NOVA SCHOOL OF SCIENCE & TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE

JORGE NEVES FRESCO

BSc in Computer Science and Engineering

IMPROVING USER EXPERIENCE IN APPS GENERATED ON A LOW-CODE/NO-CODE PLATFORM WITH TABLE ACTIONS

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2025

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

IMPROVING USER EXPERIENCE IN APPS GENERATED ON A LOW-CODE/NO-CODE PLATFORM WITH TABLE ACTIONS

JORGE NEVES FRESCO

BSc in Computer Science and Engineering

Adviser: David Aveiro
Assistant Professor, Universidade da Madeira

Co-adviser: Miguel Goulão
Associate Professor, NOVA University Lisbon

Examination Committee

Chair: Doutora Teresa Isabel Lopes Romão
Associate Professor, FCT-NOVA

Rapporteur: Doutor Vasco Miguel Moreira do Amaral
Associate Professor, FCT-NOVA

Adviser: Doutor David Aveiro
Associate Professor, Universidade da Madeira

Co-adviser: Doutor Miguel Carlos Pacheco Afonso Goulão
Associate Professor, FCT-NOVA

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon
September, 2025

Improving user experience in Apps generated on a low-code/no-code platform with table actions

This document was created with the (pdf/Xe/Lua)LaTeX processor and the NOVAthesis template (v7.1.18) [1].

Acknowledgements

I want to thank my parents for their unwavering support, encouragement, and guidance throughout the years. Their belief in me and assistance in various situations were instrumental in helping me reach this milestone in my life.

I am also grateful to my colleagues who accompanied me on this journey, sharing both challenges and successes, and turning difficult days into memorable ones, making this experience all the more rewarding.

I would like to express my sincere appreciation to my thesis advisors, Professors David Aveiro and Miguel Goulão, whose invaluable guidance, advice, and support enabled me to complete this work and successfully conclude my studies at the NOVA School of Science and Technology after five years of diligent effort.

I would also like to thank Vítor Freitas for his time, patience, and willingness to help and provide guidance whenever needed.

Finally, I appreciate Professor João Lourenço for creating and freely providing the NOVAthesis LaTeX template, which was used to make this document.

Abstract

Low-code and no-code platforms have revolutionized software development by enabling individuals with minimal programming expertise, also known as citizen developers, to create functional applications through intuitive graphical interfaces and pre-built components. Despite their benefits, many of the applications generated by these platforms suffer from usability and inclusiveness challenges due to standardized, rigid interface components that do not adequately accommodate diverse cognitive and interactional needs. This research focuses on investigating ways to improve the usability and inclusiveness of applications generated by the Dynamic Information System Modeler and Executor (DISME), an open-source low-code platform (LCP) developed by the Enterprise Engineering Lab, a research group part of the Madeira Branch of NOVA Laboratory for Computer Science and Informatics (NOVA LINCS). The study was conducted by evaluating the current state of the art in terms of usability and inclusiveness of applications produced by low-code platforms, as well as through a comparative analysis between DISME and other leading market alternatives, such as OutSystems, PowerApps, and Mendix, to identify potential areas for improvement. Several functionalities and differences were identified, but the focus ultimately converged on a more specific objective: the implementation of Table Actions in applications generated by DISME. This feature, which allows interactive actions to be embedded directly into table rows, was identified as a crucial addition to improving usability of the applications generated by DISME. The implementation of table actions required substantial architectural changes, including the development of a new Action Items Management component within DISME. Due to time constraints, it was not possible to conduct usability testing with real DISME users. Instead, a comparison was carried out by sharing a Google Forms questionnaire, presenting participants with two versions of a page to manage hearings part of an application generated by DISME, one with the traditional execution method and one with table actions, alongside a step-by-step demonstration of how to reschedule a hearing requested by a citizen. The results indicated that participants found executing tasks through table actions to be more intuitive, more efficient, less prone to errors, and overall more satisfying compared to the original method. This comparison highlights the potential of table actions to reduce interaction complexity, improve task

execution efficiency, and move DISME toward a more user-centered design. In addition to this technical contribution, we also identified a set of promising improvements for both DISME and the current implementation of the Municipality Hearing Process application generated by DISME.

Keywords: Low-Code Platforms, Information Systems, Dynamic Information System Modeller and Executor, Design and Engineering Methodology for Organizations Usability, Inclusivity, User-centric Interfaces, User Experience, Gender Inclusiveness Magnifier (GenderMag), Comparative Analysis, Table Actions

Resumo

As plataformas *low-code* e *no-code* revolucionaram o desenvolvimento de *software* ao permitirem que indivíduos com pouca experiência em programação, conhecidos como *citizen developers*, criem aplicações funcionais através de interfaces gráficas intuitivas e componentes pré-construídos. Apesar dos seus benefícios, muitas aplicações geradas por estas plataformas enfrentam desafios de usabilidade e inclusão devido ao uso de componentes de interface padronizados e rígidos, que não acomodam adequadamente as diversas necessidades cognitivas e interativas dos utilizadores. Esta pesquisa tem como foco a melhoria da usabilidade e inclusividade das aplicações geradas pelo *Dynamic Information System Modeler and Executor* (DISME), uma plataforma *low-code open-source* desenvolvida pelo *Enterprise Engineering Lab*, um grupo de investigação que faz parte da Filial da Madeira da *NOVA Laboratory for Computer Science and Informatics* (NOVA LINCS). O estudo foi feito através da avaliação do atual "estado da arte" em termos de usabilidade e inclusividade das aplicações produzidas por plataformas *low-code* e também através de uma análise comparativa entre o DISME e outras alternativas líderes do mercado, como OutSystems, PowerApps e Mendix, com o objetivo de identificar potenciais áreas de melhoria. Foram identificadas várias funcionalidades e diferenças, mas o foco acabou por se centrar num objetivo mais específico: a implementação de Ações de Tabela nas aplicações geradas pelo DISME. Esta funcionalidade, que permite incorporar ações interativas diretamente nas linhas das tabelas, foi identificada como uma adição crucial para melhorar a usabilidade e exigiu alterações arquiteturais substanciais, incluindo o desenvolvimento de um novo componente de Gestão de Itens de Ação no DISME. Devido a limitações quanto ao tempo e à disponibilidade de utilizadores, não foi possível realizar testes de usabilidade com utilizadores reais do DISME. Em alternativa, foi realizada uma comparação através da partilha de um questionário no Google Forms, no qual foram apresentadas aos participantes duas versões de uma página para gestão de audiências de uma aplicação gerada pelo DISME: uma com o método tradicional de execução e outra com ações de tabela, acompanhadas de uma demonstração passo a passo de como reagendar uma audiência solicitada por um cidadão. Os resultados indicaram que os participantes consideraram a execução de tarefas através das ações de tabela mais

intuitiva, mais eficiente, menos propensa a erros e, de forma geral, mais satisfatória quando comparada com o método original. Esta comparação evidenciou o potencial das ações de tabela para reduzir a complexidade da interação, melhorar a eficiência na execução de tarefas e aproximar o DISME de um design mais centrado no utilizador. Para além deste contributo técnico, foi também identificado um conjunto de melhorias promissoras tanto para o DISME como para a implementação atual da aplicação do Processo de Audiências Municipal gerada pelo DISME.

Palavras-chave: Plataformas *Low-Code*, Sistemas de Informação, Modelador e Executor de Sistemas de Informação Dinâmicos (*DISME*), Metodologia de Design e Engenharia para Organizações (*DEMO*) Usabilidade, Inclusividade, Interfaces Centradas no Utilizador, Experiência do Utilizador (*UX*), Ampliador de Inclusão de Género (*GenderMag*), Análise Comparativa, Ações de Tabela

Contents

List of Figures

List of Tables

List of Listings

Glossary

citizen developers A non-professional programmer who creates applications using low-code or no-code platforms, enabling them to build functional software without extensive coding expertise *(pp. 5, 8, 12)*

CSS Cascading Style Sheets (CSS) is a computer language for laying out and structuring web pages (HTML and XML *(pp. 26, 29, 39)*

Drag-And-Drop When a user selects an object/component and drags it to the desired location. The object is "dropped" when the user releases the used input method *(pp. 1, 5–7, 25)*

end-user An end-user is the person who ultimately interacts with and uses a software application, system, or product to achieve a specific goal *(p. 8)*

modal Large user interface elements that sit on top of an application's main window *(pp. 22, 29)*

open-source Source code of a project, software, or platform is publicly available for anyone to view, modify, and distribute *(pp. 1, 8)*

tooltip A tooltip, also known as infotip or hint, is a common graphical user interface (GUI) element in which, when hovering over a screen element or component, a text box displays information about that element *(p. 34)*

vendor lock-in When someone is essentially forced to continue using a product or service regardless of quality because switching away from that product or service is not practical *(p. 6)*

Wizard A software wizard or setup assistant or multi-step form is a user interface that leads a user through a sequence of small steps, providing a simplified and user-friendly interface to guide the user through the process *(pp. 37, 38, 78, 80)*

Acronyms

AI Artificial Intelligence *(p. 7)*
AOM Adaptive Object Model *(p. 8)*
API Application Programming Interface *(pp. 5, 7, 45, 46, 52, 55, 58)*

BPMN Business Process Model and Notation *(p. 20)*

CRUD Create Read Update Delete *(p. 24)*

DEMO Design and Engineering Methodology for Organizations *(pp. 8, 9)*
DISME Dynamic Information System Modeler and Executor *(pp. 1–4, 8, 9, 19, 22, 26, 33, 38–45, 65, 72, 77–80)*

GenderMag Gender-Inclusiveness Magnifier *(pp. 2, 3, 8, 13, 15, 17, 19, 34, 72, 78–80)*
GUI Graphical User Interface *(p. 1)*

HCI Human-Computer Interaction *(p. 15)*
HTTP Hypertext Transfer Protocol *(pp. 44, 55)*

IDE Integrated Development Environment *(p. 7)*

JSON JavaScript Object Notation *(pp. 44, 45)*

LCP Low-Code Platform *(pp. 1–3, 6–8, 12–14, 38, 73, 78, 80)*
LCPs Low-Code Platforms *(pp. 1–6, 12, 14, 19, 22, 48, 78)*
LINCS Laboratory for Computer Science and Informatics *(p. 8)*

MHP Municipality Hearing Process *(pp. 2, 19, 20, 26, 33, 37, 39, 44, 64, 67, 68, 78–80)*

NOVA LINCS NOVA Laboratory for Computer Science and Informatics *(p. 1)*

REST Representational State Transfer *(p. 44)*

SQL Structured Query Language *(pp. 7, 10)*

UI User Interface *(pp. 1, 5, 6, 13, 21–23, 26)*

UX User Experience *(pp. 2, 3, 12, 13, 18, 22, 37, 39, 46, 78)*

1 Introduction

1.1 Description

Low-Code Platforms (LCPs) have emerged as transformative tools in software development, enabling individuals with minimal or no programming expertise to build functional applications [2–4]. These platforms simplify and accelerate development, usually by providing intuitive graphical interfaces, pre-built components, and Drag-And-Drop functionality [4, 5].

As a result, LCPs have gained significant adoption across various industries, contributing to the democratization of software development by empowering non-developers and reducing reliance on highly skilled programmers [2, 4].

Despite their benefits, many LCPs-generated applications may face usability and inclusivity challenges, such as the lack of customization options [6–8], lack of collaboration tools [4], a high learning curve [4, 8], and challenges in User Interface (UI) testing [7].

A significant limitation of these platforms is their reliance on standardized, pre-built components, often resulting in rigid user interfaces that follow a "one-size-fits-all" approach.

This lack of adaptability can negatively impact the end-user experience, particularly for individuals with diverse cognitive and interactional needs.

This dissertation focuses on investigating how to enhance the usability and inclusiveness of the Graphical User Interface (GUI) generated by the Dynamic Information System Modeler and Executor (DISME) platform [9], an open-source Low-Code Platform (LCP) developed by research fellows and master's students at the Enterprise Engineering Lab[1], part of the Madeira branch of NOVA Laboratory for Computer Science and Informatics (NOVA LINCS)[2]. The study began with an evaluation of the state of the art in terms of usability and inclusiveness of applications produced by low-code platforms, followed by a comparative analysis between DISME and market-leading alternatives, such as OutSystems, PowerApps, and Mendix. Building on these findings, the work concluded with

[1] https://eelab.arditi.pt/
[2] https://nova-lincs.di.fct.unl.pt/

the implementation of a new feature in DISME: *Table Actions*, which introduces interactive operations directly within table rows as a way to improve the User Experience (UX) potentially.

To gauge how users perceive this feature, a questionnaire was distributed to a sample of participants comprising university peers, friends, and colleagues. Respondents were then asked to evaluate aspects such as clarity of use, task completion speed, likelihood of errors, and overall satisfaction for each version of a page to manage hearings part of the Municipality Hearing Process (MHP) application generated on DISME. Feedback consistently indicated that participants favored the table actions approach, finding it more intuitive, efficient, and reliable, while also providing a more satisfying user experience.

1.2 Motivation

While LCPs enable rapid application development, the applications they generate frequently lack flexibility in their user interfaces, which can lead to suboptimal usability and inclusivity [10]. The Gender-Inclusiveness Magnifier (GenderMag) framework[3], which emphasizes the importance of designing for cognitive diversity, including gender-inclusive design, underscores the need for more adaptable and user-centered approaches in software development [11].

DISME aims to address usability and inclusivity challenges by applying the GenderMag framework and incorporating usability-focused enhancements to ensure that the applications it generates meet higher standards of usability and inclusivity

One of the most significant concerns in DISME applications is related to how users execute actions, often on data contained in tables. Currently, actions are implemented as card widgets on each page, each containing information about the action and a clickable button that opens one or more consecutive forms. As the number of available actions increases, additional card widgets are added to the page, possibly resulting in a more crowded interface.

These inefficiencies can impact the UX by making task execution slower, less intuitive, and more frustrating for the user. A highly requested feature and present in most LCPs is the ability to add actions to tables to execute tasks, thereby skipping the process of selecting specific information, which can potentially make task execution faster, more intuitive, and simplify the interfaces of generated applications.

Besides this, existing research on the topic of this thesis primarily focuses on the developer experience when using LCPs [12], with limited attention given to the usability and none to the inclusiveness of the applications produced. The reason for this might be the belief that a better developer experience can lead to a better user experience [13–16]. This gap presents an opportunity to explore how applications generated by popular LCPs align with established usability frameworks, and ultimately, to utilize those findings to

[3] https://gendermag.org/

identify possible ways to improve DISME further and generate applications that provide a better UX.

1.3 Objectives

The central question of this research is: **Can table actions improve the usability and inclusiveness of applications generated by DISME?** Addressing this question requires a structured approach that involves analyzing existing research on usability in LCP-generated applications, particularly in relation to established usability standards and frameworks such as GenderMag. This provided us a foundation for evaluating how DISME and the applications it generates align with these principles, identifying more potential usability limitations and areas for improvement. To further explore this question and, at the same time, assert how other LCPs support table actions in their generated applications, the research conducted a comparative analysis with leading alternatives, such as OutSystems, PowerApps, and Mendix. This analysis helped identify the possible strengths and limitations of these platforms, in terms of usability as well as inclusivity, and allow us to explore different options for implementing actions embedded within tables.

Building on the analysis of existing research on this topic and the comparative study between DISME and other popular LCPs, this work aimed to establish a solid foundation for identifying opportunities to enhance the usability and inclusiveness of applications generated by DISME. By implementing table actions, we provided a more efficient way to perform operations in DISME-generated applications, reducing the number of steps required and making table interactions more intuitive.

Finally, to complement the technical implementation and evaluate its impact, we conducted a questionnaire-based assessment of the table actions feature. This evaluation involved presenting participants with two versions of the same application, one without table actions and the other with table actions. The steps required to execute a specific task were presented, and feedback was collected on perceived intuitiveness, efficiency, error likelihood, and overall user satisfaction. By doing so, the research aimed to grasp the potential impact of table actions on usability and user experience, despite time constraints that didn't allow us to conduct a comprehensive usability test with active DISME users interacting with the application itself.

1.4 Document structure

This chapter provided an introduction to the work. The remainder of the document is structured as follows:

Chapter 2 - State of Art: This chapter presents the fundamental topics for this thesis, providing an overview of LCPs, the DISME platform, usability principles, the GenderMag

framework, and the comparative analysis of DISME against other LCPs, highlighting key findings that will inform the proposed enhancements.

Chapter 3 - Requirements This chapter presents a list of functional and non-functional requirements that guided the development of our project. It also includes a set of user stories that guided the implementation of table actions.

Chapter 4 - Implementation of Table Actions: This chapter describes the implementation of table actions, including the creation of the new component and modifications to existing components.

Chapter 5 - Evaluation: This chapter presents the evaluation process of the table actions feature in DISME. It first illustrates the step-by-step process for creating table actions and adding them to applications, followed by a comparison of operation execution with and without table actions. The chapter then outlines the questionnaire-based evaluation to assess the potential impact of table actions on usability and user experience.

Chapter 6 - Conclusion: This chapter summarizes the main contributions of the work and discusses potential directions for future research.

2

State of Art

2.1 Low-Code Platforms

Unlike traditional development methods, which require extensive manual coding and in-depth programming expertise, Low-code is a software development approach that enables developers to create applications with minimal hand coding by leveraging methods with minimal hand coding, such as visual programming environments, to streamline the development process [3].

These environments typically utilize an interactive User Interface (UI) with Drag-And-Drop components, visual modeling, and pre-configured templates, enabling developers and non-technical users to build applications efficiently.

The emergence of Low-Code Platforms (LCPs) aligns with the broader shift toward democratizing software development [17], allowing business users, often referred to as citizen developers, to create and modify applications without requiring significant programming skills [3, 4, 10].

The development process in LCPs can generally be divided into five main steps [3]:

- **Data modeling**: Creating a visual representation of the system's data structure, defining data types, abstraction levels, and relationships.
- **User interface definition**: Designing the application's views, including pages, forms, and interactive elements that define the overall user experience.
- **Specification of business logic rules and workflows**: Defining the decision-making processes that govern application behavior.
- **Integration of external services**: Ensuring seamless connectivity with Application Programming Interface (API)s and third-party services.
- **Deployment**: Releasing the application for real-world use in various environments.

2.1.1 Advantages and challenges

LCPs offer numerous benefits that make them a compelling choice for modern software development. One of the most significant advantages is their low technical skill requirement, which allows individuals without extensive coding expertise to create functional applications [2–4].

Additionally, the high development speed of LCPs enables organizations to build and deploy applications rapidly, reducing time-to-market [4, 5, 7].

Cost efficiency is another significant benefit, as low-code development reduces the need for large development teams and extensive coding efforts, leading to lower operational expenses [3–5].

However, LCPs also present several challenges.

One of the most significant concerns is interoperability, as applications developed within a specific Low-Code Platform (LCP) may face difficulties integrating with other platforms or software ecosystems [4, 5].

Scalability can also be a challenge, as some LCPs may not be optimized for handling large-scale enterprise applications [4, 8].

Furthermore, there is the risk of vendor lock-in, where organizations become dependent on a specific low-code vendor, making future transitions difficult and costly [4, 8, 18].

Learnability has also been identified as a concern in some platforms, as they may present a high learning curve and little to no teaching material [3, 8]. This can result in users spending more time understanding how to use the platform, rather than focusing on actual development, which may reduce efficiency and hinder adoption.

LCPs often provide pre-built components; however, it is challenging to cover all possible scenarios and contexts solely with these components, leading LCPs to face criticism for their lack of customization and flexibility in UI design [6, 7, 10, 19].

To address this, most LCPs allow developers, especially those with programming skills, to implement customizations for more complex use cases, through manual coding [20–22].

While this solves the problem, [18] has highlighted that it can lead to applications containing thousands of lines of code, ultimately increasing their complexity, if the platform is not designed to support a modular design and the reality that applications will often require substantial amounts of code.

2.1.2 Most popular platforms

2.1.2.1 OutSystems

OutSystems [23] is a leading LCP designed to simplify the creation of enterprise-grade web, mobile, and progressive web applications. The platform provides a visual development environment that enables developers to design the UI using pre-built components and Drag-And-Drop functionality, as well as to define both client- and server-side business

logic through action flows. Additionally, OutSystems includes tools for visually creating and managing databases, eliminating the need to write Structured Query Language (SQL) code manually. In recent years, OutSystems has been infusing Artificial Intelligence (AI) across its platform and the software development life cycle to assist developers in creating higher-quality software faster while also empowering them to build AI-powered applications with ease. The platform is continuously evolving, with a future project to accelerate the development process even further by integrating generative AI features, bringing together the transformative potential of AI and low-code technologies [24].

2.1.2.2 Power Apps

PowerApps [25] is a low-code application development platform from Microsoft designed to empower organizations to build custom business applications with minimal coding effort and that can run on web browsers, mobile devices, and tablets. Power Apps offers an intuitive visual Drag-And-Drop interface, pre-built templates, suggested controls, pre-configured automation and seamless integration capabilities with other Microsoft services, such as Microsoft 365[1], Microsoft Azure[2], Dynamics 365[3], as well as a wide list of third-party services through prebuilt connectors and APIs. To further accelerate app development, Power Apps incorporates AI-powered features, such as Copilot, to assist users in designing applications, automating tasks, and generating working apps directly from natural language inputs or even design assets such as Figma[4] files, as well as the AI Builder [26], which enables users to integrate AI functionalities like form processing, object detection, and prediction models without requiring extensive machine learning expertise. Additionally, the platform supports the use of custom code via Power Fx [27], a declarative, Excel-like formula language, for more complex use cases.

2.1.2.3 Mendix

Mendix [28] is a robust LCP, based on the principles of model-driven development. Mendix enables enterprises to create web and mobile applications with minimal hand-coding and in a single Integrated Development Environment (IDE). The platform provides a visual development environment where users can build user interfaces using Drag-And-Drop components, define workflows and logic through graphical models, and manage data with built-in database tools. Some of the platform's main features include the support for more advanced customization through Java and JavaScript coding[20], as well as through some JavaScript frameworks (React and React Native), support for collaboration with other developers using Git-based version control, a AI-assisted development tool that includes three specialized real-time recommendations for logic, best practices, and workflow [29].

[1] https://www.office.com/
[2] https://azure.microsoft.com/
[3] https://www.microsoft.com/en-us/dynamics-365
[4] https://www.figma.com/

2.2 DISME

2.2.1 Introduction

The Dynamic Information System Modeler and Executor (DISME) is a Design and Engineering Methodology for Organizations (DEMO)-based open-source LCP being developed by a team consisting of research fellows and master's students integrated into the Enterprise Engineering Lab, part of the Madeira branch of NOVA Laboratory for Computer Science and Informatics (LINCS)[5] [9]. Additionally, NOVA students have also contributed to its development, further enhancing its evolution and implementation. The platform is built using various technologies and frameworks, including PHP (Laravel)[6], Typescript (Angular)[7], and Bootstrap [8].

DISME focuses on creating model-driven applications for information systems by enabling users to model and execute systems with minimal coding expertise and, unlike other LCPs that generate code for static representations of organizational processes, the DISME platform treats organizations as dynamic, living systems by leveraging the principles of Adaptive Object Model (AOM) [9, 30].

Another aspect is that DISME places a strong emphasis on usability and inclusiveness, to develop applications that are user-friendly and adaptable to diverse user needs. The platform is designed for both citizen developers and end-user, with a focus on bridging the gap between technical and non-technical users. It employs usability best practices and interfaces that can be tailored to different personas, leveraging methods like Gender-Inclusiveness Magnifier (GenderMag)[9] to ensure inclusivity and broader adoption in organizations. DISME also stands out for its alignment with both academic research and practical enterprise needs, offering a robust tool and unique features for digital transformation in information system development.

