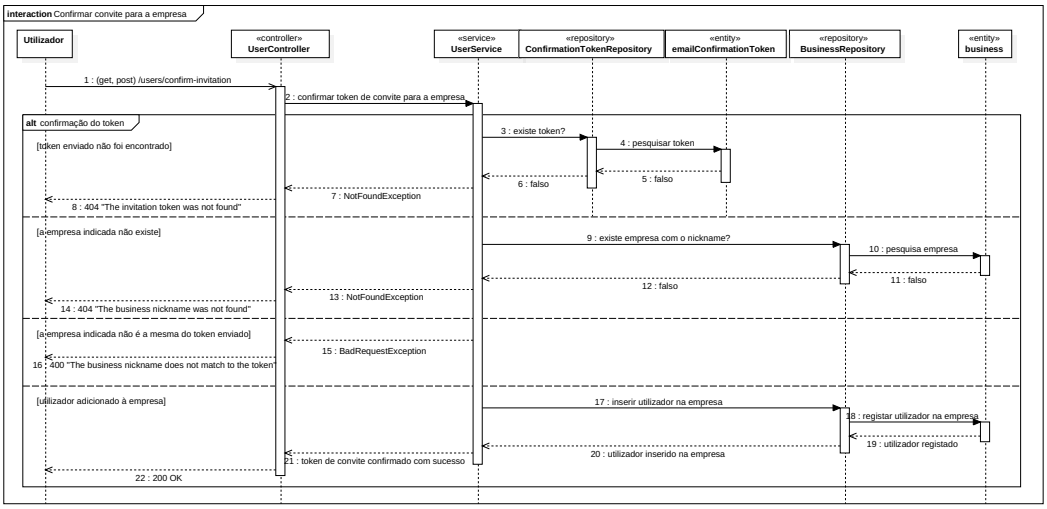

Figura 4.52: Confirmar convite para a empresa

4.3.9 Cron job

Dando-se o caso de existirem utilizadores que se registem na plataforma, mas nunca cheguem a confirmar o seu registo, ou sejam efetuados convites a utilizadores não registados na plataforma e estes nunca aceitem os mesmos e como tal ficará um utilizador temporário na base de dados apenas com o email preenchido. Outro exemplo de utilizadores registados na plataforma que estejam inativos seria a utilização de um bot para efetuar o registo de inúmeros utilizadores, sem proceder à sua confirmação.

Para colmatar o problema da existência de diversos utilizadores inativos que só ocupam espaço de armazenamento na base de dados foi criado um cron job dentro do servidor de autenticação que será executado todas as segundas feiras ao meio dia, removendo da base de dados todos os utilizadores inativos em que a sua data de criação seja superior a mais de uma semana.

4.4 Aplicação Web

De modo a ser possível ao utilizador ter um contacto visual com a plataforma e uma melhor precessão das funcionalidades do sistema, foi criado um protótipo de uma aplicação web que comunica com a API criada e que contém as principais funcionalidades desenvolvidas no sistema.

A figura 4.53 mostra lado a lado a página inicial da plataforma e à esquerda um utilizador acabado de se registar no sistema e que não está integrado em qualquer empresa Desta forma, o utilizador tem imediatamente a possibilidade de criar ele mesmo uma empresa e começar a convidar utilizadores para a mesma. No lado esquerdo da imagem visualizamos a pagina inicial quando um utilizador já se encontra agregado a uma ou mais empresas e são apresentados atalhos para o mesmo escolher em que empresa pretende efetuar operações.