2.2.2 DEMO

DEMO is an Enterprise Engineering method rooted in a comprehensive body of theories that aim to address software failure challenges that are often responsible for projects that fail to meet end customers' initial expectations. These challenges include unrealistic project objectives, incomplete or inconsistent requirements, lack of stakeholder and user involvement, poor project management and control, ineffective communication, and reliance on new technologies for which developers lack experience. These issues make software and information systems inflexible, making it difficult to adapt to changing requirements, regulations, and other evolving conditions [10].

[5] https://nova-lincs.di.fct.unl.pt/
[6] PHP: https://www.php.net/, Laravel: https://laravel.com/
[7] Typescript: https://www.typescriptlang.org/, Angular: https://angular.dev/
[8] Bootstrap: https://getbootstrap.com/
[9] https://gendermag.org/

These challenges often make software and information systems inflexible and challenging to modify, as it requires a lot of work to keep up with continuously changing requirements, laws, etc [10].

DEMO models organizational commitments and activities through four key perspectives. The **Cooperation Model** focuses on coordinating roles and transactions. The **Process Model** outlines workflow execution and interaction. The **Fact Model** defines the facts and rules of the system, while the **Action Model** specifies operational regulations that guide the execution of tasks. Despite its importance as a foundational component, the Action Model is rarely applied in projects and is often overlooked in practice.

2.2.3 Architecture

DISME architecture can be divided into two main parts: the **System Modeller**, which deals with the specification of the system through different diagrams, forms, and tables; and the **System Executor**, which handles the daily execution of processes and information flow, according to the specifications done in the System Modeller and the current state of the live information system of the enterprise running it [5].

2.2.3.1 System Modeler

The System Modeler is the "server-side" of the platform and is used to design and describe the system. Users who model the system require a basic understanding of enterprise engineering modeling, without needing any programming-specific knowledge. The system modeler can be divided into five distinct software components:

- **Diagram Editor**: Allows users to create and maintain organization diagrams using the DEMO methodology and integrates with draw.io[10]. Diagram transformations are automatically reflected in the database [3, 5].
- **System Specifier**: Enables users, even without any specific programming skills, to add details to system processes, including users, roles, transactions, and properties, using a table/forms-based approach [3, 9].
- **Action Rules Management**: Handles specific events through business rules. It uses the Blockly library[11]. This open-source visual programming tool supports various input types (labels, drop-downs, checkboxes, text/number fields) and offers syntax validation to ensure correct block connections and configurations (Figure 2.1).
- **Forms Management**: Integrates with Form.io[12], a low-code, open-source platform designed for form-based application development, to develop necessary forms and templates, determining the types of fields users need to fill (Figures 2.2 and 2.3).

[10] https://github.com/jgraph/drawio
[11] https://developers.google.com/blockly
[12] https://form.io/

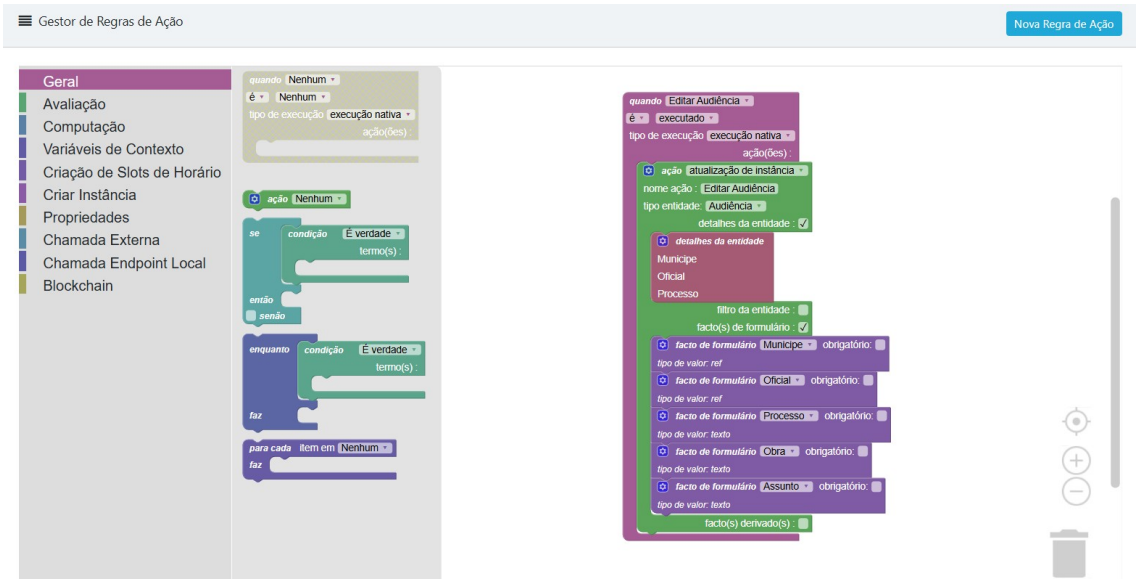

Figure 2.1: Action Rule for editing a hearing in DISME

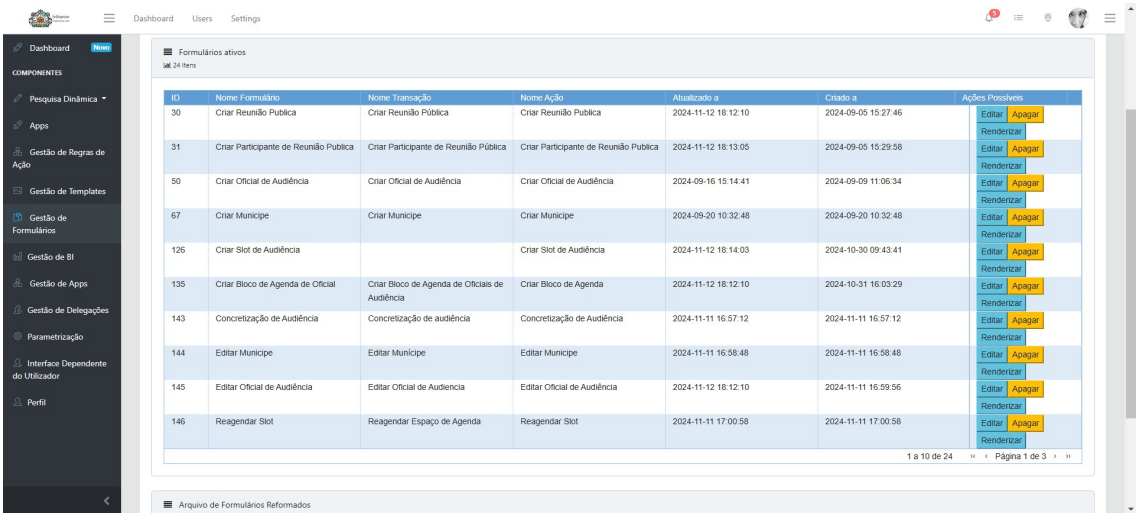

Figure 2.2: DISME's Forms Management page displaying active forms

- **Dynamic Query Management**: A "query-based" mechanism designed to help less-experienced users obtain the desired results without using SQL. The users start by selecting an entity type, then the relevant properties, and finally the desired filters to be applied. The query results are then presented in a table [3, 5].

2.2.3.2 System Executor

System Executor is the "client-side" of the platform that executes and displays the designs previously created in the System Modeller[3], while also depending on *"the current state of the live information system of the enterprise running it"* [9]. It is made of the **Dashboard**

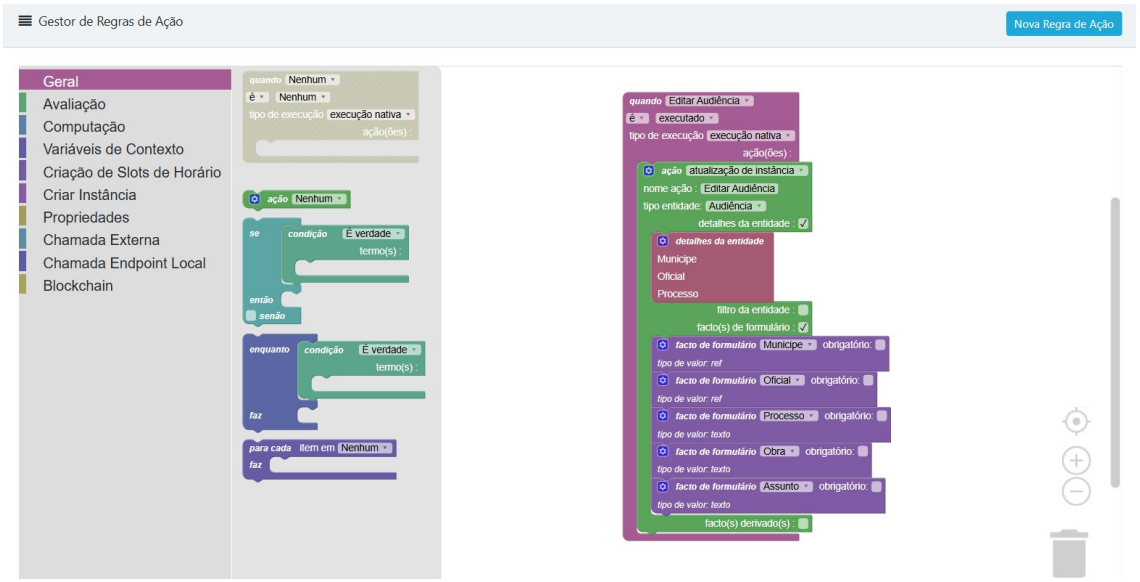

Figure 2.3: Form management of the one used to create new citizens, in DISME

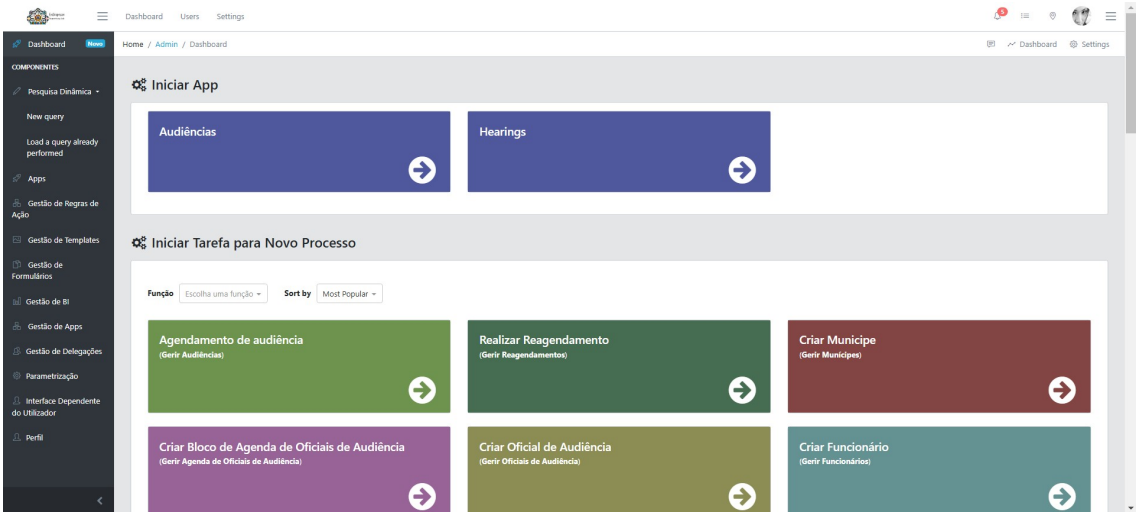

Figure 2.4: DISME Dashboard Component

(Figure 2.4), which is the first thing users see upon login, displaying operations they have access to and can execute based on their roles. The Dashboard also allows users to initiate processes, perform tasks, and manage pending, completed, and delegated tasks. The **Execution Engine** is the second component of the System Executor and is triggered by interactions on the Dashboard or certain foreseen events in the system specification [9].

2.3 Usability

2.3.1 Definition

To introduce the concept of usability, it is essential to define it as *"the extent to which a system, product, or service can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use"* [31]

These three qualities are defined as follows:

- **Effectiveness**: The accuracy and completeness with which users achieve the specified goals.
- **Efficiency**: The resources expended, typically time, human effort, costs, and materials, considering the results achieved.
- **Satisfaction**: The degree to which the user experience that results from actual use meets the needs and expectations of the user.

According to Nielsen Norman Group, usability is the second level in the hierarchy User Experience (UX), placed after utility and before desirability [32]. This means that after determining whether a product can meet user needs (utility), designers must focus on making it usable before considering emotional aspects, such as desirability.

For the end-user, software usability is essential because it is a key determinant of performance, enabling users to complete tasks faster and more efficiently. For managers, usability is a major decision point in selecting a product, as it influences the learnability of the system and the productivity of those who use it [33].

2.3.2 Usability in applications generated by LCPs

A key consideration in this context is the distinction between the *citizen developer* and the *end-user*. Citizen developers refer to individuals, usually with limited programming experience, using the LCP to create new applications. At the same time, end-users are the individuals who use or will use those applications.

Most current studies focus primarily on the developer experience [6, 34–38] using these platforms, as it is often assumed that an improved experience for designers and developers will result in a better quality result and, therefore, a better experience for end-users [7, 18, 39]. The focus on a clean and simple interface for developers can translate to more user-friendly applications, especially for internal tools and business applications, as it enables developers to test and deliver a higher-quality product more easily.

This focus on the developer experience may lead these platforms to overlook specific usability needs of end-users that citizen developers or platforms may not adequately address. Evaluating the UX of applications generated on LCPs is essential to ensure that these tools not only cater to developers but also create genuinely accessible, efficient, and satisfying experiences for all users.

2.3.2.1 Lack of UI Customization and Testing

Despite the many advantages of LCP, they often face criticism for their lack of customization and flexibility in UI design [6, 7, 19]. Studies have highlighted that while these platforms facilitate quick prototyping and basic application development, some of them impose constraints on more complex or highly customized projects [8, 40].

To address this issue, these platforms often allow developers to write custom code for complex use cases [20–22] or support direct transformation from design tools like Figma or Sketch into development models, as seen in platforms like OutSystems [41].

Limitations in customization can directly impact the usability of the applications generated by these platforms. Without the ability to tailor the interface and interaction mechanisms in detail, designers may struggle to create user experiences that cater to a wide range of end-users with varying needs, skills, and preferences. A literature review [7] found that a significant number of bugs (16.5%) in LCPs relate to incorrect graphics in the user interfaces, such as incorrect size, position, visibility, or display strategies in the UI, potentially impacting the usability of the generated applications. The fact that UI bugs are common highlights the need for UI-based testing approaches to automatically detect UI bugs to avoid issues that could negatively affect the UX.

The inherent design and testing limitations of many LCPs may hinder the ability to address gender inclusivity concerns comprehensively. For example, the lack of support for different information processing styles and the limited options for personalization may reduce the platforms' capacity to accommodate the diverse personas outlined in GenderMag [42–44].

2.3.2.2 Accessibility

Accessible interfaces ensure that users of all abilities, including those with disabilities, can effectively interact with digital systems. In some contexts, users may not have access or are unable to use a mouse and/or a keyboard, and interfaces need to be adapted to accommodate these situations to ensure accessibility.

For users with visual impairments, a standard solution is the use of screen readers or text-to-speech technologies, which translate on-screen information into speech or Braille [45]. For users with motor impairments or those who cannot use a mouse, interfaces must support full keyboard navigation by allowing navigation using the Tab key to move between elements and the Space/Enter key to interact with them, providing visible focus indicators to highlight selected elements, and designing interactive components to be operable without a mouse [46].

Another solution found was the use of adaptive and personalized interfaces, such as the Supple system [47], which offers foundational insights into generating user-centered interfaces dynamically.

The Supple system is an adaptive interface generation tool designed to enhance usability by tailoring interfaces to users' needs, device constraints, and interaction preferences.

Instead of relying on static designs, Supple dynamically adjusts interfaces through an optimization-based approach, ensuring accessibility and ease of use.

Supple also supports user-driven customization, allowing users, both designers and end-users, to modify interface elements according to their preferences [47]. For instance, a user might replace a drop-down menu with a slider for easier access. Supple records these customizations and ensures they are preserved and reapplied, even when the interface is displayed on a different device. This feature enhances user experience by maintaining consistency across devices while adapting to individual needs.

A key strength of Supple is its user-driven customization, which allows individuals to modify interface elements according to their preferences. For instance, a user might replace a drop-down menu with a slider for easier access. Once a user customizes an interface, Supple retains and applies these adjustments across different devices, maintaining consistency while adapting to individual needs. This ensures a more inclusive experience, particularly for users with diverse motor abilities, cognitive styles, or interaction preferences.

In the context of low-code/no-code platforms, which often generate standardized, static interfaces, Supple's adaptive approach could serve as a model for improving personalization and accessibility. By integrating similar customization mechanisms, these platforms could better accommodate users with different usability needs, including those with disabilities or specific interaction styles.

2.3.2.3 Mobile Applications

Mobile applications offer users seamless access to digital systems on the go, making usability a crucial factor in their design. Unlike desktop software, mobile apps must accommodate unique constraints such as screen size, connectivity, power consumption, processing capabilities, and diverse input modalities. Additionally, users often interact with mobile applications in dynamic environments, such as while walking, multitasking, or in varying lighting conditions, making factors like readability, touch precision, and response time even more crucial for an intuitive user experience. [48]

Some LCPs support the creation of mobile applications, but ensuring usability and inclusivity in these auto-generated interfaces remains a challenge. A study [49] analyzing Task Manager, a sample application developed in OutSystems, identified usability limitations specific to LCPs-generated apps. These platforms often prioritize speed and cost efficiency over customization and flexibility, which can impact user experience.

To better understand user perceptions, researchers conducted a literature review, interviews, and focus groups, leading to the proposal of usability guidelines tailored to LCDP applications. These align with the ISO 9241-11 usability standard [31], emphasizing effectiveness, efficiency, and satisfaction.

Overall, users found the mobile app easy to use and learn, but particular usability challenges were noted. The task deletion process was not intuitive due to the multiple

navigation steps, and users recommended clearer action indicators, such as color-coded headers and buttons, to improve navigation. Additional concerns included unclear instructions for accessing certain features and inconsistencies in animations and transitions, which, while not affecting core functionality, impacted perceived usability.

2.4 GenderMag

Individual differences in the way people use software often vary by gender, particularly in problem-solving tasks [50, 51].

A large study [52] supporting this claim investigated gender differences in software interaction, aiming to understand how cognitive, behavioral, and motivational variations between men and women influence the use of digital systems. The research draws on a comprehensive conceptual review of prior studies across multiple scientific domains, combined with empirical work focused on Human-Computer Interaction (HCI) and technology usage. Its goal was to identify patterns that could inform more inclusive software design.

The methodology involved a systematic review of existing literature in five steps: targeted searches in the ACM Digital Library; expansion to Google Scholar to include studies linking gender and technology design; reference mining; informal web searches for additional sources; and updates with recent statistics in areas such as social media, e-commerce, and mobile applications.

The findings reveal meaningful gender-related differences in cognition, behavior, and technology use. Women tend to process information more comprehensively, exhibit higher risk aversion, and are more motivated by the practical utility of technology. In contrast, men often demonstrate more selective processing, higher self-efficacy, and intrinsic motivation. Usage patterns on digital platforms also differ: women tend to engage more with social media and productivity applications, while men tend to dominate gaming, business, and navigation applications.

Many software designs unintentionally prioritize problem-solving strategies that align more closely with those typically used by men, overlooking the diversity of cognitive styles across genders [50].

GenderMag[13] is an inspection method designed to evaluate problem-solving software from a gender-inclusiveness perspective [50]. The technique helps developers recognize gender-inclusiveness issues, understand the reasons behind these issues, and determine what changes are necessary. Designing software to be more gender-inclusive can benefit all users, regardless of gender [50].

GenderMag is structured around three core components that collaboratively address inclusivity issues in technology design. The first component is **Cognitive Facets**, which represents individual differences in cognitive styles and problem-solving approaches that

[13] https://gendermag.org/

have been extensively investigated in the literature. The second is **Personas**, which are a set of user archetypes that encapsulate the various cognitive facets to bring them to life. Finally, the **Cognitive Walkthrough** provides a systematic approach to assess how well a system supports users with varying cognitive styles, combining both the Cognitive Facets and Personas, ensuring a thorough and inclusive analysis.

2.4.1 Cognitive Facets

At the core of the GenderMag method is a set of five cognitive styles that influence how individuals interact with technology. These styles, referred to as facets, have been shown to statistically vary across genders [50]. These facets are:

- **Motivation** - User motivations for using technology and the results they expect to obtain. Men are more likely to use technology based on enjoyment and ease of interaction. At the same time, women tend to be more objective and task-oriented, prioritizing functionality and efficiency to accomplish their goals [11].

- **Information processing styles** – How users understand and process information before trying to accomplish a specific goal or solve a problem. Women are more likely to employ a comprehensive approach, gathering a thorough understanding of the problem before proceeding. In contrast, men may use selective information processing, following the first promising information and potentially backtracking if needed [50].

- **Computer self-efficacy** – Defines how confident a user is in succeeding at a specific task. It affects the features users choose to use and how willing they are to persist with features that are difficult to use. Women, on average, exhibit lower confidence in using unfamiliar technology compared to men [11].

- **Risk aversion** – Indicates how likely users are to interact with available features they might not be familiar with or are not sure of the outcome of such interaction [53]. Females tend to be more risk-averse than males in various decision-making domains [11].

- **Tinkering** – Indicates how users interact with new features and learn about them. Females are statistically less likely to playfully experiment ("tinker") with new features compared to males. However, when females do tinker, they may reflect more in the process and sometimes benefit from it more than males do, suggesting that males tend to tinker excessively, becoming less effective [11].

These cognitive facets reveal key differences in how users approach problem-solving, interaction, and learning in software environments, underscoring the importance of designing inclusively [11, 54, 55].

2.4.2 Personas

To make the cognitive facets more relatable and actionable, GenderMag brings them to life through a set of fictional personas. All of them have equivalent backgrounds, responsibilities, math skills, domain knowledge, and skills with the technology that they use regularly. Still, each one embodies specific values for each of the five cognitive facets [11]. To make these personas more relatable in a given context, they can be customized within certain bounds: the difference must be empirically supported, must not suggest a difference in intelligence or education and must align with that persona's facet values or skill level [11].

Personas are used to communicate user needs during the design phases of software development, such as through ideation and role-playing during informal tests of prototypes.

Empirical studies [11] have identified four key personas that reflect gender-related cognitive patterns and help in understanding diverse user behaviors in technology design.

The first persona, Abby, represents patterns more commonly associated with women [11]. Abby tends to have lower motivation to explore new technologies, preferring to use them for practical and task-oriented purposes. She is typically more risk-averse and exhibits lower computer self-efficacy, meaning she is less likely to experiment with unfamiliar features and requires confidence in her abilities before engaging with complex software tasks.

The second persona, Tim, reflects patterns often associated with men [11]. Tim is highly motivated by new technology and enjoys exploring and experimenting with software. He is less risk-averse and has higher computer self-efficacy, making him comfortable with trial and error when learning new tools or interacting with unfamiliar systems.

The third and fourth personas, Pats (Patricia as female and Patrick as male), represent a blend of characteristics that transcend strict gender associations [11]. The Pats embody traits seen across both men and women, serving as a reminder that cognitive styles are not rigidly tied to gender identity [56]. They exhibit neutral or mixed responses in areas such as technological motivation, self-efficacy, and risk tolerance, offering a broader representation of user behaviors.

2.4.3 Cognitive Walkthrough

The final component of GenderMag is an analytical evaluation method that focuses specifically on ease of learning and supports the systematic evaluation of how a first-time user would carry out a task by using interface features [11]. In a Cognitive Walkthrough, a team of evaluators "walks through" the interface step by step, evaluating the interface's usability and learnability at each step, in the sequence the selected persona would do when completing some particular task for the first time [56].