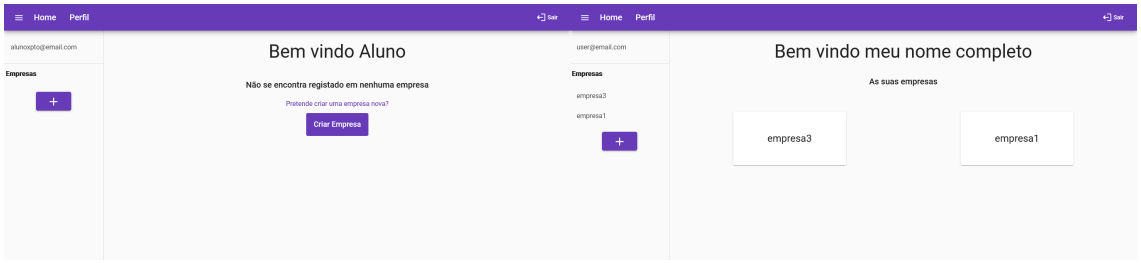

Figura 4.53: Página principal da plataforma com e sem empresas

Dentro da empresa escolhida o utilizador tem acesso às despesas efetuadas nessa mesma empresa e a visualização de alguns detalhes, como por exemplo o estado em que se encontra (figura 4.54). Caso pretenda criar uma nova despesa, seleciona o botão no canto superior direito logo a cima da tabela das despesas e, como é apresentado na figura 4.55, surge uma *dialog* para preenchermos os campos da despesa. A *dialog* tem dois separadores para os dois tipos de despesa existentes, um para uma despesa simples e outro para despesas por distância. Esta divisão é importante visto que cada despesa tem diferentes parâmetros e, de forma a facilitar o trabalho ao utilizador, esses parâmetros já se encontram segmentados possibilitando a não ocorrência de nenhum erro no envio da despesa para a API.

Os utilizadores que são coordenadores de um grupo, têm a opção de aprovar ou reprovar as despesas dos membros dos seus grupos. Visualizamos então na figura 4.56 a lista das despesas que o utilizador tem para aprovar e na imagem da esquerda uma *dialog* com as informações de uma determinada despesa e dois botões no canto inferior esquerdo com a possibilidade de a validar ou reprovar.

O utilizador que é administrador de uma determinada empresa consegue visualizar os grupos existentes na mesma, assim como os recursos existentes, como podemos observar pela figura 4.57. Existem ainda dois botões nesta figura, um para se proceder à criação de um grupo e o outro para adicionar um recurso no grupo.