This walkthrough is done in two phases: In the preparatory phase, the team of evaluators describes the target user, the evaluation task, and an "ideal" sequence of goals,

sub-goals, and respective actions to achieve the task. In the analysis phase, they use a prototype of the system to systematically work through the "ideal" sub-goal sequence as if they were the target user, using a set of questions (acting as prompts) to structure their evaluation and uncover possible usability or learnability issues. For each sub-goal step, evaluators ask whether users will have formed this sub-goal as a step to achieving their overall goal. For each of the action steps, the evaluators will ask if the persona will notice that the action is available to them, associate the intended effect with the action, and understand that they have made progress towards completing the task [11].

Negative answers to these questions indicate the presence of potential issues that may affect usability and learnability, potentially leading users to struggle with reaching the overall goal effectively [11].

By systematically applying these personas to real-world software tasks, the walk-through can identify areas where the design may favor one gender's problem-solving or learning style, ultimately leading to more inclusive and user-friendly solutions. [11]

2.4.4 Related work

The GenderMag method has been validated as a practical approach for identifying and addressing gender-related usability issues in software. Empirical studies demonstrate its ability not only to uncover usability problems but also to categorize them according to cognitive facets, providing actionable insights for improving inclusiveness.

Burnett et al. [11] conducted a field study in which four software teams evaluated 99 actions and sub-goals in their applications. Across all teams, issues were identified in 55 of the 99 actions (55%), with a significant portion representing gender inclusiveness problems. By leveraging the GenderMag facets—such as motivation, risk aversion, and learning style—teams were able to classify issues more precisely than with standard cognitive walkthroughs, highlighting both the presence and the type of gender-related barriers.

In a complementary study, Burnett et al. [50] compared the GenderMag Cognitive Walkthrough method with a standard Cognitive Walkthrough approach across ten UX practitioners. The results showed that while both methods could uncover usability issues, GenderMag was more effective at specifying the types of errors found, providing richer insights into how diverse users might interact with the software.

The InclusiveMag generalization [44] extended the GenderMag approach to a broader range of diversity dimensions, including age, socio-economic status, and education. In a multi-case study involving eight teams, InclusiveMag guided the creation of personas for underserved and mainstream populations. Teams applied these personas in prototype evaluations, identifying usability and inclusivity issues that were subsequently addressed through heuristic adjustments and iterative testing. This study illustrates how GenderMag's principles can be generalized to foster inclusivity beyond gender, while maintaining a structured, facet-driven evaluation method.

A quasi-experiment with 180 participants investigated the impact of GenderMag facets on iStar 2.0 tasks [57]. Participants were characterized by motivation, information processing style, computer self-efficacy, attitude towards risk, and learning style. Results showed that participants with facet values more frequent in women (comprehensive processing, cautious risk attitude) worked more slowly but more accurately. In contrast, those with facet values more common in men (selective processing) were faster and more efficient. Differences were also observed in mental effort and stress, supporting that individual differences in software usage vary by gender.

Overall, these studies demonstrate that GenderMag is not only a diagnostic tool but a method that guides concrete improvements in software design. By revealing differences in user interaction patterns and enabling developers to make data-informed design decisions, GenderMag helps create more inclusive, practical, and user-centered applications. Its relevance is further amplified through its generalization via InclusiveMag, showing the potential to improve usability for diverse populations and cognitive styles, making it a key tool in designing equitable digital experiences.

2.5 Comparative analysis

2.5.1 Context

To evaluate the capabilities, strengths, and weaknesses of current LCPs and identify potential improvements, we conducted a systematic comparative analysis by exploring and implementing an example application based on a real-world scenario in three popular LCPs: OutSystems, PowerApps, and Mendix. This allowed us to get a broader understanding of usability and inclusivity challenges, as well as design trade-offs across different platforms, which helped us gather insights to inspire new enhancements that could be applied to DISME to improve generated applications' usability, and a better design that addresses the diverse needs of users, considering the GenderMag framework.

Furthermore, this analysis enables us to observe how these LCPs support citizen developers in adding actions directly into tables, highlighting the variety of approaches, limitations, and customization options available. By comparing these capabilities, we can identify best practices for enabling intuitive, flexible, and inclusive interactions within applications.

The real-world scenario, on which this evaluation was carried out, is the Municipality Hearing System, with a version currently implemented through DISME [58]. The system is designed to enhance citizen engagement with municipal authorities by facilitating structured and transparent hearings. The focus of the evaluation was the Municipality Hearing Process (MHP), the system that handles the scheduling and management of hearings.

This process involves several steps, but before hearings can take place, Hearing Officers must first have their schedules defined, typically on a specific, recurring day of the week.

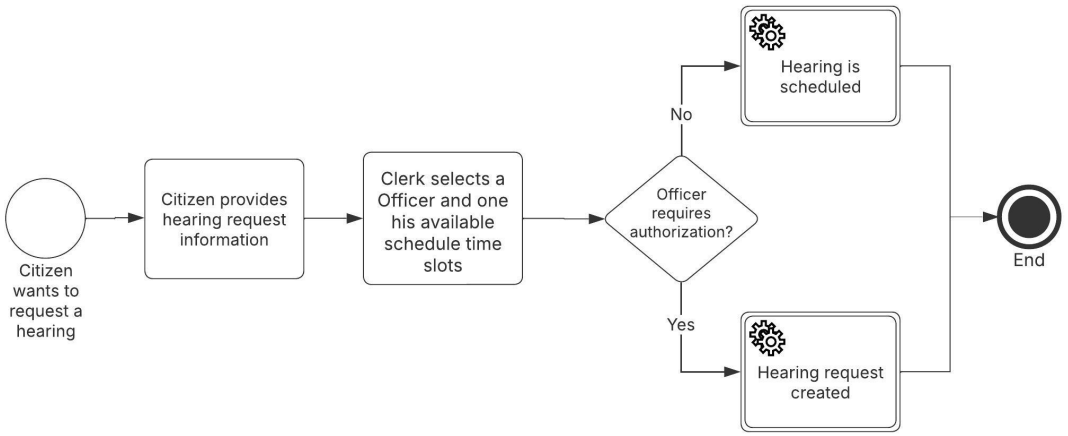

Figure 2.5: BPMN for requesting a hearing

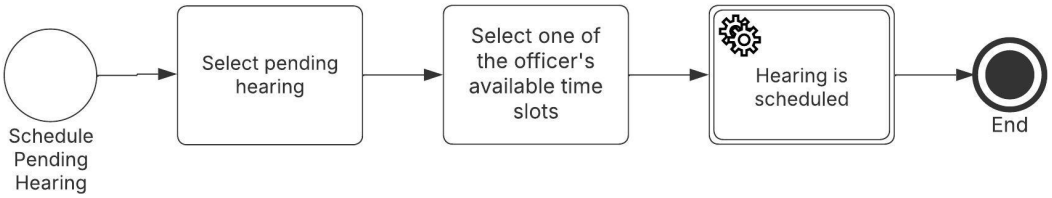

Figure 2.6: BPMN for scheduling a pending hearing

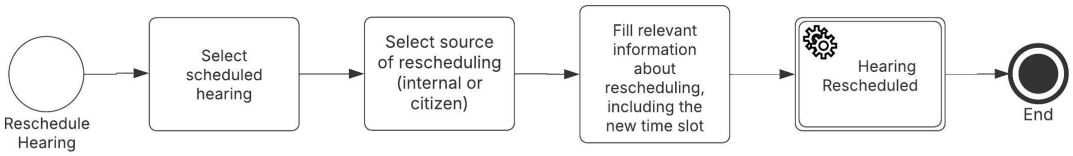

Figure 2.7: BPMN for rescheduling a hearing

Once their schedules are set, they become available for Hearing Requests.

A Hearing Request is initiated by a citizen who visits the municipality's service desk and provides all necessary information, including personal identification, process details, the subject of the request, and any relevant observations.

The next step involves identifying the most appropriate Hearing Officer to address the citizen's claim. Once the relevant officer is selected, a hearing can be scheduled.

Between the request submission and the scheduled hearing, both the citizen and the Hearing Officer have the option to request a cancellation or rescheduling.

Figures 2.5, 2.6 and 2.7 show, in a Business Process Model and Notation (BPMN), the three relevant MHP actions.

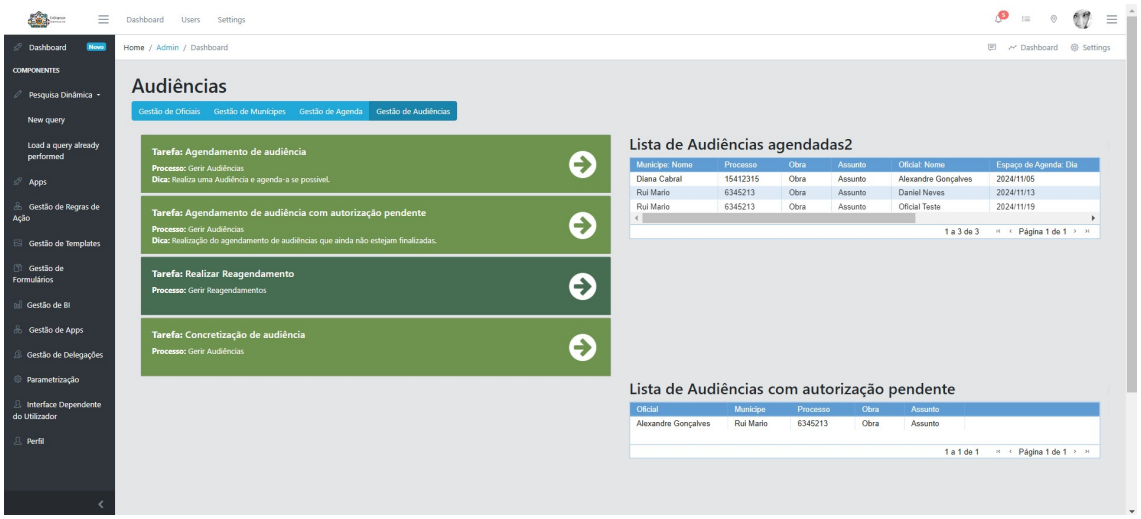

Figure 2.8: Hearing management interface in DISME

The primary interface for this process (Figure 2.8), whose development and final implementation was analyzed and compared, is the one responsible for managing hearings. It displays pending hearing requests and scheduled hearings and allows new hearing requests to be made. It also allows the selection of a pending hearing to assign it a Hearing Officer and a schedule slot. Scheduled hearings can also be rescheduled through this interface.

This UI contains the following components:

- A table displaying scheduled hearings, providing an overview of all of them.
- A table displaying pending hearing requests waiting for an officer's authorization and assignment.
- An action to request a new hearing, allowing users to initiate the process.
- An action to authorize and schedule a pending hearing request, enabling the assignment of an officer and one of his available schedule slots.
- An action to reschedule an existing hearing, allowing users to assign it a new schedule slot.

Requesting a new hearing requires the user to first select a Hearing Officer, a scheduled time slot, and the citizen involved, and provide the necessary information about the hearing, including the process, subject, and any relevant observations. If the selected officer doesn't require authorization, the hearing can be scheduled immediately; otherwise, it remains pending, awaiting confirmation from the officer.

The action to schedule a pending hearing requires the user first to select the hearing he wants to schedule and then choose one of the official's schedule slots.

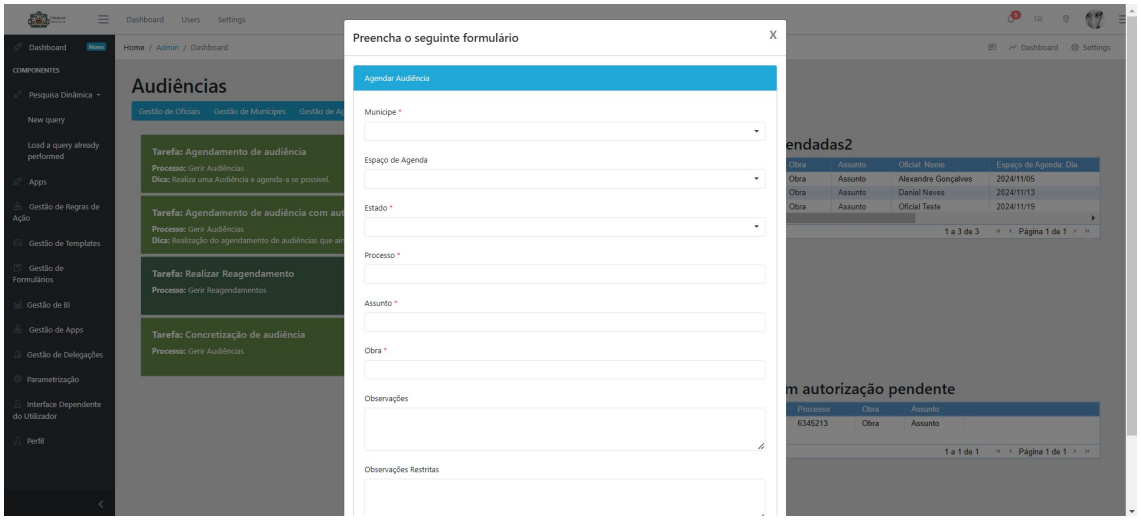

Figure 2.9: Modal with the form to request a new hearing

Finally, the action to reschedule a scheduled hearing requires the user first to select the hearing, then select the source (who is requesting the reschedule), select the new schedule slot, provide the reason behind it, and indicate when and how the involved parties should be notified (through email, phone call, or simple text message).

In the version implemented through DISME, all actions are executed through modal popups, as shown in Figure 2.9, that guide users in providing the required information through forms. After finishing each action successfully, the system dynamically updates the displayed tables to reflect the changes, ensuring a seamless and comprehensive UX.

2.5.2 Interface-first or database-first approach

Starting by developing the interface can help in understanding user requirements and preferences, leading to a more user-friendly design. This approach enables developers to gather user feedback early in the process, informing both the UI design and the database design [59].

Some LCPs offer greater flexibility in separating the definition of the UI from the underlying database schema, while others tightly couple UI components, especially tables and forms, to data models. In the latter case, it becomes necessary to define the database schema before designing or fully customizing the UI.

In OutSystems, the platform promotes a modular approach by allowing the development of the UI independently of the data model. Developers can begin by prototyping the interface and later define the data structures and business logic.

In PowerApps, the creation of forms and tables is more dependent on pre-existing data sources, limiting the interface-first approach. UI components such as tables and forms (for viewing and editing records) require an associated data source before they can be meaningfully customized. Although one can start with a layout or screen, significant

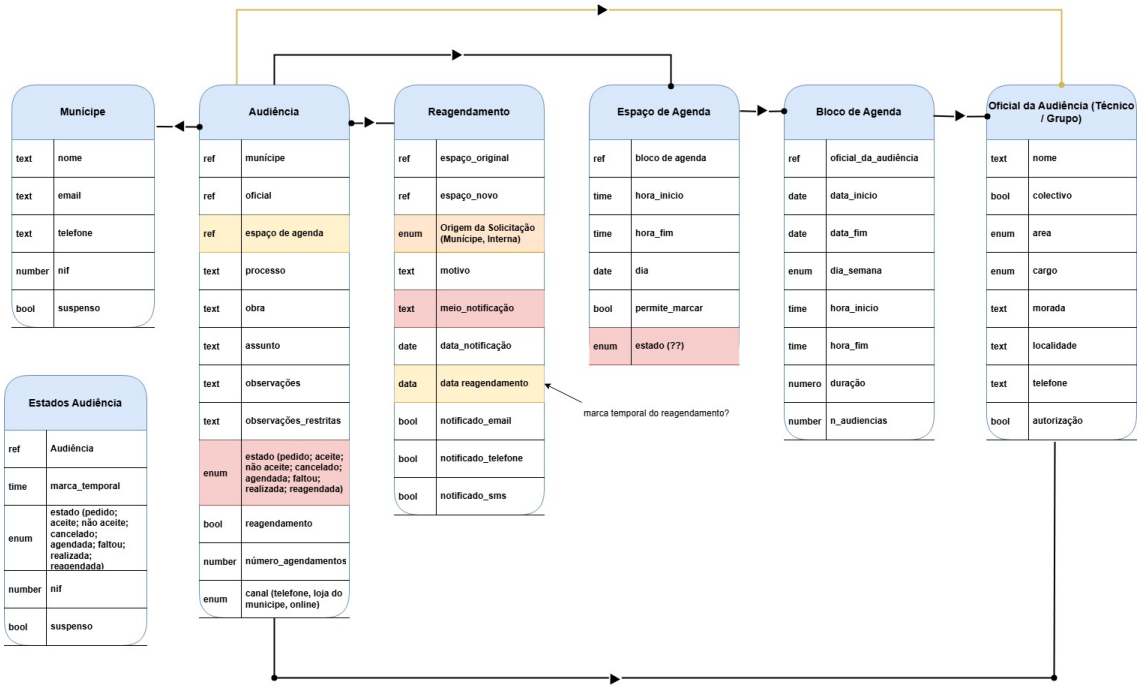

Figure 2.10: Municipality hearing process database diagram

UI customization (e.g., binding controls, defining validation rules, or specifying field formatting) depends on having the data schema in place.

In Mendix, greater flexibility is provided when defining the UI before the database is defined. Developers can design pages with tables (Data Grids) and forms, fully customize them, and bind them to a data source, such as entities or microflows (logic functions), later.

2.5.3 Database Creation

An essential step in creating the system across different platforms is establishing its database by defining the necessary entities and establishing the relationships between them.

Figure 2.10 shows, in a diagram, the entities and the relationships that were replicated across the four platforms.

The tables were then populated with some mock-up data (such as citizens, officers, schedule blocks, and slots) to be able to use the application without the need to implement the part of the system responsible for handling their creation, and to simplify this analysis.

2.5.3.1 OutSystems

Based on the diagram in Figure 2.10, we managed to obtain the database shown in Figure 2.11 in OutSystems.

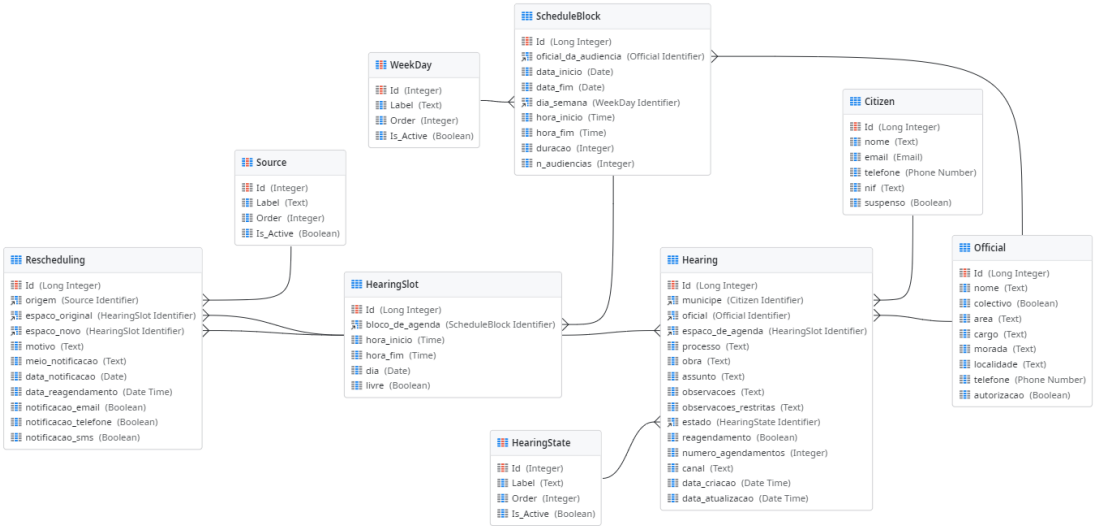

Figure 2.11: Entity diagram of the implemented application in OutSystems

In OutSystems, there are Entities and Static Entities [60]. Entities represent dynamic data that can be modified at runtime through Create Read Update Delete (CRUD) operations. These are used for storing business-related information, such as Hearings and Officers. Static Entities, on the other hand, contain pre-defined, immutable data that remains constant during runtime. They are typically used for categorical values, such as hearing states (e.g., Pending, Scheduled, Canceled) or days of the week.

To establish relationships between Entities, an attribute with a data type referencing another Entity's Identifier must be added. For example, the Hearing entity has attributes such as Citizen and Officer, which store the Citizen Identifier and Officer Identifier, respectively, linking them to the corresponding Entities.

Static Entities can also be linked to standard Entities in a similar manner. For instance, the Hearing entity contains an attribute state, whose data type is the Hearing State Static Entity. This allows hearings to be assigned a specific state from a predefined list of values.

2.5.3.2 PowerApps

PowerApps offers a visual approach to database definition, allowing users to create and manage tables through a graphical interface. Each Table consists of Columns, where each one has a data type (e.g., Text, Number, Boolean, Date, Lookup), and additional formatting options can be applied to enforce specific data constraints.

PowerApps enables the creation of relationships between tables through Lookup columns, which function as foreign keys by referencing records from other tables. For instance, in the Hearings table, the Officer attribute is defined as a Lookup column, linking it to the Officers table.

However, PowerApps lacks a dedicated time-only data type. To represent fields such

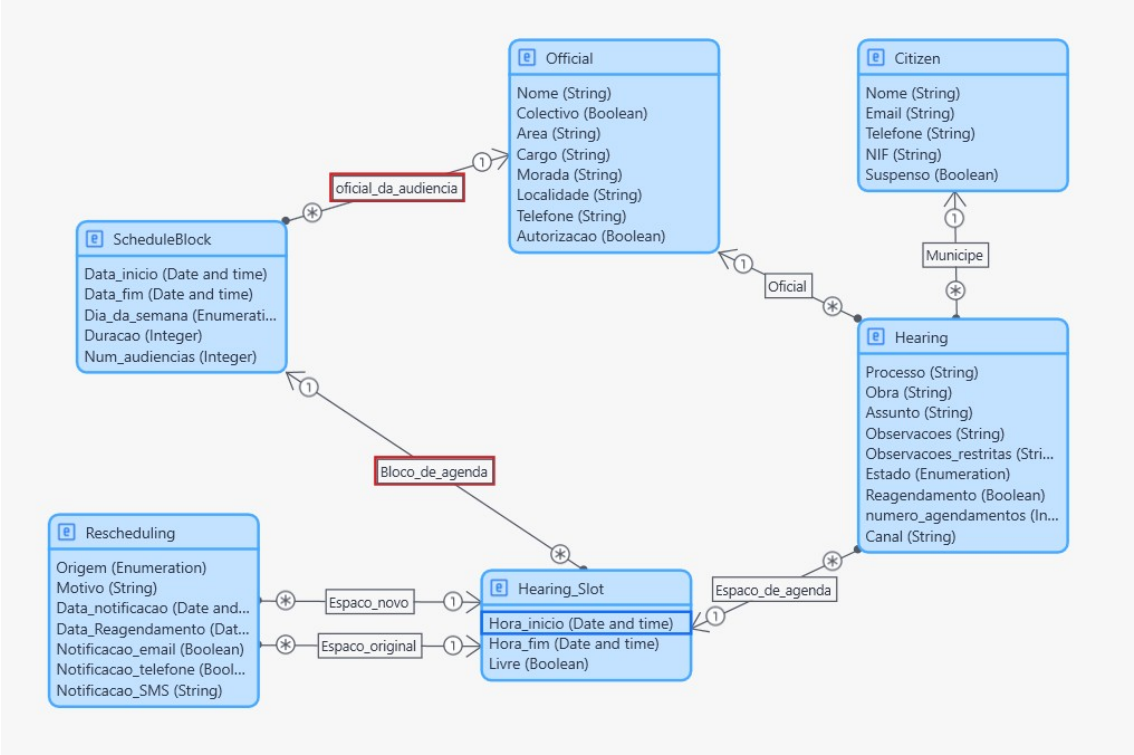

Figure 2.12: Diagram of the database entities defined on Mendix

as hearing slot start and end hours, we adapted by using DateTime fields, which store both date and time. This required handling the data in a way that extracts only the time portion when needed.

2.5.3.3 Mendix

Based on the diagram in Figure 2.10, we managed to obtain the database shown in Figure 2.12 in Mendix.

In Mendix, the database is defined visually, using Drag-And-Drop functionality, through the domain model, which represents the data structure of the application. Each module in Mendix can have its own domain model, making it easier to organize and encapsulate data relevant to specific features. An Entity in Mendix corresponds to a database table, each one with multiple attributes. Mendix models relationships between entities through Associations, which represent the links between tables. These are also added visually in the domain model by dragging a connector between two entities.

2.5.4 Designing the main interface

The next step in the process is designing the main interface, where users can perform the three available actions and view relevant hearing data. This page includes the action

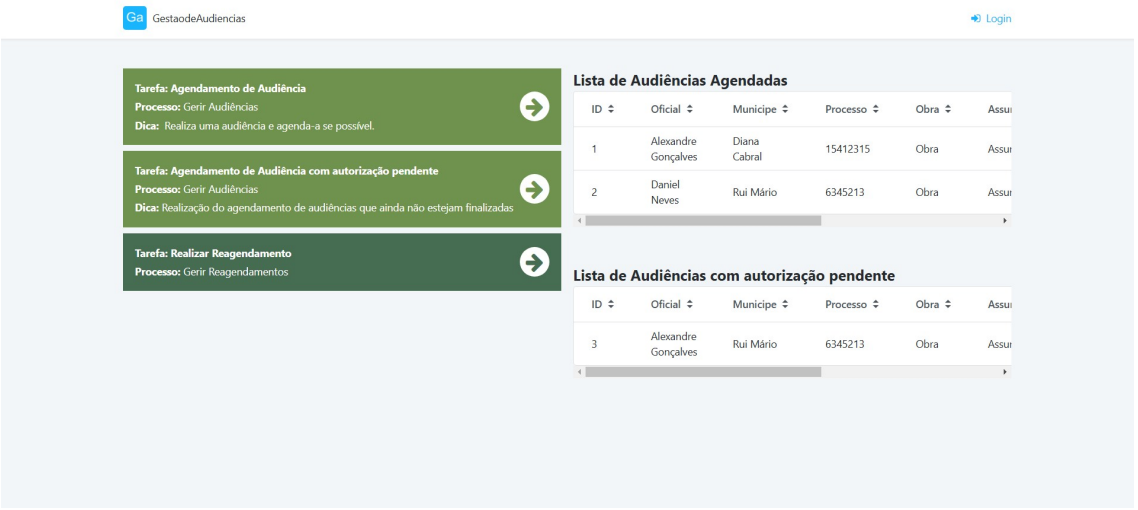

Figure 2.13: Hearing management page interface in OutSystems

controls that allow access to the key functionalities of the MHP and the tables displaying scheduled and requested hearings.

2.5.4.1 OutSystems

Following the same structure of the UI in the DISME version (Figure 2.8) we successfully implemented a very similar version in OutSystems (Figure 2.13).

We did so using pre-built UI components from OutSystems. The Adaptive Columns component was used to separate the main page into two columns, where the first one contains the actions and the second one contains the scheduled and pending hearings tables.

For the actions, Container and Text components were used to present the information about each action, as well as a clickable Icon to initiate the respective action. To perfectly align and position the Container and Text components, we had to resort to writing some CSS code manually for those components.

As for the Table components in the second column, it was necessary to wrap them in a Scrollable Area component to make them horizontally scrollable, as they contained many columns that did not fit appropriately into the screen.

Each table has a source, which can be assigned to an aggregate [61], an optimized query used to fetch data from the database. The goal was to display scheduled hearings in the first table and pending hearing requests in the second one, so an aggregate was created for each one. Both were the join of the Hearing, Officer, Citizen, Hearing Slot, and Hearing State Entities, but one was filtered to show only hearings where the state is scheduled (Figure 2.14), and the other where it is requested.

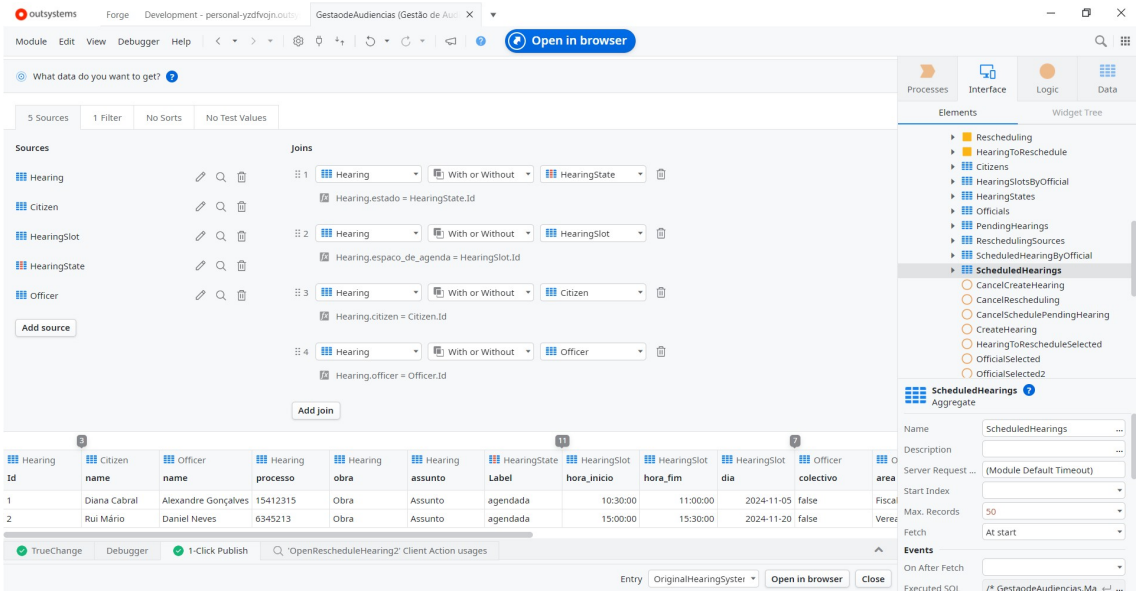

Figure 2.14: Aggregate in OutSystems to fetch scheduled hearings

2.5.4.2 PowerApps

PowerApps provides flexibility in designing the user interface, allowing components to be positioned according to a structured layout with components arranged within a grid or layout system defined by parent containers, ensuring alignment and responsiveness. Alternatively, components can be placed manually on the page, allowing for a more customized arrangement but requiring manual alignment adjustments.

For our implementation, we chose the free placement approach, as it provided greater control over the interface structure, a faster development process by reducing setup complexity, and an easier learning curve, especially for new users.

To display pending and scheduled hearings, we created two separate views on the hearings table: one containing only scheduled hearings and the other containing only requested hearings. We used Power Fx, PowerApps' declarative formula language similar to Excel functions, to filter the hearings by their state, extract and format the date, start time, and end time from the associated hearing slot using the *Text()* and *AddColumns()* functions, and finally sort the results by start time with the *Sort()* function, as shown in the following code.

The final result is displayed in Figure 2.15

2.5.4.3 Mendix

In Mendix, we designed the main interface using the *Atlas UI* framework, which provides pre-built templates, widgets, and layout structures that support responsive design and consistent styling across the application [62]. We selected a blank template to serve as the foundation of our primary interface.

```
Sort(
    AddColumns(
        Filter(Hearings; 'Hearings (Vistas)'.'Scheduled Hearings');
        'Date'; Text('Hearing Slot'.'Start Hour'; "dd/mm/yyyy");
        'Start Hour'; Text('Hearing Slot'. 'Start Hour'; "hh:mm");
        'End Hour'; Text('Hearing Slot'. 'End Hour'; "hh:mm")
    );
    'Start Hour';
    SortOrder.Ascending
)
```

Listing 1: PowerFX code to filter scheduled hearings, add custom columns, and sort by start hour

Figure 2.15: Main page of the Municipality Hearing Process in PowerApps

To structure the content on the page, we used the *Layout Grid* component, which allowed us to define a responsive column-based structure. Like the other platforms, our layout was divided into two main sections: the first contained the action controls, and the second displayed the scheduled and requested hearings.

For the action cards, we used a *Container* with a *Layout Grid* within each one to organize the action name, description, hint, and a button to initiate the action.

To display hearing data, we added two *Data Grids*, one configured to show scheduled hearings and the other for requested hearings. We bound each grid to a specific data source defined in the domain model and filtered the results using XPath constraints to only show records in the relevant state. The columns in each grid were customized to display the most relevant attributes, including the hearing date, start and end times, officer, and the names of both the officer and the citizen involved.

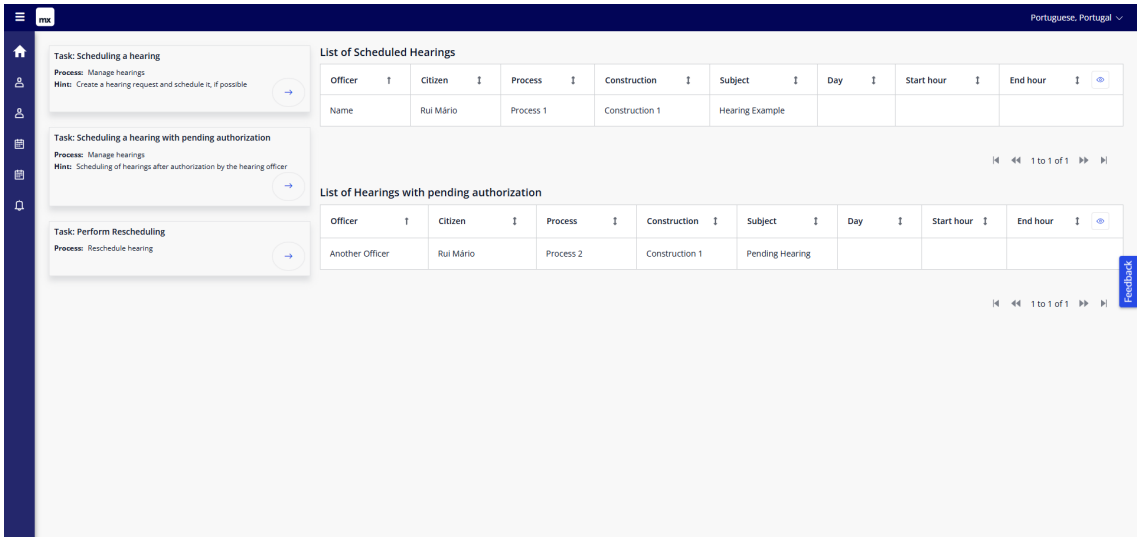

Figure 2.16: Hearing Management interface created on Mendix

Additionally, the Data Grids support built-in sorting and pagination, which improve usability when handling large amounts of hearing data.

The final result is shown in Figure 2.16.

2.5.5 Creating the forms

Besides the main page, it is necessary to implement forms that require the essential information to complete the execution of available actions, such as requesting a new hearing, scheduling a pending hearing, and rescheduling an existing hearing.

2.5.5.1 OutSystems

To this end, we utilized OutSystems' Modal, Form, and various input component types. Modal's visibility depends on the value of a boolean variable, and each input has to be associated with a variable of the corresponding type, which will contain its value and allow us to access it outside of the input component. We encountered some issues while implementing the modals because they lacked a built-in close feature, and some of them contained too many inputs that did not fit within the modal space.

To solve the lack of a close functionality, we had to manually add a close button and position it in the top right corner using CSS once again, and define an event so that when the button is clicked, the variable controlling the modal's visibility is set to false.

To solve the issue where the content did not fit into the modal, we attempted to use the already mentioned Scrollable Area component to make the content vertically scrollable; however, it was not working correctly in this case. So, after some research, we resorted to CSS to make the modal scrollable and set a maximum height for it.

2.5.5.2 PowerApps

PowerApps does not include a built-in Modal component for handling actions within the same page, so we implemented separate form pages for each action. To facilitate hearing selection during scheduling or rescheduling, we utilized PowerApps' table component instead of standard input fields. By embedding tables directly in the relevant form pages, users can select a hearing simply by clicking on it, potentially making the process more intuitive and efficient compared to dropdowns or manual input fields, as this approach can significantly reduce the number of steps required for selection.

Each form in PowerApps is associated with a specific data source, typically a table or database. Once linked, the form automatically generates input fields based on the selected attributes of the related table, using appropriate controls such as text inputs, date pickers, and dropdowns depending on the field type. These fields are bound to the corresponding columns in the data source, allowing for seamless data entry and updates.

2.5.5.3 Mendix

In Mendix, some pages for creating and editing entities are automatically generated, and developers can further customize their appearance and behavior.

For example, a form was generated to create or edit a hearing rescheduling. We modified the form's title and field labels to suit our specific requirements (Figure 2.17). This form includes various input fields, such as dropdown selections, text boxes, radio buttons, and date pickers, making it a good example to showcase the wide range of interface components Mendix provides.

2.5.6 Defining business logic

l To implement the application's business logic, we structured mechanisms to update the database, control the execution flow, manage navigation between screens, and display feedback to the user through screen notifications.

2.5.6.1 OutSystems

In OutSystems, the logic flows are defined through Actions that can run on the server or client side of our application [63]. We created actions to open and close modals, change the values of certain variables, perform read and write operations on the database, refresh aggregates as needed, and so on. Figure 2.18 showcases an example of a logic flow created to perform the rescheduling of a hearing.

2.5.6.2 PowerApps

In PowerApps, business logic is defined using PowerFx, a declarative formula language similar to Excel functions. We leveraged Power Fx to handle form submissions, send notifications, navigate between screens, and update the database.

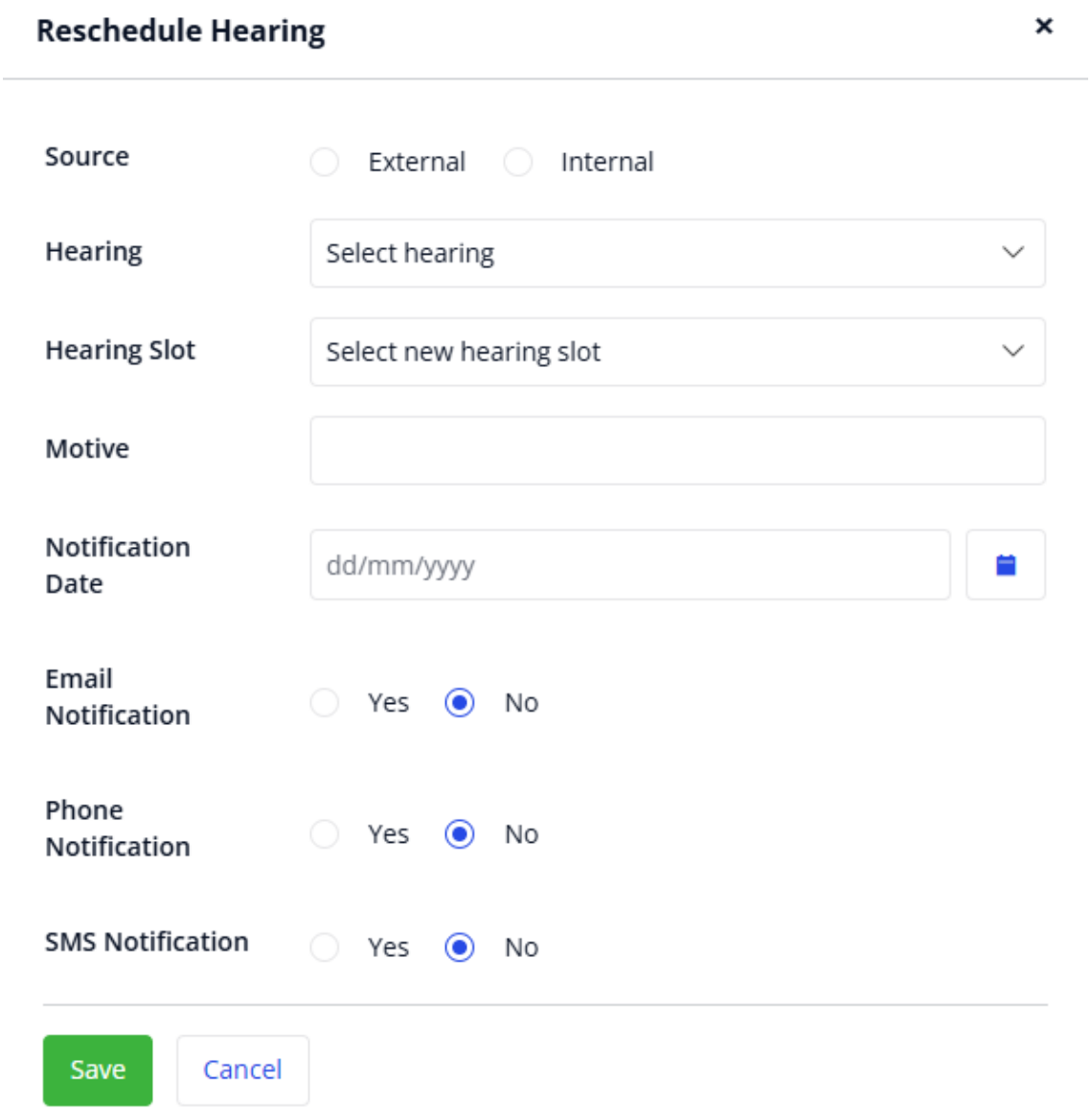

Figure 2.17: Mendix generated form to reschedule a hearing

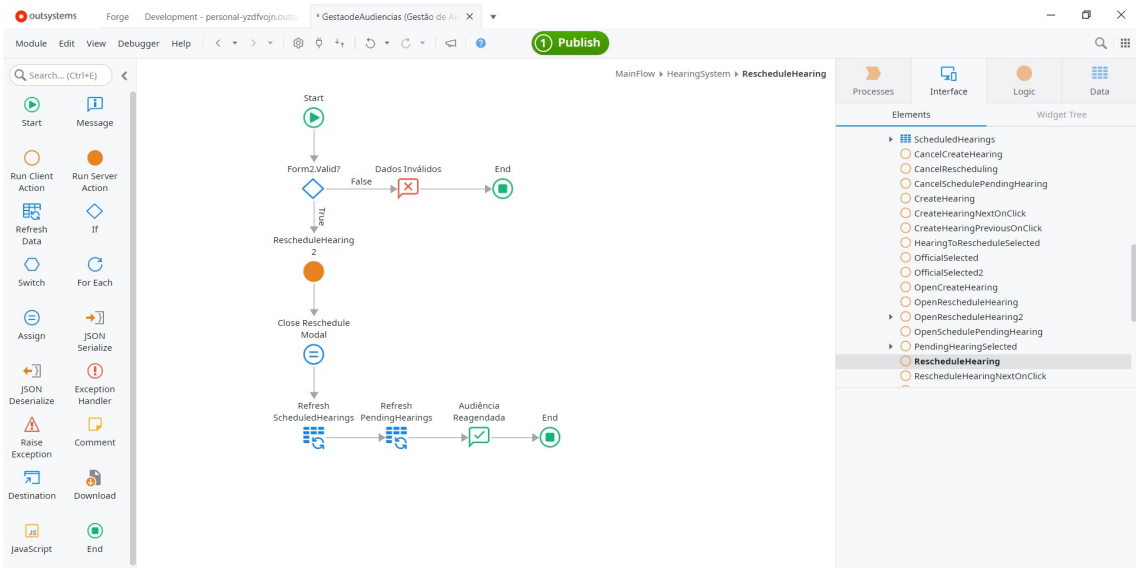

Figure 2.18: OutSystems action to reschedule a hearing

Each form page includes a submit button that triggers the *SubmitForm()* function when clicked.

The action performed depends on the form's mode, which can be view, update, or create. Forms used for requesting a new hearing or rescheduling a hearing operate in create mode, as they insert new records into the database. The form for scheduling a hearing operates in update mode, as it modifies an existing record in the Hearings table. When a form is submitted in create mode, a new row is added to the corresponding data table. In update mode, an existing row is modified.

Some actions also require updating data in related tables. For instance, when scheduling or rescheduling a hearing, the availability of hearing slots must be updated accordingly. To achieve this, we utilized the *Patch()* function in PowerFx, which enabled us to modify records after a form was successfully submitted. Additionally, we used the *Notify()* function to inform users about the outcome of their actions and the *Navigate()* function to return to the main page after a successful submission. The following code showcases an example of PowerFx code used to send a notification, update the hearing's slot availability, and return to the main screen after successfully submitting the form to schedule a pending hearing.

When submitting the form to schedule a hearing, updating the hearing state from *Requested* to *Scheduled* required a workaround. The hearing state field had to remain visible, even if set to view-only mode, with *Scheduled* selected by default. If the field was hidden, PowerApps did not update the state correctly, resulting in database inconsistencies.

```
Notify(
    "Hearing scheduled successfully";
    NotificationType.Sucess
) & Patch(
    'Hearing Slots';
    LookUp(
        'Hearing Slots';
        ID = HearingSlotDropdown.Selected.ID
    );
    {Available: false}
).'Hearing Slot' & Navigate('Main Screen')
```

Listing 2: PowerFX code to display a notification and update hearing slot availability

2.5.6.3 Mendix

In Mendix, business logic is primarily defined through microflows, which are visual representations of application logic. These microflows allow developers to model complex workflows without writing traditional code. Each micro-flow consists of various elements such as events, decisions, activities, loops, and flows, enabling the creation of dynamic and responsive applications.

A micro-flow begins with a start event and concludes with an end event. Between these events, developers can incorporate activities like creating or updating objects, calling other microflows, or interacting with external systems. Decisions are used to branch the flow based on conditions, allowing for dynamic routing. Loops facilitate iteration over collections, and flows connect these elements to define the sequence of execution.

To illustrate the application of these concepts, consider the micro-flow shown in Figure 2.19. This micro-flow defines the initial state of a hearing at the moment it is created. The logic depends on whether the assigned officer requires authorization: if authorization is needed, the hearing state is set to *Pending*; otherwise, it is directly set to *Scheduled*.

2.5.7 Implementing table actions

The primary focus of this thesis is the implementation of table actions within DISME. In particular, we aim to enable citizen developers to execute MHP actions directly from the hearing tables, providing immediate access to hearing details. This feature represents a key enhancement to DISME, potentially improving usability and offering a new workflow for executing actions for end-users of the generated applications.

2.5.7.1 OutSystems

OutSystems facilitates the implementation of actions and access to hearing details directly from the tables by allowing the insertion of new columns and button components within them.