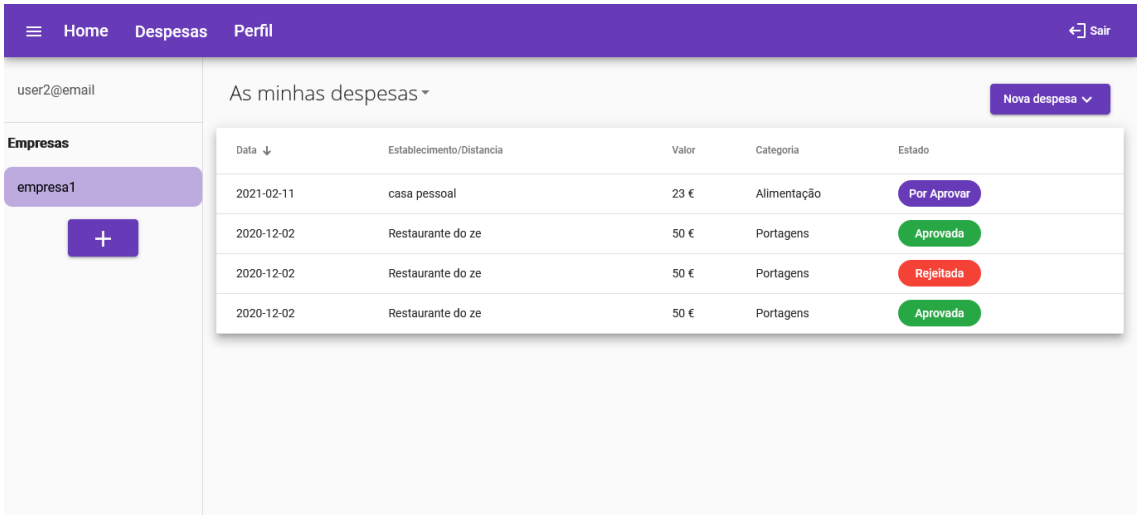

Figura 4.54: Lista das despesas

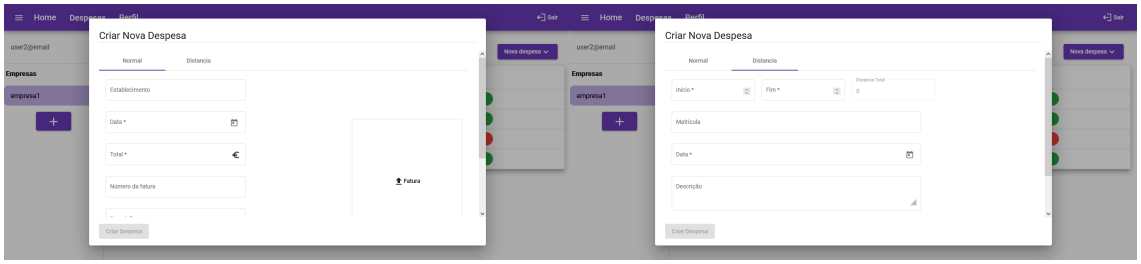

Figura 4.55: Adicionar despesa

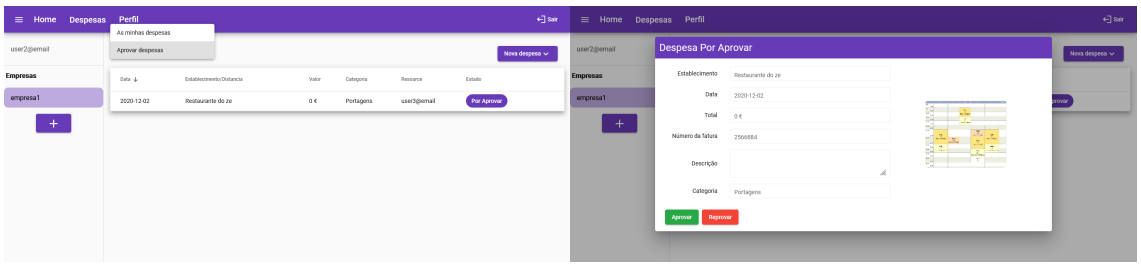

Figura 4.56: Lista das despesas por aprovar e *dialog* de aprovação ou reprovação de uma despesa

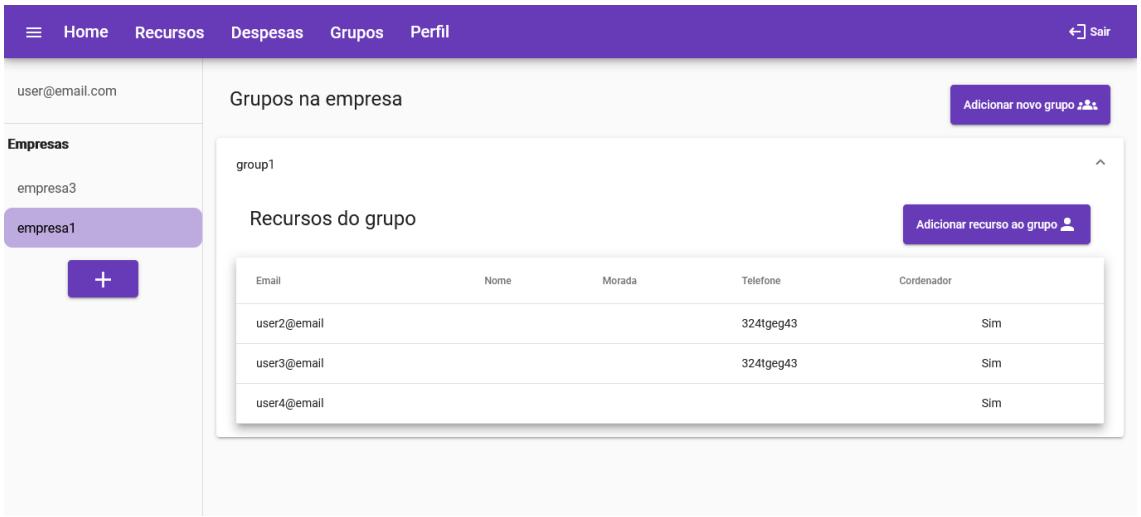

Figura 4.57: Visualização dos grupos numa empresa

5

Conclusões e Trabalho Futuro

5.1 Conclusões

O objetivo desta dissertação passava por elaborar uma plataforma para gestão de despesas de colaboradores de diversas empresas, capaz de numa primeira tentativa implementar as funcionalidades mais simples como a criação de despesas e um sistema de aprovação das mesmas.