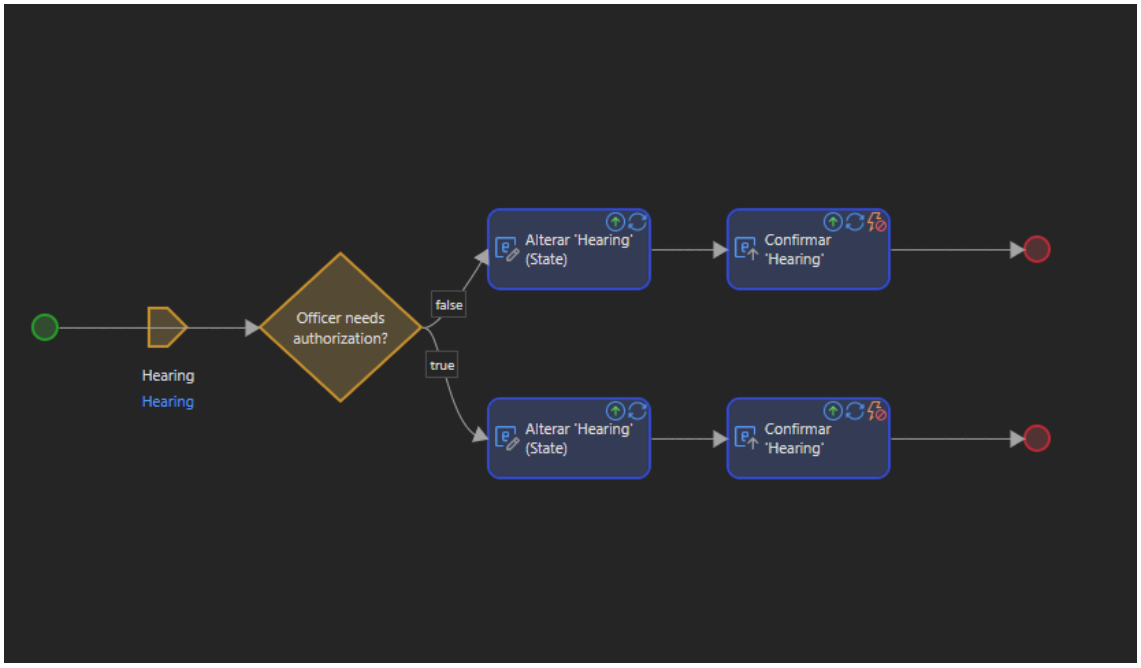

Figure 2.19: Mendix micro-flow to create a new hearing

For example, to streamline the rescheduling process of a hearing, a rescheduling button was added to a new column in the table containing the scheduled hearings, enabling users to bypass initial selection steps, because the selected hearing is set as the one in the same row as the clicked button, and proceed directly to selecting the rescheduling source and filling in the needed information. Besides this button, a button to see the details of a hearing was added, and in the table of pending hearings, a button to schedule a pending hearing was added (Figure 2.20).

Adding just the buttons with an icon would not be enough for users to understand their functionality, so we wrapped each one in a tooltip component to display information about it when the user hovers over the buttons. This use of a tooltip plays a crucial role in enhancing inclusiveness because, as highlighted in the GenderMag framework, some users, particularly those who are more risk-averse or less inclined to explore unfamiliar features, may hesitate to interact with buttons if they are unsure of their functionality.

2.5.7.2 PowerApps

PowerApps tables do not support embedding actions directly within them. However, they include a *Selected* attribute, which references the currently selected row and can be accessed externally in various contexts. We leveraged this feature to streamline the selection process for pending and scheduled hearings, reducing the need for additional input fields or manual selection mechanisms.

Figure 2.21 illustrates how a table displaying scheduled hearings was used to select a hearing for rescheduling, enhancing usability and efficiency.

Figure 2.20: Hearing management improved interface in OutSystems

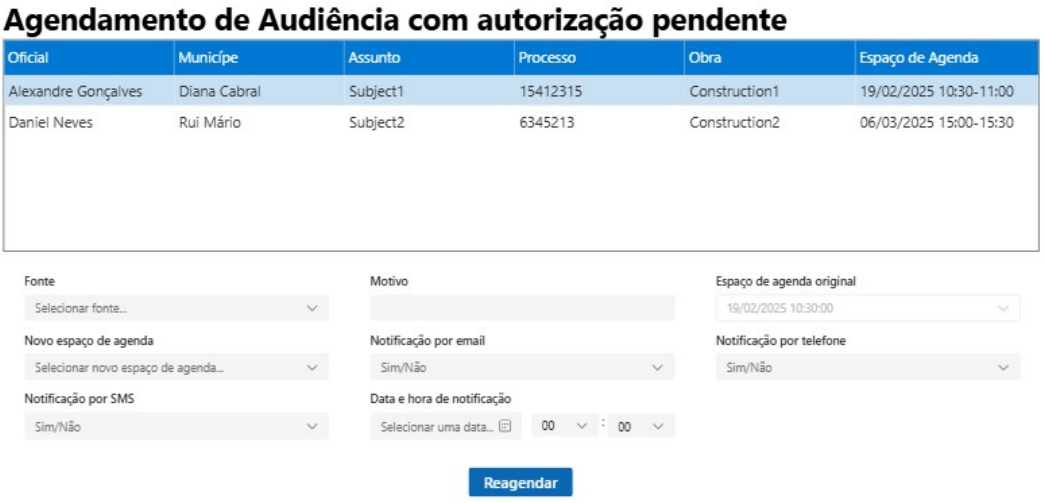

Figure 2.21: Reschedule hearing form implemented in PowerApps

@default@default

Figure 2.22: Hearing Management interface with table actions implemented in Mendix

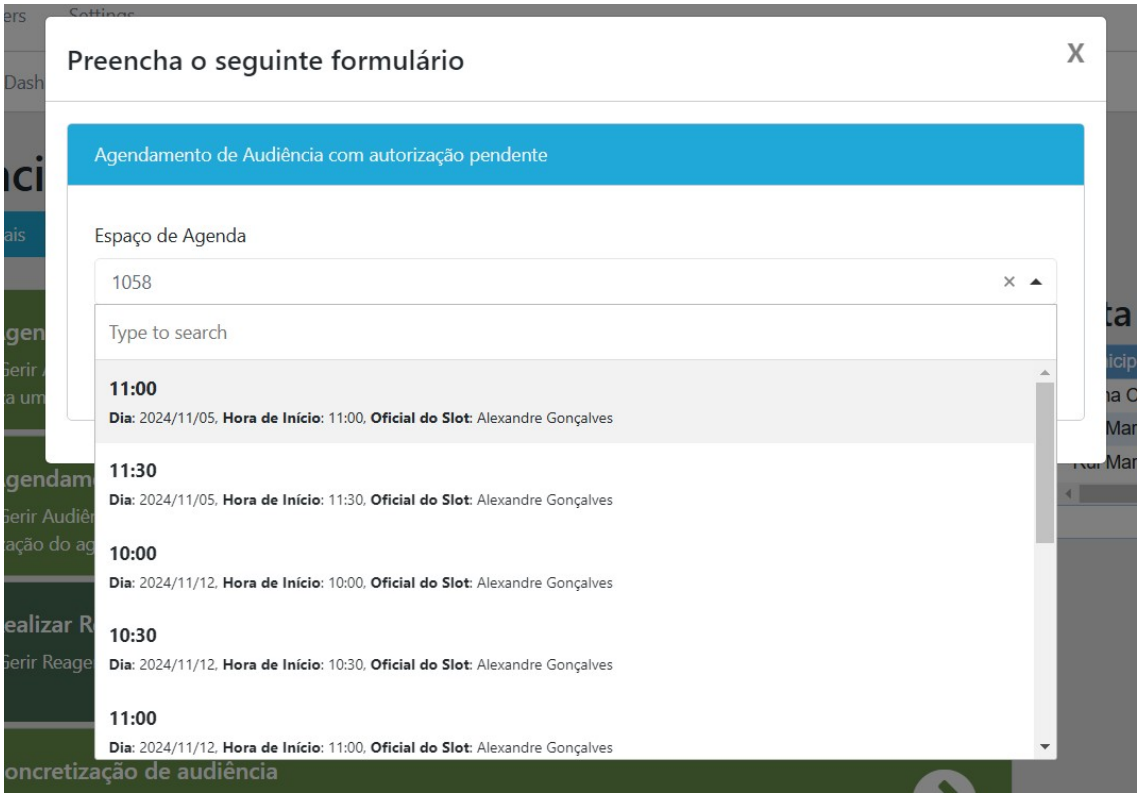

Figure 2.23: Drop-down list to select a schedule slot when scheduling a pending hearing

2.5.7.3 Mendix

Mendix allows citizen developers to easily attach table actions to the top of a table, enabling them to open new pages or trigger workflows to perform specific operations. These actions can receive parameters based on the selected rows, allowing execution for one or multiple records as needed.

The appearance and behavior of these action buttons are customizable: developers can select an icon, write a tooltip, or display them as links instead of standard buttons, offering flexibility in design. Actions that depend on row selection are only active when at least one row is selected, ensuring proper user interaction and preventing errors.

Figure 2.22 shows the updated interface of the Hearing Management page, now with table actions integrated to edit, reschedule, and schedule pending hearings.

2.5.8 Improving hearing slot selection in forms

A key usability improvement requested was enhancing the process for selecting a hearing slot. Currently, hearing slots are chosen through a drop-down list that displays each slot's date, start time, and associated hearing officer, as shown in Figure 2.23.

To improve this process, we aimed to implement a two-step selection approach: first, selecting a date using a calendar input that highlights and restricts choices to days with available slots, and then selecting a specific time slot for the chosen date using a drop-down list or a similar component. This method would make the selection process more intuitive while also displaying the weekday of the scheduled hearing, which is an essential factor for those involved in choosing a suitable date and time.

2.5.8.1 OutSystems

In OutSystems, implementing this feature was not feasible, as its calendar component only allows setting a minimum and maximum date range, rather than selectively highlighting available dates. This limitation prevented us from restricting and visually indicating only the days with open hearing slots.

2.5.8.2 PowerApps

Similar to OutSystems, PowerApps' date picker component only supports defining a start and end date, but does not provide functionality to highlight or filter specific dates based on availability.

2.5.9 Further changes

Given that each action in the MHP comprises multiple steps displayed in sequential modals, without an option to navigate back, we found it helpful to make that addition in the platforms that made sense. This would allow users to return to previous steps and make changes to the filled-in information without needing to restart the entire process and lose all progress.

In OutSystems, we used the Wizard component to further improve UX, in addition to adding the option to navigate to the previous step. This addition provides users with a clear visual representation of the process steps, allowing them to track completed steps, identify their current position, and also navigate back, by clicking the newly added "previous" button, to the desired step to correct mistakes without restarting the entire process and losing all progress, making the whole process more efficient if backtracking is necessary (Figure 2.24).

The Wizard component does not support an easy and direct way to define its logic, so all of it had to be manually handled. This included creating a new variable for each MHP action to track the current step, merging the multiple modals of each action into a single one, and dynamically updating both the modal's content and the Wizard's visual representation based on the current step.

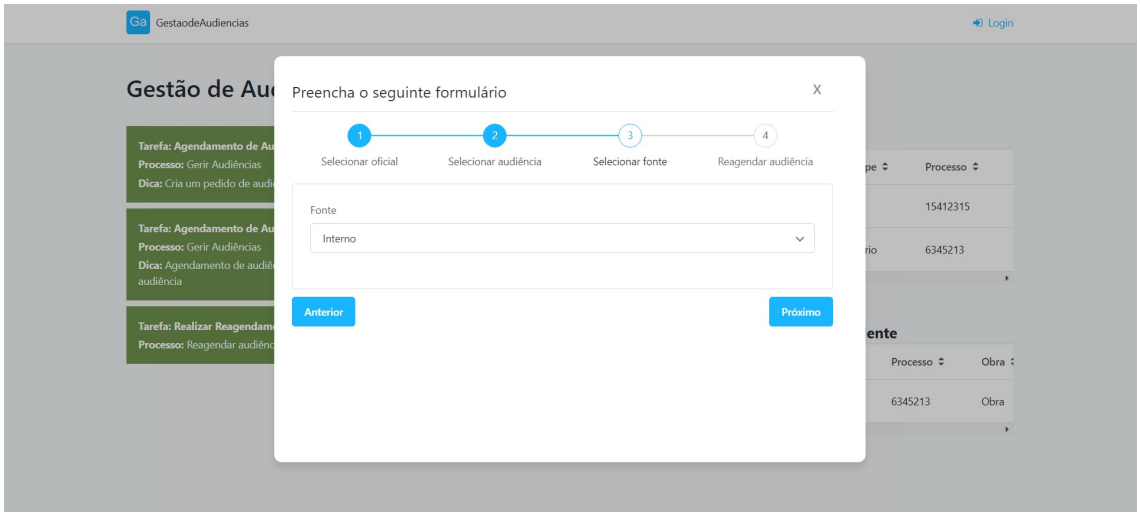

Figure 2.24: OutSystems wizard used to track the steps of the action to reschedule a hearing

2.5.10 Conclusion

We successfully implemented a working version of the application in OutSystems, adding some improvements in some cases that could also be applied to DISME to enhance its ability to address efficiency, usability, and inclusivity issues.

The comparative analysis between different LCPs was particularly valuable in informing our design decisions for implementing table actions in DISME. Each platform offered unique approaches to embedding actions in tables, which allowed us to reflect on the different options:

- **OutSystems** demonstrated the effectiveness of embedding action buttons directly in tables, but also highlighted the importance of tooltips for inclusiveness, especially for users who are more risk-averse and hesitant to click unfamiliar buttons.
- **PowerApps** lacks embedded table actions but provides a *Selected* attribute on tables to allow the passage of records to actions outside the table.
- **Mendix** places actions above the table and ties their availability to row selection.

We chose to adopt the implementation style of table actions used in OutSystems to guide our implementation on DISME, as it reflects a pattern typical in many applications: actions are attached to a specific row through an additional column. This makes it easy for users to immediately identify which record will be passed as a parameter, reducing ambiguity. However, this approach does not naturally support multi-record operations, a limitation that contrasts with the selection-based strategies offered by PowerApps and Mendix.

Beyond table actions, the exploration of OutSystems also revealed the potential benefits of introducing a Wizard component to guide users through multi-step workflows. Wizards

can improve task clarity but need careful design to avoid too many steps or overly long steps that negatively impact the UX [64]. Alternatives, such as minimizing task fragmentation to reduce interaction cost, must also be considered [65]. As such, usability testing is the most reliable method for determining the optimal balance in structuring workflows.

To confirm that these additions can indeed improve the usability and inclusivity of applications, it is necessary to conduct an experimental evaluation comparing a version of the application with certain additions and changes against the original version. Such a study should evaluate whether users perform tasks more effectively, more efficiently, and with greater satisfaction.

Moreover, we identified opportunities to improve the usability of the current implementation of the MHP in DISME, such as:

- Supporting navigation between steps of the MHP actions, allowing users to backtrack and correct mistakes without losing progress unnecessarily.
- Removing the input to select the hearing's state when requesting a new one, since its state can be automatically defined based on whether the assigned hearing officer requires authorization, preventing errors and inconsistencies.

Finally, our hands-on experience with OutSystems also revealed some challenges: the recurrent need to rely on CSS for layout customization, limited flexibility in specific components (e.g., drop-down lists), and the lack of default behaviors (e.g., modals without close options, tables and modals not automatically adapting to size). These limitations underscore the importance of providing higher-level abstractions in DISME that make common interactions and expected behaviors more accessible to citizen developers, or already integrated into DISME interface components, without requiring manual workarounds.

3

Requirements

Requirements define the basis for designing and developing the new feature set. They establish the functional and non-functional expectations of the system, as well as the perspective of its intended users. This specification of requirements reduced ambiguity during the implementation and provided a reference point for the evaluation.

This chapter distinguishes between functional requirements, which describe the expected system behaviors and features, and non-functional requirements, which specify quality attributes such as usability, performance, and scalability. Additionally, user stories are included to capture the perspective of the DISME citizen developers and end-users.

3.1 Functional Requirements

Functional requirements define a group of features that the system shall support. For each requirement, we describe the expected system behavior and its expected result. These requirements are usually expressed through "shall" statements.

3.1.1 Table Actions Management

The Table Actions Manager shall serve as the central hub where citizen developers can:

- Visualize all available table actions and lists of table actions;
- Create, edit, and delete both table actions and table action lists;
- Visualize a preview of how the table action button will look while creating a table action.
- Associate lists of table actions with one or more queries.
- Associates lists of table actions with one or more table actions.

Additionally, each table action shall include an icon to represent the rendered button and a tooltip text to provide contextual help when hovering over the button.

3.1.2 App Management

The App Manager, responsible for managing applications created on DISME, shall enable the citizen developer to insert tables, both with and without a list of table actions, into an application.

3.1.3 App Render

The App Renderer, responsible for rendering the generated applications to end-users, shall:

- Render tables with an embedded list of action buttons, if defined by the citizen developer;
- Enable end-users to initiate operations directly from table rows;
- Ensure that table actions skip unnecessary input steps by reusing data already available in the selected row.

3.1.4 Language Support

The system shall support multiple languages to ensure accessibility for citizen developers and end-users whose primary language is not Portuguese. For example, an English-speaking user shall be able to interact with the Table Actions Manager without language-related issues. This includes supporting multilingual labels, tooltips, and descriptions.

3.2 Non-Functional Requirements

Non-functional requirements specify the quality attributes of the system, focusing not on what the system does, but on how it behaves. These requirements are critical to ensuring adoption and long-term maintainability.

3.2.1 Usability

The Table Actions Manager shall provide an intuitive interface designed for new and experienced citizen developers of DISME. Basic functionalities such as creating, editing, and deleting table actions and lists of table actions shall be easily discoverable and accessible with minimal navigation effort.

The App Render shall ensure that end-users can interact with tables containing table actions without much trouble. Action buttons shall be visually distinguishable, consistently positioned, and accompanied by icons and tooltips to clarify their purpose. The system shall minimize cognitive load by providing clear feedback after each interaction, such as confirming when an operation has been successfully initiated or completed.

3.2.2 Performance

The Table Actions Manager shall ensure low latency during interactions. As a general rule, response times for common operations, such as loading the list of table actions, loading select form field options, or submitting a creation form, should not exceed 2 seconds.

3.2.3 Scalability

The system shall allow the unrestricted creation of table actions and lists of table actions. Citizen developers shall be able to associate multiple queries and actions with each list, without practical limitations on scale. The design shall ensure that performance remains acceptable even as the number of table actions and lists grows substantially.

3.2.4 Accessibility

The table actions manager and the other new interface elements created should provide multilingual support, such as Portuguese and English. The text should be clear and understandable for diverse user demographics.

3.3 User Stories

User stories are concise, informal descriptions of a system's functionality from the user's perspective. They capture the user's perspective on what the system needs to do to achieve their goals or complete specific tasks. User stories are typically written in the format of "As a [user], I want to [action], so that [benefit]".

The following stories summarize the expected interactions with the Table Actions Manager and generated applications.

- **User Story 1** : As a DISME citizen developer, I want to access the table actions management page, so that I can manage table actions and lists of table actions from one location.
- **User Story 2** : As a DISME citizen developer, I want to create a new table action, so that I can reuse it in applications across different contexts.
- **User Story 3** : As a DISME citizen developer, I want to preview how the table action button will appear in an application, so that I can evaluate its clarity and design before saving it.
- **User Story 4** : As a DISME citizen developer, I want to edit an existing table action, so that I can update its properties or correct mistakes.
- **User Story 5** : As a DISME citizen developer, I want to delete an outdated table action, so that I can maintain a clean and relevant set of actions.

- **User Story 6** : As a DISME citizen developer, I want to see all existing table actions, so that I can reuse or modify them without recreating similar actions.
- **User Story 7** : As a DISME citizen developer, I want to create a new list of table actions, so that I can group them into reusable collections.
- **User Story 8** : As a DISME citizen developer, I want to edit an existing list of table actions, so that I can keep the list aligned with evolving business processes.
- **User Story 9** : As a DISME citizen developer, I want to delete a list of table actions, so that I can remove obsolete or redundant lists.
- **User Story 10** : As a DISME citizen developer, I want to insert a table with a list of table actions into an application, so that end-users can initiate operations directly from the table.
- **User Story 11** : As an end-user of DISME applications, I want to execute operations directly from a table row, so that I avoid manually entering information that is already available in the table.
- **User Story 12** : As an end-user, I want the action buttons in the table to be clearly labeled and visually consistent across applications, so that I can recognize and use them without confusion.

4

Implementation

This chapter presents the technical implementation of the new table actions feature in DISME. It details the modifications made to the system's architecture, data model, controllers, and user interface to support executable actions embedded directly in table rows. The chapter also explains how the feature integrates with existing multilingual logic.

4.1 Introduction

To improve the efficiency in executing specific tasks and enhance the interactivity of tables in applications generated by DISME, we implemented a feature that enables the addition of executable actions directly on table rows. These actions are rendered as buttons, visually integrated into each row, and are associated with a predefined action rule that is executed when clicked, with the given row data passed as a parameter. For example, in the context of the MHP, these actions can support scheduling pending hearing requests, rescheduling hearings, or opening a detailed view of the hearing.

4.2 Data Flow in DISME

To understand how table actions operate within the system, it is useful to follow the path of data from the moment a user interacts with the interface to when the backend processes the request and returns a response. The following steps outline the flow of data in DISME applications:

1. **Front-end Request** Users interact with the management pages through the System Modeler interface. When an action is performed (e.g., creating a new entity or retrieving data), the front-end issues an Hypertext Transfer Protocol (HTTP) request to the corresponding Representational State Transfer (REST) endpoint. These requests are typically sent in JavaScript Object Notation (JSON) format and may include data.
2. **Controller Handling** The appropriate controller receives the request, validates the incoming payload, and delegates the logic to the model layer. For example:

- The `store` method creates new entries in both the base table and its multilingual text table.
- The `update` method modifies the existing records.
- The `destroy` method deletes the target record and cascades deletions in related pivot or text tables. If the table has soft deletes, only the `delete_at` and `deleted_by` columns are changed.

3. **Database Interaction** Once validated, the controllers interact with the Eloquent models, which translate the high-level operations into SQL queries. These queries handle inserts, updates, deletes, or joins across the related base and multilingual tables.

4. **Resource Transformation** Before sending the response back to the client, the raw Eloquent model instances are wrapped inside API Resources. These resources define the JSON format of the response, ensuring consistency across endpoints. Additionally, multilingual fields are dynamically resolved using the `getMultilingualConceptName` helper function, so that the client always receives localized names and tooltips according to the user's language preference.

5. **Frontend Rendering** The client receives the standardized JSON response and updates the management tables accordingly.

4.3 New DISME architecture

This new feature required modifications to existing DISME components, both on the System Modeler and System Executor sides, as well as the creation of a new component on the System Modeler to manage table actions and group them into lists.

Figure 4.1 shows the new DISME architecture and highlights the newly created component in the System Modeler and the components that had to be updated to implement the new table actions feature, in this case, the App Management component in the System Modeler and the Dashboard in the System Executor.

4.3.1 System Executor

On the System Executor side, we made modifications to the **Dashboard** to integrate the new management component.

The Dashboard has been extended with a new navigation entry under the action rules menu in the sidebar, as it is highly related to action rules: each table action is linked to an action rule.

Previously, the Action Rules menu was a single direct entry to the Action Rules Management page; however, it now opens a sub-menu with two navigation entries: the previous entry to the Action Rules Management page and one to the new Table Actions Management page.

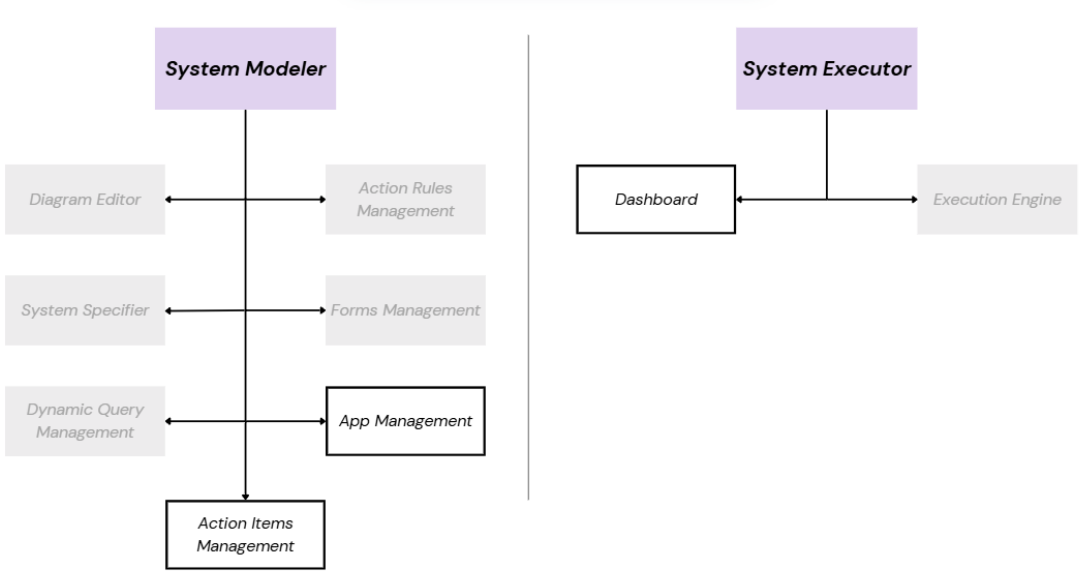

Figure 4.1: DISME architecture, highlighting components where changes were made to implement the new table actions feature

4.3.2 System Modeler

On the System Modeler side, we created a new **Table Actions Management** component.

This new component features table views of existing entities, enabling users to create, update, and delete table actions and lists of table actions through an interface that adheres to the same design principles as other existing management components, ensuring consistency in the UX.

We updated the App Management component to support the integration of tables with embedded actions on applications.

When adding a new table within an application, users are now prompted to specify whether the table should include action buttons.

Based on this selection, the available queries are dynamically filtered: if table actions are enabled, only queries already associated with lists of table actions are displayed.

Once configured, the rendered table includes an additional dedicated column, where the action buttons are rendered.

These buttons enable users to trigger predefined action rules directly from each row, thereby accelerating the process by skipping steps where some required data for the action is selected, as the row's data is passed to the execution engine.

4.4 Back-end changes

The back-end of the project is implemented in Laravel, handling data persistence, business logic, and exposed API endpoints. The back-end manages the creation, retrieval, update, and deletion of entities and enforces validation and security rules.

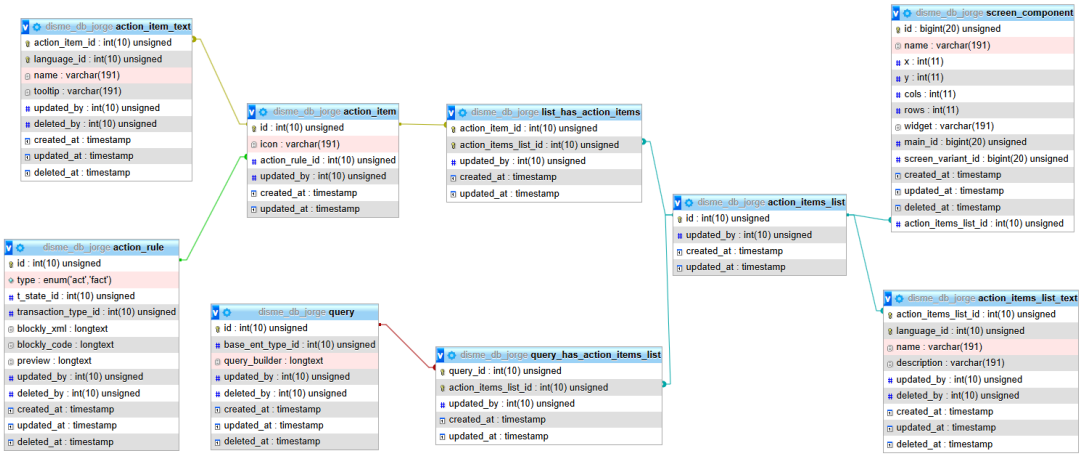

Figure 4.2: Simplified diagram of the new and related database tables and their relationships

4.4.1 Data model

To support the implementation of table actions, we introduced new entities into the system. The core entities are `ActionItem` and `ActionItemsList`, which represent individual table actions and reusable lists of actions, respectively.

Their relationships are managed through two pivot tables. The `ListHasActionItems` table establishes the many-to-many relationship between `ActionItemsList` and `ActionItem`, while the `QueryHasActionItemsList` table links a `Query` to a specific `ActionItemsList`.

Since user interface elements, such as tables, tasks, or forms, are stored in the database as `ScreenComponent` entities, we extended this model with a new nullable attribute, `action_items_list_id`. This field stores the identifier of the associated list of table actions, enabling the App Management component to distinguish between tables that support table actions and those that do not.

Additionally, multilingual support is provided through textual tables (`ActionItemText` and `ActionItemsListText`), which store language-specific attributes such as names, descriptions, and tooltips.

Figure 4.2 illustrates the structure of the newly introduced tables, their relationships with existing entities, and the integration of multilingual support and pivot tables.

Regarding deletion policies, we made a distinction between core and textual tables. For the newly introduced core entities (`ActionItem` and `ActionItemsList`) and their pivot tables, soft deletes were intentionally excluded. Unlike other fundamental entities of the platform, where historical recovery may be critical, these tables are lightweight and can be recreated quickly if needed. Their primary purpose is to serve as links between textual data, action rules, queries, and screen components, so permanent deletion has minimal system impact.

In contrast, the multilingual textual tables do support soft deletes. The current implementation requires this of the `getMultilingualConceptName` function, which relies on soft delete semantics to retrieve localized values correctly. This design ensures that multilingual consistency is preserved while avoiding unnecessary complexity in the core tables. Only the most relevant auditing information, such as `updated_by` and creation/update timestamps, is stored in the non-textual entities to keep them as simple as possible.

4.4.1.1 ActionItem

The **ActionItem** model defines a single executable Table Action that can be grouped into a list of Table Actions. It encapsulates both its visual representation and the logic it triggers.

Its attributes are the following:

- `id`: Primary identifier of the table action.
- `action_rule_id`: Foreign key referencing the **ActionRule** that defines the operation executed.
- `icon`: FontAwesome icon representing the action visually.
- `updated_by`: Reference to the user who last updated the record.
- `created_at`, `updated_at`: Timestamps for auditing.

Choice of Icon Library - For the representation of Table Action buttons, we chose the FontAwesome library, because it offers vector-based icon fonts for web and mobile applications, allowing for simple adjustments to size, color, and responsiveness, if necessary, without requiring image editing. This icon library has been mentioned in a research paper that covered the technical implementation and its real-world use in a web application [66] and another that provided an overview of the advantages of using icon fonts in web and mobile applications, especially when the goal is responsive design [67]. This choice was also motivated by both technical compatibility with the project's stack and the versions of its dependencies, as well as the extensive set of free icons it provides. FontAwesome is also adopted across some LCPss. For example, Budibase utilizes it as its primary icon library, while others support it through their marketplace modules or by uploading external icon collections. By adopting FontAwesome, our project benefits from a well-supported, flexible, and widely recognized standard icon library.

4.4.1.2 ActionItemText

The **ActionItemText** model stores multilingual values for each **ActionItem**. It ensures that each action can be displayed in the user's preferred language, supporting usability and accessibility requirements.

Its attributes are the following:

- `action_item_id`: Foreign key referencing the **ActionItem**.
- `language_id`: Language identifier.
- `name`: Localized name of the Table Action.
- `tooltip`: Localized tooltip shown on hover.
- `updated_by`: Reference to the user who last updated the record.
- `created_at`, `updated_at`: Timestamps for auditing.
- `deleted_at`: Timestamp used for soft deletes.

This table is indexed by the composite key `(action_item_id, language_id)`, ensuring one localized entry per language.

4.4.1.3 ActionItemsList

The **ActionItemsList** represents a reusable collection of Table Actions. Each list groups related table actions together, which can then be bound to one or more tables.

Its attributes are the following:

- `id`: Primary identifier of the list of table actions.
- `updated_by`: Reference to the user who last updated the record.
- `created_at`, `updated_at`: Timestamps for auditing.

4.4.1.4 ActionItemsListText

The **ActionItemsListText** model stores multilingual attributes for an **ActionItemsList**, ensuring lists can be presented in different languages.

Its attributes are the following:

- `action_items_list_id`: Foreign key referencing the **ActionItemsList**.
- `language_id`: Language identifier.
- `name`: Localized name of the list.
- `description`: Localized description of the list.
- `updated_by`: Reference to the user who last updated the record.
- `created_at`, `updated_at`: Timestamps for auditing.
- `deleted_at`: Timestamp used for soft deletes.

This table is indexed by the composite key `(action_item_list_id, language_id)`.

4.4.1.5 ListHasActionItems

The **ListHasActionItems** model is a pivot entity that represents the many-to-many relationship between **ActionItemsList** and **ActionItem**. Each Table action can belong to multiple lists of table actions, and each list of table actions can contain several table actions.

Its attributes are the following:

- `action_items_list_id`: Foreign key referencing the **ActionItemsList**.
- `action_item_id`: Foreign key referencing the **ActionItem**.
- `updated_by`: Reference to the user who last updated the record.
- `created_at`, `updated_at`: Timestamps for auditing.

4.4.1.6 QueryHasActionItemsList

The **QueryHasActionItemsList** model is another pivot entity that represents the many-to-many relationship between **Query** and **ActionItemsList**. This model helps identify which queries have an associated list of table actions and which lists of table actions are associated with them.

Its attributes are the following:

- `query_id`: Foreign key referencing the **Query**.
- `action_items_list_id`: Foreign key referencing the **ActionItemsList**.
- `updated_by`: Reference to the user who last updated the record.
- `created_at`, `updated_at`: Timestamps for auditing.

4.4.1.7 ScreenComponent

The `ScreenComponent` model represents the configurable visual elements that make up a screen variant within the application.

A screen component now includes the following key attributes:

- `id`: Primary identifier of the screen component.
- `name`: Name of the screen component, which is the concatenation of the `widget` and `main_id` attributes;
- `x` and **y**: Positional coordinates for layout placement within a screen;
- `cols` and **rows**: Dimensional values defining the component's size in a grid layout;
- `widget`: Type of widget represented by the component (Task, List, or Form);

- `screen_variant_id`: Reference to the parent `ScreenVariant` in which this component is contained;
- `action_items_list_id`: Optional reference to an `ActionItemsList`, enabling the attachment of table actions to the list widgets;
- `main_id`: Reference to the entity associated with this component. `Process` if it's a task widget,`Query`, if it's a list widget or `Form` if it's a form widget;
- `updated_by`: Reference to the user who last updated the record.
- `created_at`, `updated_at`: Timestamps for auditing.

4.4.1.8 Migrations

Migrations in Laravel function as version control for the database schema, allowing developers to evolve the database structure in a controlled, step-by-step manner. Each created migration defines the structure of a single table, specifying its columns, keys, and relationships [68].

A migration class contains two methods:

- `up` – applies changes to the database, such as creating new tables, adding columns, or defining relationships.
- `down` – reverses the changes made by the `up` method, allowing the database to return to its previous state.

In this work, migrations were used to create tables for each new entity and to update the existing `screen_component` table by adding the `action_item_list_id` attribute. Listing 3 shows an example of an `up` method that creates the `ActionItem` table, defining its columns and foreign key relationships.

```
public function up()
{
    Schema::create('action_item', function (Blueprint $table) {
        $table->increments('id');
        $table->string('icon');
        $table->unsignedInteger('action_rule_id');
        $table->unsignedInteger('updated_by')->nullable();
        $table->timestamps();

        $table->foreign('action_rule_id')->references('id')->on('action_rule')
            ->onDelete('no action')->onUpdate('no action');
        $table->foreign('updated_by')->references('id')->on('users')
            ->onDelete('no action')->onUpdate('no action');
    });
}
```

Listing 3: Up method of the migration that creates the `ActionItem` table

4.4.2 Controllers

In Laravel, controllers act as intermediaries between client requests and the application's business logic. They handle HTTP requests, validate input, and orchestrate responses. In this work, we defined new controllers to manage Table Actions and Lists of Table Actions and made changes to the Query and User App controllers.

To standardize API responses, all controllers return data wrapped in API Resources defined by us. Specifically, `ActionItemResource` and `ActionItemsListResource` ensure a consistent output format across endpoints, including multilingual attributes. This decouples the database schema from the API response, allowing the inclusion of computed fields, such as localized names and related entity data.

4.4.2.1 ActionItemController

The `ActionItemController` manages individual Table Actions. Its key endpoints are:

- `GET /actionItems (index)`: Retrieves all Table Actions, including their name, tooltip, and the associated action rule's transaction type and entity type.
- `POST /actionItems (store)`: Creates a new Table Action with its base attributes and multilingual text.
- `GET /actionItems/id (show)`: Retrieves a specific Table Action by its ID, along with the same information mentioned in the first endpoint.
- `PUT /actionItems/id (update)`: Updates an existing Table Action, including multilingual text table.
- `DELETE /actionItems/id (destroy)`: Permanently deletes a Table Action, its multilingual text, and removes pivot table associations.

Listing 4 shows an example of how to retrieve data from multiple tables in the index method of the `ActionItemController`.

```php
public function index(Request $request)
{
    $userLangId = $request->user()->language_id;

    $items = DB::table('action_item')
        ->join('action_rule', 'action_item.action_rule_id',
            '=', 'action_rule.id')
        ->join('transaction_type', 'action_rule.transaction_type_id',
            '=', 'transaction_type.id')
        ->join('ent_type', 'action_rule.transaction_type_id',
            '=', 'ent_type.transaction_type_id')
        ->select(
            'action_item.*',
            'action_rule.transaction_type_id as transaction_type_id',
            'action_rule.t_state_id as t_state_id',
            'transaction_type.process_type_id as process_type_id',
            'ent_type.id as ent_type_id')
        ->get();

    foreach ($items as $item) {
        $this->getActionItemText($item, $userLangId);
    }

    return ActionItemResource::collection($items);
}
```

Listing 4: Index method of the `ActionItemController`

4.4.2.2 ActionItemsListController

The `ActionItemsListController` handles lists of Table Actions. Its endpoints allow clients to create, retrieve, update, and delete these lists:

- `GET /actionItemsLists (index)`: Retrieves all lists, including multilingual names/descriptions and their associated Table Actions and queries.
- `POST /actionItemsLists (store)`: Creates a new list with multilingual text and links to selected Table Actions and queries.
- `GET /actionItemsLists/id (show)`: Retrieves a specific list with all associated multilingual attributes.
- `PUT /actionItems/id (update)`: Updates an existing list, including multilingual text table.
- `DELETE /actionItems/id (destroy)`: Permanently deletes a List of Table Actions, its multilingual text, and removes pivot table associations.

4.4.2.3 QueryController

The `QueryController` manages queries and exposes endpoints for creating, retrieving, updating, and deleting them.

A new method was introduced to retrieve only the queries that are linked to at least one List of Table Actions. This endpoint performs a join with the `QueryHasActionItemsList` table to filter queries with existing associations. Unlike a standard query retrieval, it also returns the associated Table Actions and their relevant information.

Similar to other `GET` operations, this endpoint preserves multilingual attributes for both the queries and the Table Actions, ensuring that the response respects the user's language preferences.

The primary purpose of this endpoint is to support efficient filtering on the back-end, allowing citizen developers to select queries when adding a new table with embedded Table Actions without making multiple API calls.

The endpoint is exposed as:

- `GET /queries/with_action_items_list`: Returns queries linked to at least one Table Actions list, including all associated Table Actions and multilingual attributes.

4.4.2.4 User App Controller

The `UserAppController` manages user applications and exposes endpoints for creating, retrieving, updating, and deleting them.

With the addition of the `action_items_list_id` attribute to the `ScreenComponent` model, the creation and update methods of this controller were modified to incorporate this new field. This allows each screen component within a user application to reference a specific List of Table Actions.

Both methods process the widgets or components of each screen, creating new ones or updating existing ones. Therefore, the `action_items_list_id` must be included in this workflow to ensure it is saved in the database.

4.4.2.5 Integration with Multilingual Logic

The `getMultilingualConceptName` trait provides a centralized method to retrieve the name or textual attribute of a concept in the preferred language of the currently authenticated user.

When a controller returns an entity, this helper method is called to include relevant multilingual information that matches the current user's primary language in the returned data.

Specifically, the method works as follows:

- It first attempts to retrieve the attribute for the user's selected language.

- If no record exists for the user's language, it falls back to the system's default language (as configured by `app.fallback_locale`).

- If neither the user's language nor the fallback language exists for the concept, it returns the first available entry that has not been soft deleted.

- Optionally, it can return both the language ID and the attribute, and/or prepend the language abbreviation to the value to provide context about which language the text belongs to.

The method ensures that all entities exposed via the API respect the user's language preferences while maintaining a safe fallback mechanism, preventing missing data in multilingual contexts.

4.5 Front-end changes

The front-end of the project is implemented in Angular and consumes the back-end API endpoints. It presents, manipulates, and allows interaction with the data through the user interface, leveraging model interfaces and services to ensure consistency and type safety when handling the server's responses.

4.5.1 API Services and Model Interfaces

Model interfaces were introduced to replicate the structure of the resources retrieved from the back-end, defining the expected properties of each entity.

By using these interfaces, the front-end code can enforce type safety and ensure consistency when handling data received from the server.

To interact with the back-end, `ApiServices` were implemented. Each service is responsible for performing operations on a specific entity by making HTTP requests to the endpoints exposed by the corresponding back-end controllers.

For example, the `ActionItemApiService` provides methods to retrieve all action items, get a single action item by its identifier, create new action items, update existing ones, and delete items. These services also handle HTTP headers, including authentication tokens, and manage error handling using observable streams with `retry` and `catchError` operators.

Similar to the changes made to the back-end data models and controllers, we also had to modify the query model to optionally include the list of table actions when a call is made to the new endpoint that returns queries and their associated lists of table actions and to it's API service by adding a new method to make a GET request to that new endpoint.

4.5.2 User Interface Design

4.5.2.1 Dashboard changes

A new navigation item has been added to the application sidebar, providing direct access to the Table Actions Management page from any other page of the platform. This new entry appears as a sub-item under `Action Rules`. Selecting this item opens the Table Actions Management page, allowing users to view, create, edit, or delete Table Actions and Lists of Table Actions.

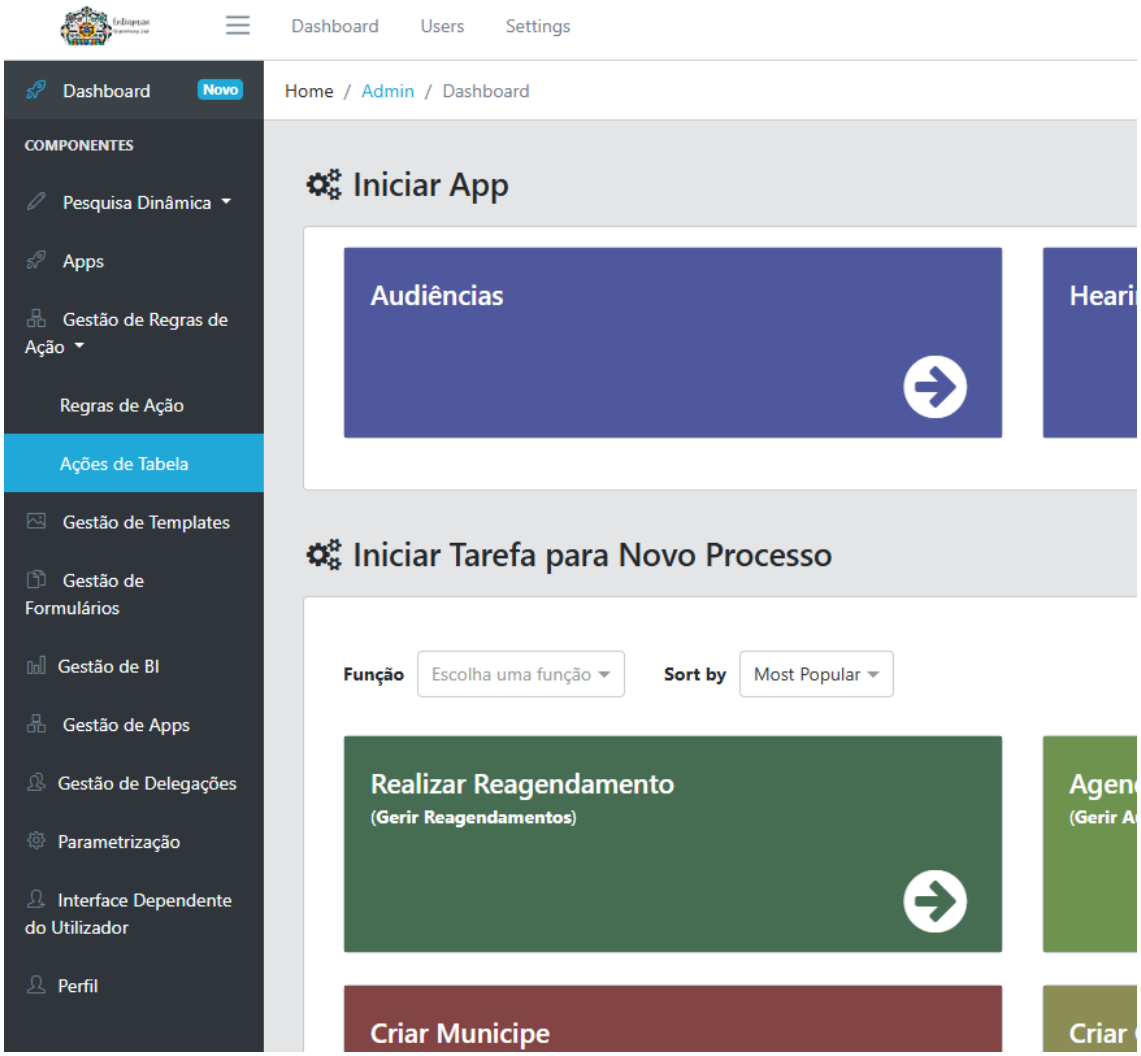

Figure 4.3: Sidebar with the new Table Actions Management menu item located under the Action Rules section.

4.5.2.2 Table Actions Management Page

To facilitate the administration of this new functionality, we created a management page within the platform.

This page contains two main tables displayed in separate collapsible cards. The first card includes the Table Actions table, while the second card contains the Table Actions Lists table. Each card header shows the table's title and a button to create a new entity, which opens a modal form. The modal allows users to enter all necessary information to create a new Table Action or List of Table Actions; the details of these modals are discussed in the following subsections.

The card headers also display the total number of items currently shown in each table, providing the user with an overview of the available actions or lists. Users can expand or collapse each card by clicking on the header, giving flexibility in displaying multiple tables on the same page.

The tables are implemented using the ag-Grid Angular component. Both tables are configured with features such as pagination, sortable and filterable columns, automatic height adjustment, and row animations. The Table Actions table displays all individual Table Actions, including their identifiers, localized names, transaction types, states, tooltips, icons, timestamps for creation and last update, and the language of textual attributes. A dedicated column contains interactive buttons for available operations over each row, including editing or deleting a Table Action.

The Table Actions Lists table shows each list's identifier, name, description, associated Table Actions, linked queries, timestamps, and the language of textual attributes. Similarly, a dedicated column contains buttons for available actions, such as editing or deleting the list.

We created custom cell renderers for each column whose content exceeds simple text. These include the icon column, the columns displaying associated Table Actions and linked queries, and the columns containing available action buttons. These renderers enable the table to display HTML elements, interactive components, or apply special formatting within individual cells.

For example, the icon column displays a visual representation of the Table Action, while the associated Table Actions and queries columns present aggregated or structured information rather than plain text. The available actions columns provide interactive buttons for editing or deleting each row.

Both tables support user interaction, enabling efficient organization and location of data. Pagination allows users to navigate through large datasets, while sortable and filterable columns facilitate searching and sorting by relevant attributes.