O foco principal desta dissertação passou por criar um servidor Web bastante completo e bem estruturado, e foi nele onde se despendeu mais tempo para a sua modelação e concepção. Como resultado proveio um servidor bem desenvolvido e com um bom conceito de arquitectura, mais concretamente a sua independência com o SGBD que efetua a comunicação com a base de dados, que poderá ser tido em consideração para a reestruturação dos serviços da ARTSOFT, como explicado na Motivação e Contexto desta dissertação.

De modo a se conseguirem visualizar de forma mais concreta as funcionalidades da plataforma, foi criado um protótipo de uma aplicação web que comunica com a API do servidor desenvolvido. A aplicação, por se tratar apenas de um protótipo, não implementa na totalidade todas as funcionalidades do servidor, mas foram desenvolvidas as mais importantes, como a criação e aprovação de despesas.

Devo ainda salientar que toda a situação por que passamos atualmente com a pandemia revelou-se numa dificuldade para realização desta dissertação. Sendo eu uma pessoa que é bastante mais produtiva quando trabalha fora de casa, ter de me adaptar a toda uma nova realidade de teletrabalho foi desmotivante para mim, fazendo com que todo o processo de concretização do projeto se tornasse mais demorado.

5.2 Trabalho futuro

Como anteriormente referido, foi desenvolvido um protótipo de uma aplicação web somente com algumas funcionalidades básicas suportadas pelo servidor criado. Como tal, seria vantajoso acrescentar as funcionalidades que o servidor Web apresenta.

No sentido de facilitar a criação de despesas por parte dos colaboradores de uma empresa, seria muito mais vantajoso caso existisse a possibilidade de o seu registo ser preenchido automaticamente apenas com uma fotografia da fatura. Como trabalho futuro, seria muito vantajoso esta plataforma ter integrado um sistema de reconhecimento automático das despesas com recurso à inteligência artificial e ao OCR. Fornecendo a possibilidade de extrair todos os dados relevantes de uma fotografia a uma fatura, reconhecer os padrões tendo em conta o conhecimento de outras faturas já registadas e fazer o registo da mesma no sistema. Inicialmente foi pensada a realização desta funcionalidade na plataforma, o que levou até ao estudo e à pesquisa das tecnologias por detrás do reconhecimento ótico de caracteres que foi efetuada no capítulo 2. Não chegou a ser implementada pois foi dada maior importância à realização de um servidor Web mais complexo.

De forma a aproveitar a funcionalidade descrita anteriormente, seria uma mais valia a criação de uma aplicação móvel que seja mais simples e intuitivo para o colaborador registar as suas despesas, usufruindo das propriedades de um smartphone tal como a sua câmera, para fotografar a sua fatura e fazer o registo automático da mesma.

No capítulo da Implementação da plataforma, quando é abordada a modelação da base de dados do servidor de recursos, refiro no mesmo duas tabelas que apesar de estarem já implementadas no sistema não estão a ser utilizadas. Refiro-me às tabelas que guardam informações sobre as diferentes moedas existentes e as suas taxas de câmbio. Com a utilização destas tabelas o sistema passaria a suportar o registo de despesas com diferentes tipos de moedas e seria calculado o valor da despesa de acordo com a moeda com a qual a empresa escolheu operar.

Seria também interessante a incorporação de mais um tipo de despesas na plataforma. Sabendo que o sistema já suporta despesas simples e despesas por distância, seria conveniente adicionar despesas de ajudas de custo. Neste tipo de despesas o colaborador indicaria a data de início e de fim da sua viagem e o sistema calcularia o valor que o mesmo tem a receber consoante o tempo que demorou na sua deslocação. Portugal tem valores fixos decretados em lei sobre as diárias concebidas aos colaboradores, no entanto esses valores só são aplicados para trabalhadores do setor público e apenas servem de referência para o setor privado. Desta forma cada empresa poderia definir qual o valor que pretende atribuir aos seus colaboradores, sendo que o *standard* seria o definido por lei.

Entrando na comercialização da plataforma, algo que poderia ser desenvolvido seriam determinados pacotes para os utilizadores do sistema. Imaginando que a criação de empresas é limitada a cada utilizador, visto que é necessário a criação de uma base de dados sempre que algum utilizador pretende criar uma, o que tem custos de armazenamento. Desta forma poderiam ser criados pacotes que englobavam a quantidade de empresas que um utilizador podia criar, bem como o número de utilizadores que podem ingressar nas mesmas.