All delete operations for both `ActionItems` and `ActionItemsLists` are protected by a confirmation prompt. When a user attempts to delete an entity, a dialog appears requesting explicit confirmation of the action. This mechanism prevents accidental removal of table actions or lists, ensuring that deletions are intentional and allowing users to reconsider before the operation is finalized.

Figure 5.3 shows the Table Actions Management page, displaying both tables with several entities already created.

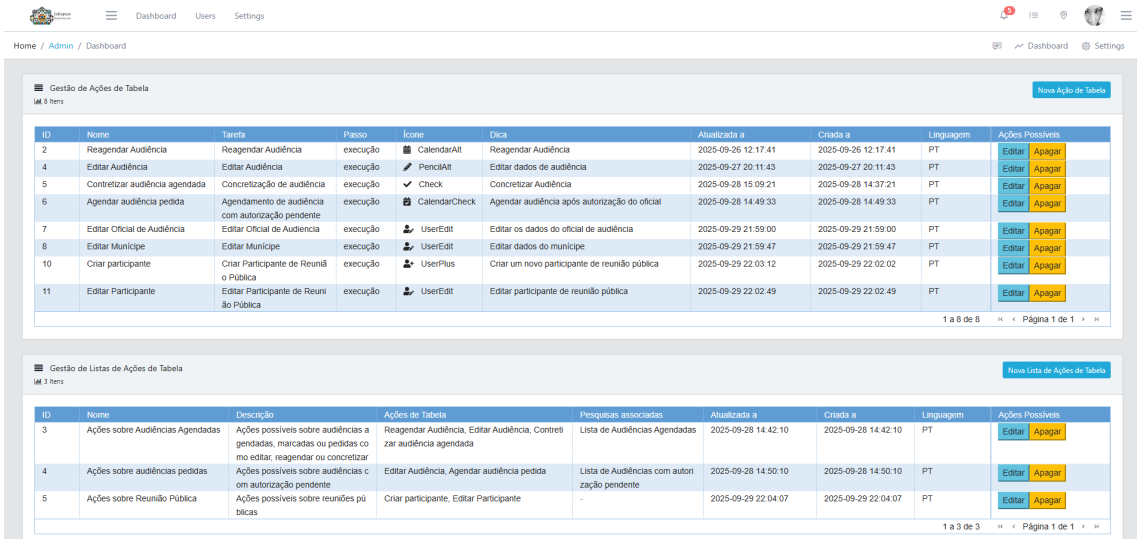

Figure 4.4: Table Actions Management Page with separate tables for Table Actions and Lists of Table Actions, showing all relevant columns.

4.5.2.3 modal to create and edit Table Actions

From the Table Actions table, users can open a modal form to create a new `Table Action` by clicking the *Create* button located in the table header. The same modal is also used when editing an existing Table Action. In this case, the modal is pre-populated with the selected row's data, allowing users to modify any of the existing attributes.

The modal form contains fields for all relevant information of a Table Action, including the localized name, the associated `ActionRule`, an icon representing the action, and a tooltip that provides descriptive text. The `ActionRule` selection utilizes a drop-down component that displays both the transaction type and transaction state associated with each action rule, enabling users to clearly identify the intended action. The icon field also uses a drop-down with search capability, showing a real-time preview of the selected icon alongside a readable label.

A preview section is displayed dynamically when all required fields are filled. This section displays the visual representation of the Table Action, which will turn off when enabled within tables, including the icon inside a circular button and the action name below it. The tooltip assigned to the action is displayed when the mouse hovers over the button, allowing users to verify how it will behave in the interface.

The modal enforces input validation by disabling the *Save* button until all required fields are filled. When the user saves the form, the system either creates a new Table Action or updates the existing one via API calls to the endpoints we created. Success and error notifications are displayed for both operations, ensuring users are informed of the result of their action.

Figure 4.5 shows the modal that appears when the user clicks the button to create a new Table Action. The modal also displays a message indicating that a preview of the

action button will be available once all required fields have been completed.

Figure 4.5: Modal form for creating and editing a Table Action.

4.5.2.4 modal to create and edit Lists of Table Actions

From the Table Actions Lists table, users can open a modal form to create a new `ActionItemsList` by clicking the corresponding Create button. In this modal, users provide a name and a description for the list, select a base entity type to ensure compatibility of associated queries and table actions, and choose which existing `ActionItems` and queries should be linked to the list.

If the modal is opened through the Edit button, the form is pre-populated with the existing details of the selected list, including previously associated queries and table actions. Users can then update any of these fields and save the changes.

The modal dynamically updates the available queries and table actions when the base entity type is changed. Selecting a new base entity type clears any previously selected queries, table actions, and filters, and refines the available options to include only those compatible with the selected entity type. This behaviour is to ensure consistency and prevent users from linking incompatible queries or table actions to the list, which could lead to undesired behaviors.

Figure 4.6 illustrates the modal for creating and editing an `ActionItemsList`.

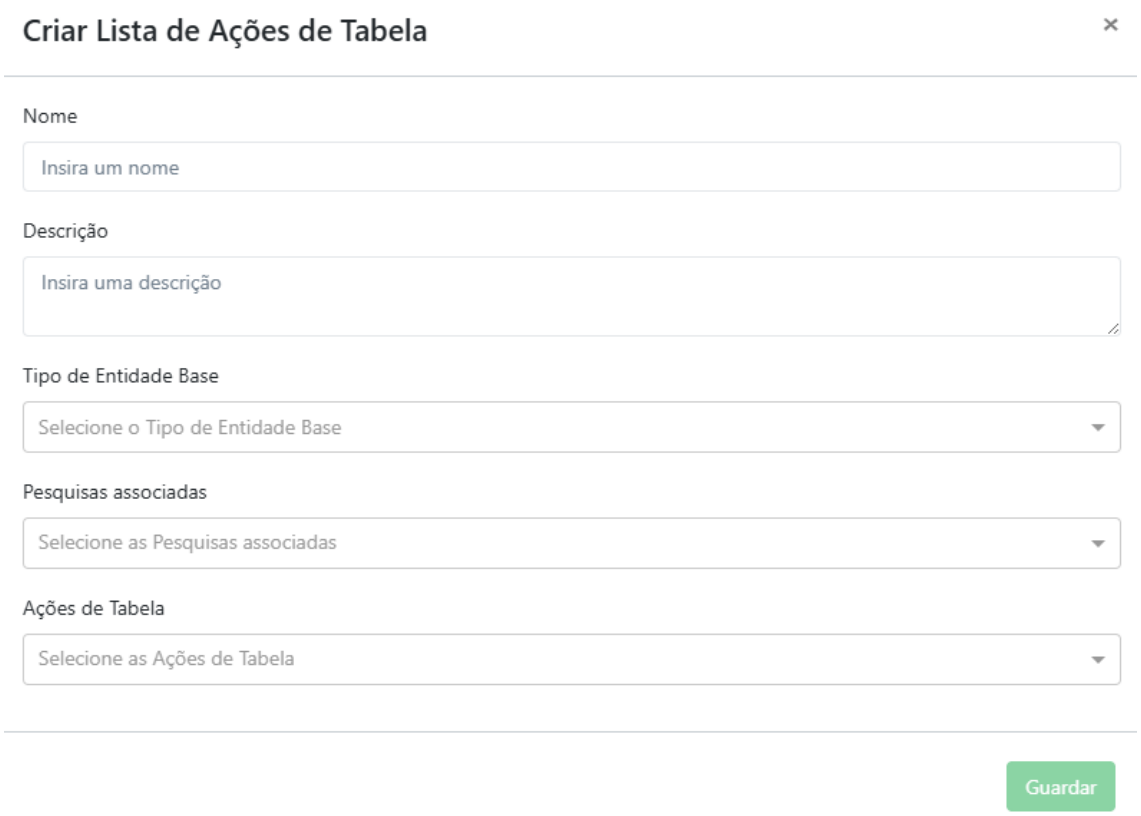

Figure 4.6: Modal form for creating and editing a List of Table Actions.

4.5.2.5 User App Modal changes

The **User App Modal**, which is displayed when dragging a new screen widget onto an application's screen, was also updated.

Previously, when adding a **List** widget—which renders as a table populated with the results of a given query—the modal only included a single selection field for choosing the Query (Figure 4.7).

Figure 4.7: Previous version of the User App Modal

With the introduced changes, a new selection field was added to allow the user to specify whether the list should include **Table Actions**. When this option is set to affirmative, an additional field becomes visible after the query selection, enabling the user to choose from the available **Action Item Lists** associated with the selected Query (Figure 4.8).

Figure 4.8: Updated version of the User App Modal with new selection fields

4.5.2.6 User App and User App Render Changes

The **User App** page (Figure 4.9) is the environment where citizen developers can create and edit applications. From this page, users can add and configure screens, widgets, screen variants, and role-based access, thereby shaping how the application behaves and who can interact with it.

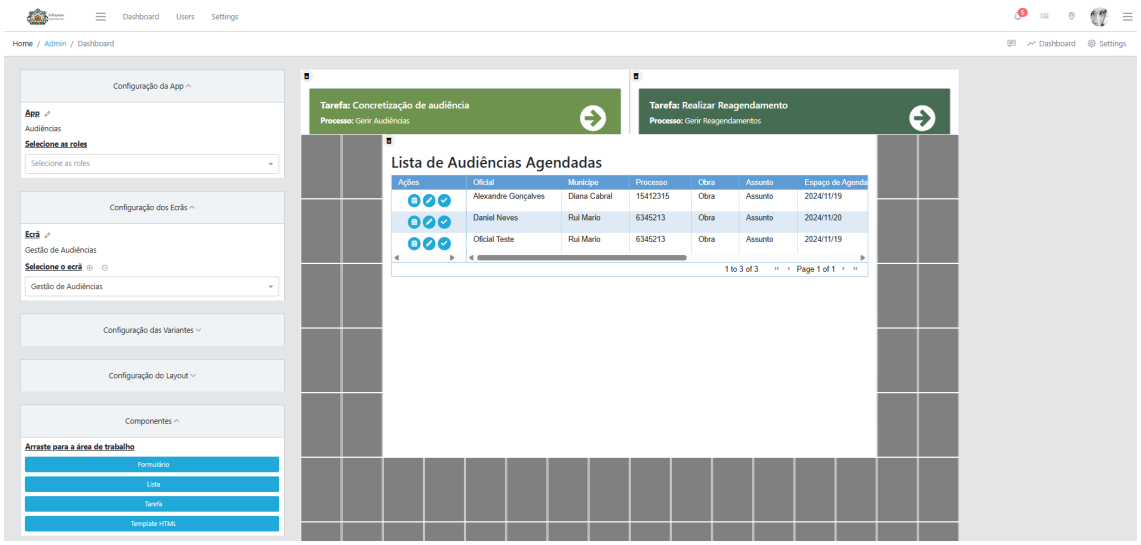

Figure 4.9: User App Page showcasing an application table actions

The **User App Render** page (Figure 4.10), in contrast, is where the resulting applications are actually displayed to end-users. On this page, users interact with an application designed by a citizen developer, navigating through screens and executing the operations available in its interface.

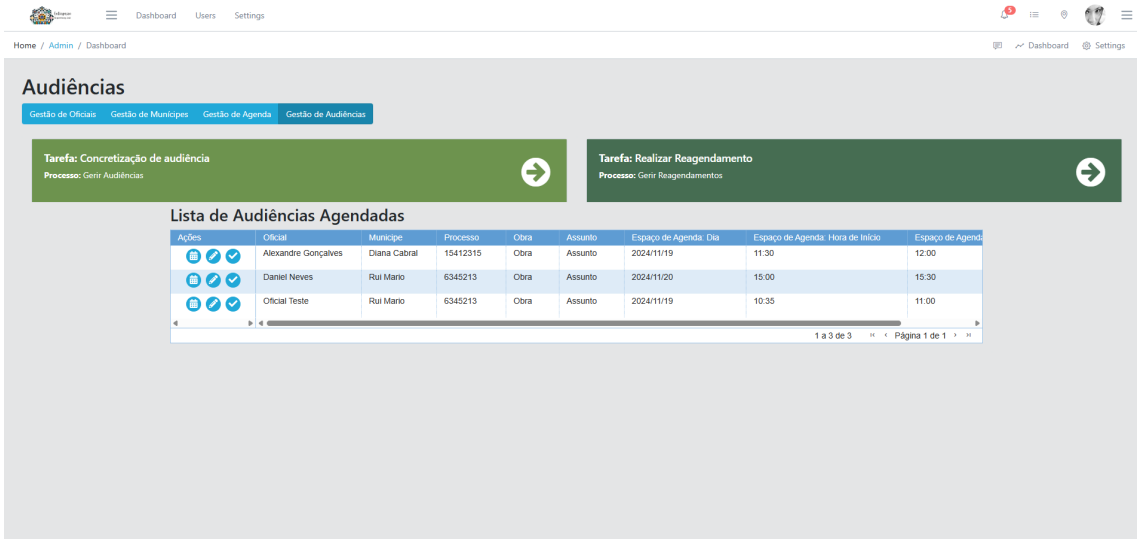

Figure 4.10: Rendered User App with table actions

In both pages, we introduced modifications to support the retrieval of **Table Actions** when a table widget is linked to an Action Item List. For this purpose, we implemented a custom cell renderer in both pages to define how available actions are displayed in each row of the table.

In the **User App** page, the cell renderer only displays the action buttons with their corresponding icons and tooltips. In contrast, the **User App Render** page extends this

behavior by attaching an `onClick` event to each button, enabling the execution of the corresponding Action Item process directly from the table row.

When a user clicks on a button in the available actions column of a table, the method `startActionItemProcess` is triggered. The method first requests the full definition of the Action Item from the back-end. This definition includes the associated action rule and its `process_type_id` and `transaction_type_id`, which uniquely identify the kind of process that can be started.

Using these identifiers, a call is made to `getProcessesAvailableToInitiateTask`, which returns a collection of process instances available for initiation. The process instance whose data matches the row's data is picked and used to retrieve the actual process definition with `getProcessByTransactionTypeId`.

Finally, the method `startTaskOfInitProcess` is invoked on the retrieved process, and its execution is initiated. At this stage, the system automatically skips steps for which the required information has already been obtained from the row data, prompting the user to fill in only the remaining necessary details to complete the operation.

For instance, in operations such as editing, scheduling, or rescheduling a pending hearing, the first step typically requires the user to select the relevant hearing process instance. By initiating the operation directly through a table action in a hearings table, this selection step becomes redundant, as the correct hearing is already determined from the context of the row.

5 Evaluation

5.1 Description

To evaluate the newly implemented table actions functionality, it was essential to test it under conditions that closely resemble real-world usage. Instead of manually creating an application with users, action rules, entities, and queries from scratch, we used a pre-populated database. This approach ensured that the system contained realistic and consistent data, including a fully configured application of the MHP, which served as the basis for the evaluation.

The evaluation focuses on testing the table actions management and executing one of the core operations of the MHP application: rescheduling a hearing.

The evaluation process was guided by the requirements defined in Chapter 3. Specifically, the functional requirements related to *Table Actions Management*, *User App Management*, and *User App Render* were validated through practical use cases. In parallel, the non-functional requirements of *Usability*, *Performance*, *Scalability*, and *Accessibility* provided the basis for qualitative assessment of the feature.

Thus, the evaluation not only tested the technical correctness of the implementation but also verified how well the feature aligned with the expectations and workflows expressed in these requirements and user stories.

5.2 Setup

In the **Setup** section, we describe the steps taken to configure the system with table actions, including not only rescheduling, but also scheduling a pending hearing, holding a scheduled hearing, and editing both scheduled and pending hearings.

This section was designed to validate the implementation against the defined user stories, specifically user stories 1, 2, 3, 6, 7, and 10 (see 3.3).

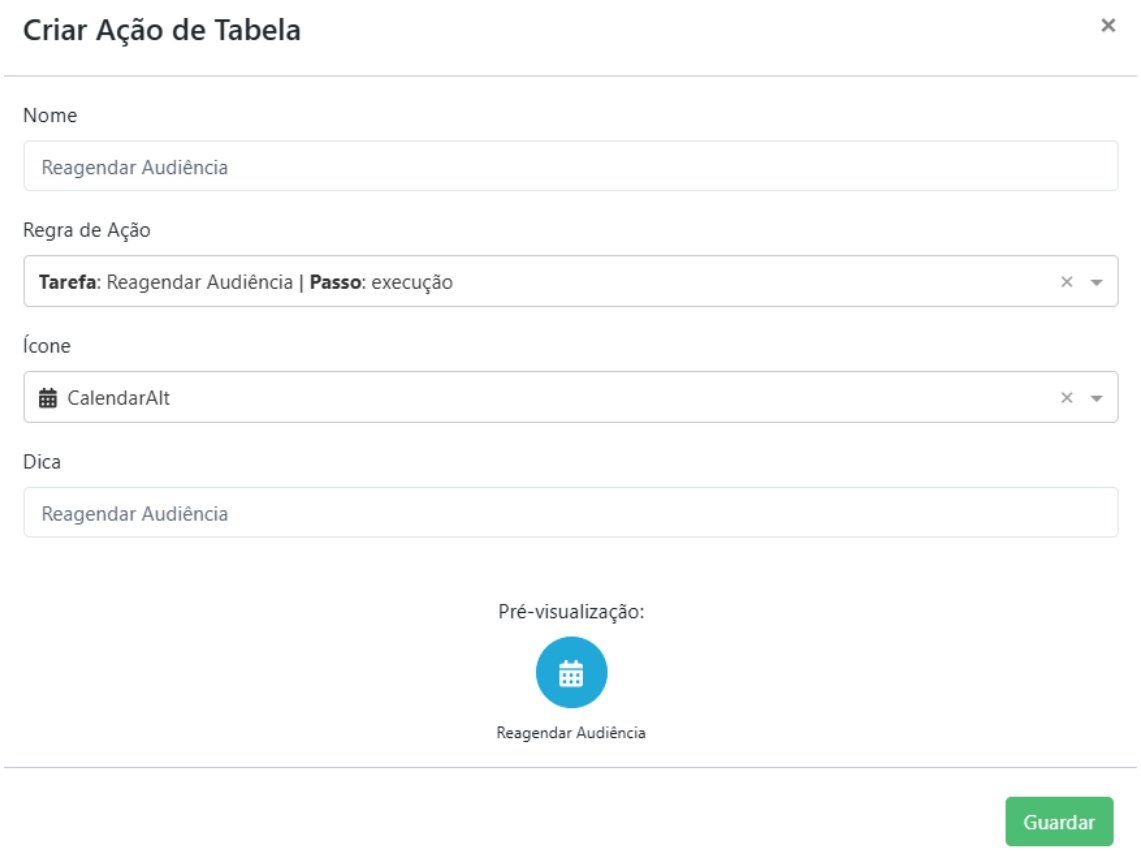

Figure 5.1: Creation of a new table action to reschedule a hearing

5.2.1 Access the Table Actions Management Page

To access the Table Actions Management page, the citizen developer must first navigate to the dashboard. From the side-bar menu, selecting *Action Rules Management* expands two options: *Action Rules* and *Table Actions*. By clicking on *Table Actions*, the user is redirected to the Table Actions Management page. From this interface, the citizen developer gains access to the existing table actions and lists of table actions, along with the basic operations to create, edit, and delete them.

5.2.2 Creating the table actions

The required table actions were created in the Table Actions Management component of the DISME System Modeler.

This process involved creating individual `ActionItems`, specifying the associated action rule, tooltip, and icon for each one.

Figure 5.1 shows the creation of the table action to reschedule a hearing. The actions to schedule a pending hearing, hold a hearing, and edit a hearing were created in the same way, but with the appropriate information. If mistakes occur, users can edit existing table

Figure 5.2: Creation of a new list of table actions for scheduled hearings

actions using the same form, which will be pre-filled with the saved data.

5.2.3 Creating the lists of table actions

The created table actions were then grouped into two lists: one for scheduled hearings and another for pending hearings. The first list contained the actions *Reschedule Hearing*, *Edit Hearing*, and *Hold Hearing*, while the second list contained *Schedule Pending Hearing* and *Edit Hearing*.

Figure 5.2 illustrates the creation of the list of table actions for scheduled hearings, showing the association with the scheduled hearings query and the selected table actions.

Users must specify a base entity type for each list to ensure compatibility between queries and action rules. When the entity type is changed, the queries and table actions fields are cleared to prevent inconsistencies. As with individual actions, existing lists can also be edited through the same form, pre-populated with the saved data.

After defining all actions and lists, the Table Actions Management page resembled Figure 5.3, displaying the created table actions and their respective lists.

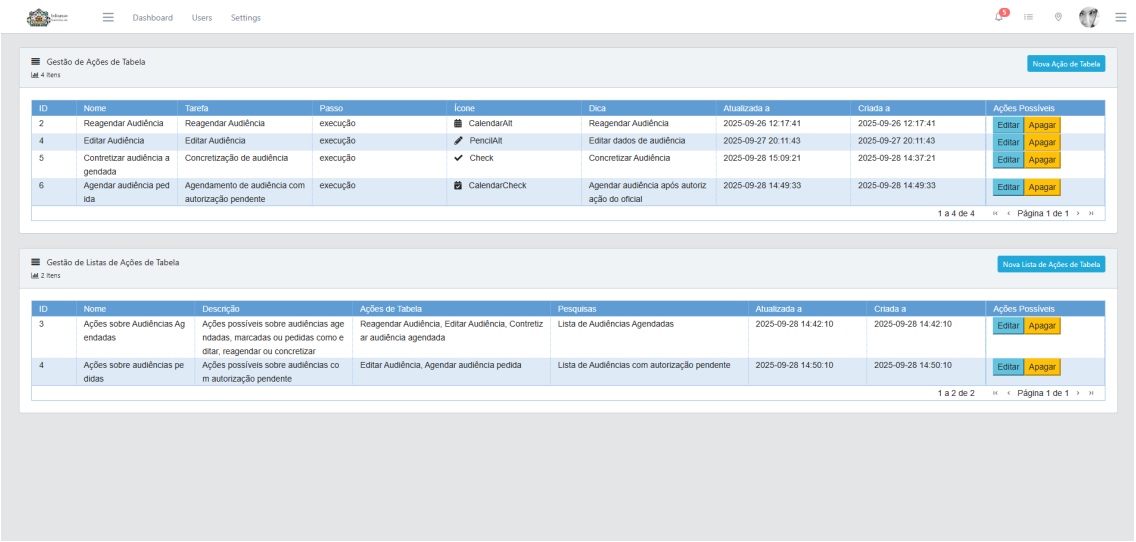

Figure 5.3: Table Actions Management Page after creating the table actions

Figure 5.4: Adding a table with scheduled hearings and a list of table actions

5.2.4 Adding the tables with table actions

The final step was to integrate the new action lists into the MHP application. This was done in the Applications Management component by editing the Hearings Management page.

On this page, we can delete the existing tables and add a new table by dragging the "list" widget to the desired position. This will open a form (Figure 5.4) that asks the user whether they want the table to include table actions or not, as well as select the desired query. If the user selects that they want to include table actions, a new select field is displayed to choose one of the lists associated with the specified query.