Bibliografia

[1] *Authorization Code Flow*. URL: https://auth0.com/docs/flows/authorization-code-flow (acedido em 11/06/2021).

[2] *Client Credentials Flow*. URL: https://auth0.com/docs/flows/client-credentials-flow (acedido em 11/06/2021).

[3] X. Feng, J. Shen e Y. Fan. "REST: An alternative to RPC for Web services architecture". Em: *2009 First International Conference on Future Information Networks*. 2009, pp. 7–10. DOI: 10.1109/ICFIN.2009.5339611.

[4] R. T. Fielding e R. N. Taylor. "Architectural Styles and the Design of Network-Based Software Architectures". AAI9980887. Tese de doutoramento. 2000. ISBN: 0599871180.

[5] D. Hardt. *The OAuth 2.0 Authorization Framework*. RFC 6749. Out. de 2012. DOI: 10.17487/RFC6749. URL: https://rfc-editor.org/rfc/rfc6749.txt.

[6] *Implicit Flow with Form Post*. URL: https://auth0.com/docs/flows/implicit-flow-with-form-post (acedido em 11/06/2021).

[8] F. Martin. *GUI Architectures*. 2010. URL: https://martinfowler.com/eaaDev/uiArchs.html (acedido em 31/01/2020).

[9] Oracle. "Top 10 Reasons to Choose MySQL for Next Generation Web Applications: A MySQL Whitepaper". Em: August (2016). URL: http://www.oracle.com/us/products/mysql/mysql-wp-top10-webbased-apps-461054.pdf.

[10] C. Pautasso, O. Zimmermann e F. Leymann. "Restful Web Services vs. "Big"' Web Services: Making the Right Architectural Decision". Em: *Proceedings of the 17th International Conference on World Wide Web*. WWW '08. New York, NY, USA: Association for Computing Machinery, 2008, pp. 805–814. ISBN: 9781605580852. DOI: 10.1145/1367497.1367606. URL: https://doi.org/10.1145/1367497.1367606.

[11] *PUT - HTTP | MDN*. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT (acedido em 22/01/2020).

[12] *Resource Owner Password Flow*. URL: https://auth0.com/docs/flows/resource-owner-password-flow (acedido em 11/06/2021).

[13] M. Sargent, D. Linthicum e D. Sivaram. *What is SOAP (Simple Object Access Protocol)?* 2014. URL: https://searchapparchitecture.techtarget.com/definition/SOAP-Simple-Object-Access-Protocol (acedido em 20/01/2020).

[14] *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. URL: https://www.w3.org/TR/soap12-part1/ (acedido em 20/01/2020).

[15] J. P. Sossoloti. *Autenticação REST OAuth2 em Java com Apache Oltu - Blog da Caelum: desenvolvimento, web, mobile, UX e Scrum*. Jan. de 2017. URL: https://blog.caelum.com.br/autenticacao-rest-oauth2-em-java-com-apache-oltu/ (acedido em 10/02/2020).

[16] S. Thomas. *Oracle the Best Choice for Your Company's Success*. 2017. URL: http://socialbarrel.com/unique-database-services-that-make-oracle-the-best-choice-for-your-companys-success/109372/ (acedido em 10/02/2020).

[17] G. M. Waleed e R. B. Ahmad. "Security protection using simple object access protocol (SOAP) messages techniques". Em: *2008 International Conference on Electronic Design*. 2008, pp. 1–6. DOI: 10.1109/ICED.2008.4786714.

[18] *What is soap ? Types of Soaps*. URL: https://www.altexsoft.com/blog/engineering/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/ (acedido em 20/01/2020).

[19] *What is the View of Data in DBMS? Data Abstraction, Three-Schema Architecture, Data Independence, Instance & Schema - Binary Terms*. URL: https://binaryterms.com/view-of-data.html (acedido em 26/01/2020).

[20] *Which OAuth 2.0 Flow Should I Use?* URL: https://auth0.com/docs/api-auth/which-oauth-flow-to-use/ (acedido em 10/02/2020).

[21] C. Woodford. *How does OCR document scanning work? - Explain that Stuff*. Dez. de 2018. URL: https://www.explainthatstuff.com/how-ocr-works.html (acedido em 12/02/2020).