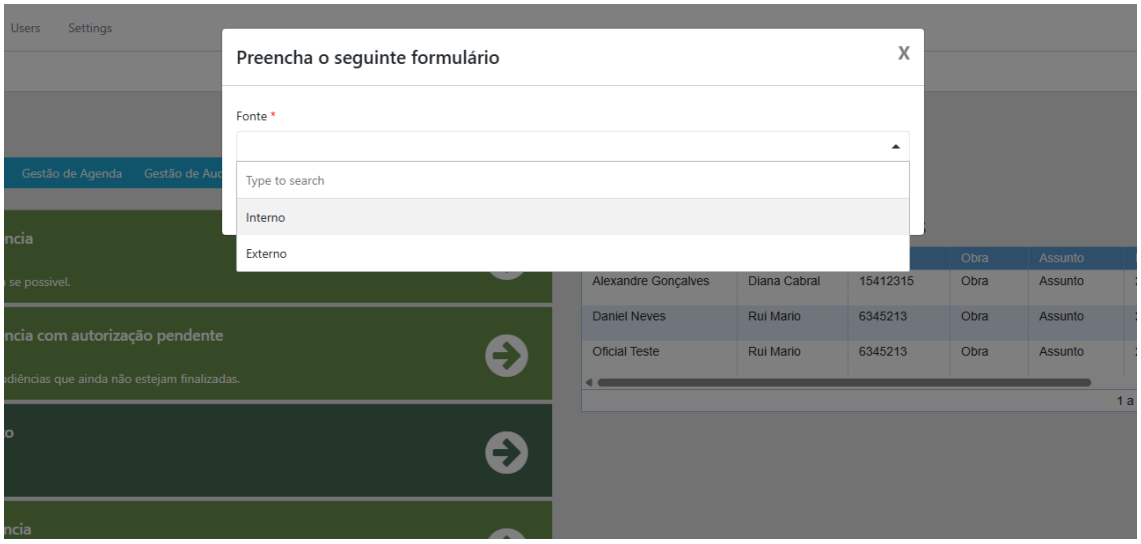

Figure 5.5: Original Hearing Management Page

We executed these steps for both the table containing scheduled hearings and the table containing pending hearings, ensuring that both include the corresponding lists of table actions.

5.3 Comparison between old and new workflow

In this section, we compare how the same operation — rescheduling a hearing requested by a citizen — is performed in both the original version of the application and the updated version with table actions.

The comparison is grounded in user stories 11 and 12, which describe an end-user interacting with table actions within the application.

To provide a concrete basis for the evaluation, we defined a specific user story that guides the execution of the operation in both versions: a citizen named "Rui Mário" has requested the rescheduling of his hearing. The actor in this scenario, such as a hearings clerk, must locate the corresponding hearing and complete the rescheduling process.

This specific scenario and operation were chosen because they encompass multiple selection and verification steps, offering a clear illustration of the differences between the two workflows.

5.3.1 Rescheduling a hearing without table actions

In the original version of the MHP, the rescheduling process involves several steps.

First, the user needs to click on the rescheduling operation from the available operations on the left of the page (Figure 5.5) to initiate the process, which opens a modal and prompts the user to indicate the source of the request: external if requested by a citizen or internal if for internal reasons, for example the assigned officer won't be able to attend (Figure 5.6).

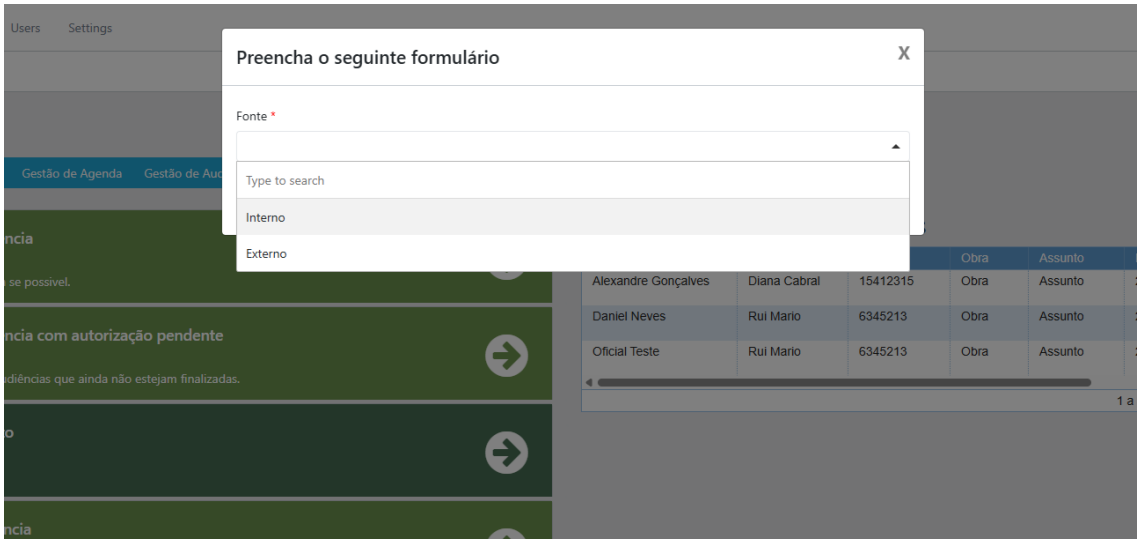

Figure 5.6: Selection of the source of rescheduling

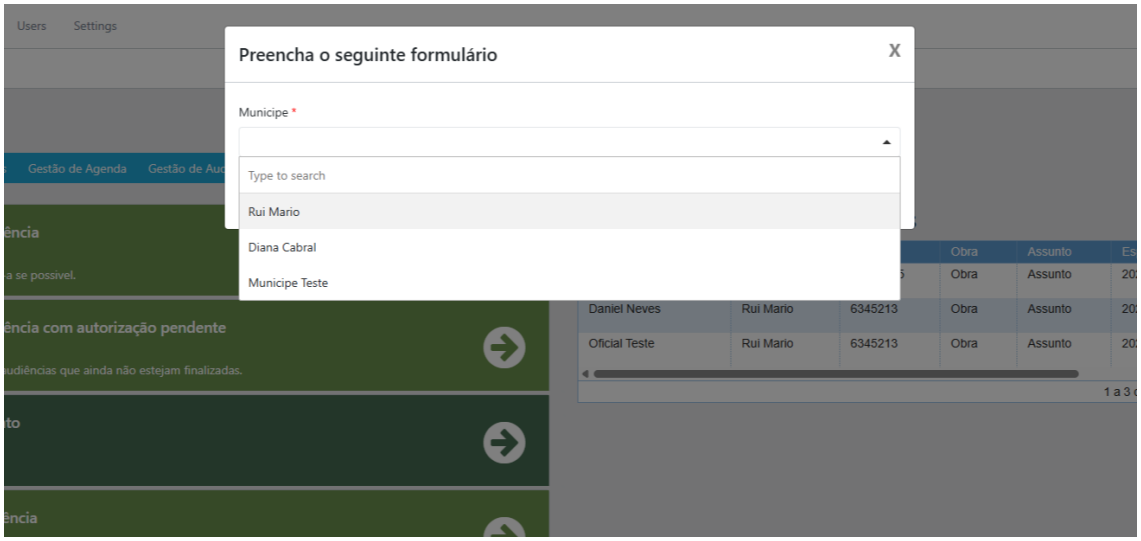

Figure 5.7: Selection of the citizen who requested the rescheduling

In the user story we follow, the citizen is the one requesting rescheduling, so the selected source is external.

Once this is done, the user must select the citizen requesting the change. In our user story, this requires locating "Rui Mário" among the list of citizens (Figure 5.7).

Once the citizen is chosen, the user selects the specific hearing to be rescheduled from the list filtered by that citizen (Figure 5.8).

Only after these selections can the user proceed to the final step, which involves providing the new scheduling details, selecting a new date and time slot, and confirming the operation by submitting (Figure 5.9).

This workflow, although functional, requires multiple intermediate selections before

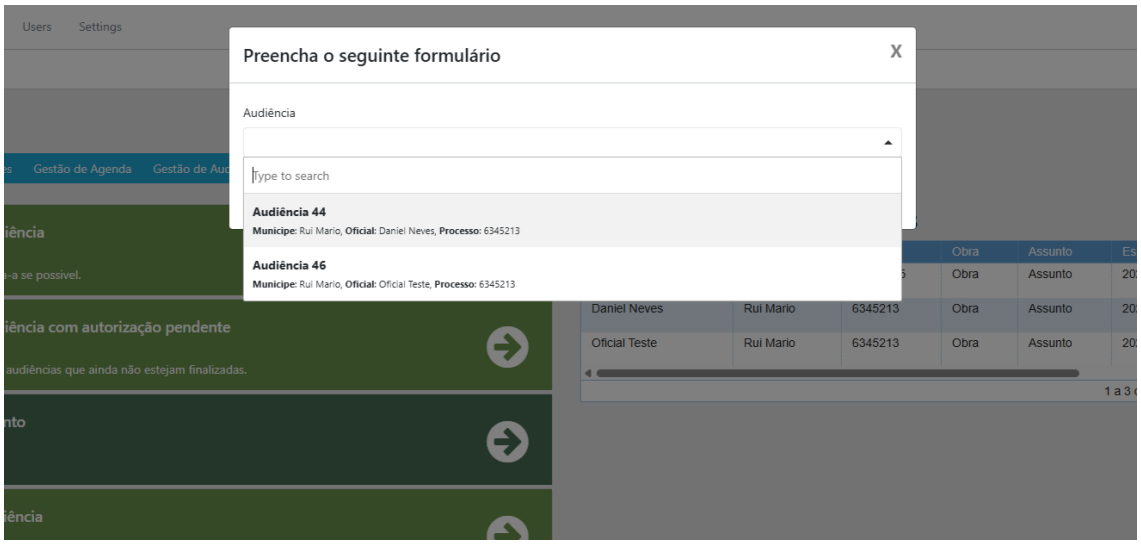

Figure 5.8: Selection of the hearing for rescheduling

Figure 5.9: Filling information about the rescheduling

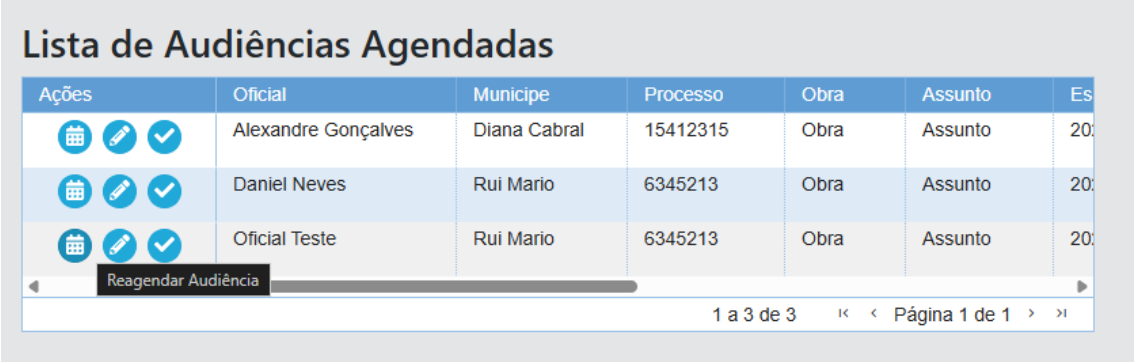

Figure 5.10: Table of scheduled hearings with table actions

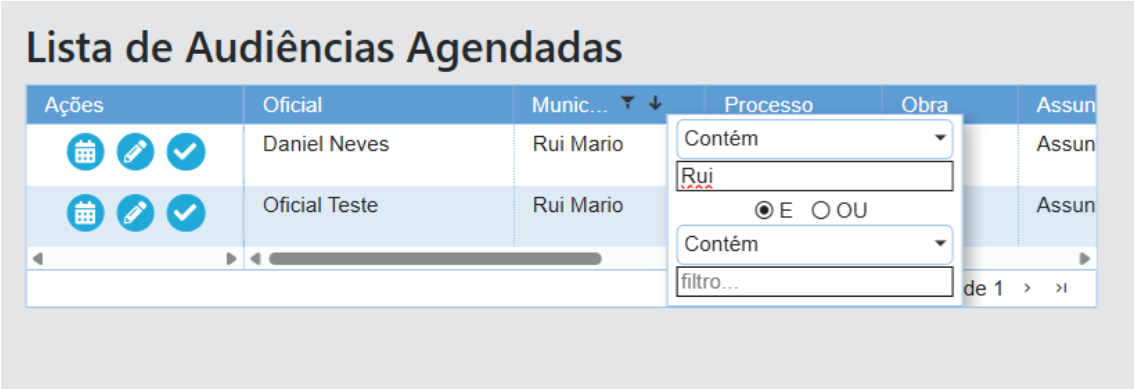

Figure 5.11: Filtering scheduled hearings by citizen

reaching its final step, thereby increasing the number of steps and the time needed to complete it.

5.3.2 Rescheduling a hearing using the table actions

With table actions enabled, the same user story is executed more directly from the scheduled hearings table.

The table actions feature introduced a more direct interaction to initiate the rescheduling process, where the relevant reschedule action is now embedded directly into each row of the scheduled hearings table, as shown in Figure 5.10.

The first step is to locate the hearing for "Rui Mário", either by visually scanning the scheduled hearings table or by using the table's built-in filter feature to show only rows for that citizen (Figure 5.11). The user then clicks the *Reschedule Hearing* button embedded in the corresponding row. By doing so, the system automatically passes the hearing and citizen data to the rescheduling process, so the user does not need to re-select the citizen or the hearing.

The user selects the source of the rescheduling (Figure 5.6, in this case, external because it's a citizen requesting it.

Since the hearing to be rescheduled has already passed, after selecting the source, all that is left is to fill in the rescheduling information, pick a new hearing time slot, and submit (Figure 5.9).

The old workflow remains available, but does not auto-populate the hearing. Compared to the original flow, the table actions approach eliminates the intermediate selection steps of identifying the citizen and the hearing, thereby accelerating the rescheduling task.

5.4 Questionnaire about perceived usability

Since it was not possible to conduct usability testing with real DISME users within the timeframe of this thesis, a questionnaire was prepared and distributed among a group of peers, including university colleagues, friends, and co-workers.

The questionnaire included demographic questions, allowing us to identify possible trends related to gender, as well as a section to help us determine how they position themselves among each of the five cognitive facets of GenderMag.

It then presented participants with both versions of the application, the original workflow and the updated version with table actions, displaying the same step-by-step demonstrations of the rescheduling operation.

After showcasing the workflow without table actions, participants were asked to indicate their level of agreement or disagreement on a scale of one to five with a series of statements about the application and the overall execution process of tasks. After showcasing the new workflow through the use of table actions to reschedule a hearing, participants were asked comparative questions to capture their perceptions of each approach and preference.

These comparative questions focused on four key aspects:

- Which version seemed more intuitive to use?
- Which version seemed to allow them to complete tasks more efficiently?
- Which version seemed less prone to errors?
- Which version seemed to provide a more satisfying overall experience?

The goal of this questionnaire is to provide an external perspective on the perceived usability improvements introduced by table actions, even if it does not replace complete usability testing with real DISME users.

5.4.1 Results

5.4.1.1 Demographic

A total of 43 participants responded to the questionnaire, of whom 15 identified as female and 28 as male. The age distribution shows that the vast majority of respondents (95.3%)

were between 18 and 34 years old, reflecting a predominantly young adult population.

In terms of occupation, most participants were students, although some reported different professional backgrounds.

We also asked participants if they had ever used a LCP before, and 76.7% of participants reported that they had never used such platforms before.

5.4.1.2 Cognitive Facets

The cognitive facet section results were analyzed by comparing responses between male and female participants across the five GenderMag cognitive facets: motivation, information processing style, computer self-efficacy, risk aversion, and tinkering. The goal was to assess whether the findings align with existing research in the field.

For each facet, participants were presented with three options: two reflecting opposing positions and one representing an intermediate stance.

Table 5.4.1.2 summarizes the results, which are further analyzed below.

Motivation: Female participants leaned more strongly towards using technology when it helps them accomplish tasks efficiently (53.33%). In contrast, male participants showed more varied responses, with a notable share indicating that they use technology either for enjoyment (28.57%) or primarily to achieve functional goals (35.71%). This confirms prior research, which suggests that women tend to be more task-oriented, while men are more likely to use technology for enjoyment and ease of interaction.

Information Processing Style: Unexpectedly, female participants were more willing to try the first available option and backtrack if needed (66.67%), whereas only 39.29% of males selected this approach. Conversely, a higher proportion of males (25.00%) than females (13.33%) preferred to gather a complete understanding of the task before starting. This result diverges from the literature, which generally reports that women tend to be more comprehensive in their information processing style, while men are more selective and exploratory.

Computer Self-Efficacy: Both groups showed confidence in their ability to figure out unfamiliar software, but males expressed higher levels (64.29%) compared to females (46.67%). Furthermore, a larger share of females reported doubts about their ability to complete tasks with new technology (26.67% vs. 14.29%). These findings align with existing research showing that women, on average, report lower confidence in using unfamiliar technology.

Risk Aversion: Two-thirds of female participants (66.67%) reported being cautious about clicking on uncertain options, compared to 39.29% of males. Additionally, more males (32.14%) indicated that they did not mind trying things out without certainty, compared

to 26.67% of females. This is consistent with previous studies suggesting that women tend to be more risk-averse.

Tinkering: Males reported higher levels of tinkering (53.57%) compared to females (40.00%), while females were more evenly divided between tinkering and step-by-step learning approaches. Notably, females also reported reflecting more often on what they discovered after tinkering (20.00% vs. 10.71%). These findings align with the literature, which suggests that women are less likely to experiment playfully with new features, but when they do, they tend to reflect more deeply on the process.

Table 5.1: Questionnaire results by Gender and Cognitive Facet

Facet	Option	Females	Males	Total
Motivation	I primarily use new features if they help me accomplish my task more efficiently.	53.33%	35.71%	41.86%
	I often explore technology because I find it fun and enjoyable, even if not strictly necessary for my tasks.	40.00%	28.57%	32.56%
	When trying out new software, my main focus is on functionality and achieving my goals.	6.67%	35.71%	25.58%
Information Processing Style	I prefer to gather a full understanding of the task before trying a new feature.	13.33%	25.00%	20.93%
	I am comfortable trying the first available option and backtracking if needed.	66.67%	39.29%	48.84%
	Before clicking something new, I usually want to see the bigger picture of how it works.	20.00%	35.71%	30.23%
Computer Self-Efficacy	I feel confident using unfamiliar software features.	26.67%	21.43%	23.26%
	If I encounter difficulties, I believe I can usually figure out a solution on my own.	46.67%	64.29%	58.14%
	When using new technology, I often doubt my ability to complete the task without help.	26.67%	14.29%	18.60%
Risk Aversion	I am cautious about clicking buttons if I am not sure of their outcome.	66.67%	39.29%	48.84%
	I prefer to avoid experimenting with features until I am sure they are safe to use.	6.67%	28.57%	20.93%
	I don't mind trying things out, even if I am unsure of what will happen.	26.67%	32.14%	30.23%

Facet	Option	Females	Males	Total
Tinkering	When I encounter a new feature, I like to play around with it to see what it does.	40.00%	53.57%	48.84%
	I prefer to learn step by step rather than experimenting without guidance.	40.00%	35.71%	37.21%
	When I tinker with a new feature, I often take time to reflect on what I discovered.	20.00%	10.71%	13.95%

5.4.1.3 Evaluation of the Original Version

Participants were asked to evaluate the original version of the application on a 5-point Likert scale, where one indicates that the participant strongly disagrees with the statement and five indicates that the participant strongly agrees, regarding intuitiveness, task efficiency, error proneness, and overall satisfaction.

The results were analyzed by gender, and overall averages were calculated to identify general trends (Table 5.4.1.3).

Table 5.2: Evaluation of the Original Version by Gender and Total Average

Aspect	Females	Males	Total Average
Intuitiveness	3.6	3.5	3.55
Task Efficiency	3.2	3.1	3.15
Error Proneness	3.5	3.4	3.45
Overall Satisfaction	3.3	3.2	3.25

Overall, the evaluation of the original version indicates moderate agreement across all aspects. Female and male participants showed very similar ratings, suggesting no substantial gender differences in perceptions of the original interface.

5.4.1.4 Comparison between the two versions

After reviewing the workflow for rescheduling a hearing in both the original version and the updated version through table actions, participants were asked to choose which version they perceived as better in terms of intuitiveness, task efficiency, error proneness, and overall satisfaction.

Table 5.3: Percentage of Participants Agreeing that the New Version is Better by Gender and Total

Aspect	Females	Males	Total
Intuitiveness	80.00%	96.43%	90.70%
Task Efficiency	80.00%	96.43%	90.70%
Error Proneness	60.00%	82.14%	74.42%
Overall Satisfaction	73.33%	96.43%	88.37%

The results indicate that a majority of participants found the new version with table actions to be superior across all aspects. Males consistently rated the new version slightly higher than females, particularly in terms of intuitiveness, task efficiency, and overall satisfaction. Nevertheless, both genders clearly preferred the table actions interface, highlighting its potential to improve task execution, user experience, and gender inclusivity.

5.5 Conclusion

The comparison between the two workflows highlights the impact of the new feature. In the original version, users had to navigate through multiple intermediate steps before reaching the point where they could enter new rescheduling details, which made the process longer. In contrast, the updated version with table actions embedded in each row allowed users to initiate the operation directly from the scheduled hearings table.

Built-in filtering and ordering options further supported this workflow by making it easier to locate the correct record before executing an action.

All of the requirements defined for the feature were successfully achieved. Actions were correctly embedded within table rows, accompanied by tooltips for clarity. Data was passed to the forms, and compatibility between entity types and action rules was ensured. The ability to reuse sets of table actions across multiple queries also ensured flexibility and consistency in different contexts.

The evaluation included a questionnaire targeting perceived usability and cognitive facets, as well as a direct comparison between the original interface and the new version with table actions.

The results indicate that the original version received moderate ratings across intuitiveness, task efficiency, error proneness, and overall satisfaction, with no substantial gender differences observed. In contrast, the new version with table actions was consistently preferred by participants, with over 74% agreement across all measured aspects and particularly high approval among male participants.

Analysis of the GenderMag cognitive facets revealed patterns largely consistent with existing research: women were generally more task-oriented, comprehensive in information processing, and cautious with unfamiliar options, while men were more exploratory

and confident in using new software. This information reinforces the importance of considering cognitive diversity when designing user interfaces.

Overall, the successful implementation of table actions met all functional requirements. Although the questionnaire involved a relatively small sample of participants, the results provide strong indications that table actions are a valuable addition to DISME. Participants agreed that this feature could improve the user experience in task execution, confirming that table actions can indeed enhance the usability of applications generated by DISME.

6

Conclusion

6.1 Results

This research investigated the usability and inclusiveness of applications generated by LCPs, with a focus on DISME. The study combined a literature review, a comparative analysis of popular platforms (OutSystems, PowerApps, and Mendix) with DISME, and a practical implementation within DISME to explore how LCPs support user-centered design. The primary goal was to enhance DISME's generated applications, with the addition of table actions.

The main technical contribution of this work is the design and implementation of the *Action Items Management* functionality, enabling table actions in generated applications. This feature allows developers to embed actions, such as scheduling or rescheduling hearings, directly within table rows. By reducing the number of steps required to perform these tasks, this functionality improves efficiency, reduces interaction complexity, and aligns with usability best practices highlighted by the GenderMag framework.

Complementing the core implementation, tooltips were added to table action buttons to provide contextual guidance, helping users understand the purpose of each action. This addresses the GenderMag facet of risk aversion, as users are more likely to interact with unfamiliar features when outcomes are clearly communicated.

Beyond the technical implementation, the research also identified specific opportunities to enhance the MHP implementation. For example, the current time slot selection mechanism, which combines day and time in a single input, can be unintuitive and inefficient. A two-step approach—selecting a day via a calendar component followed by a filtered time selection would simplify the workflow and reduce the likelihood of user errors. Similarly, removing unnecessary input fields, such as the hearing state when it can be inferred from the assigned officer, would prevent inconsistent entries and further improve the UX.

Other usability considerations were identified for multi-step workflows. Supporting navigation between steps without losing progress, possibly via a Wizard component, could enhance user control and reduce cognitive load. Additionally, the integration of

GenderMag-informed guidance in new interface components, such as table actions, could extend inclusiveness by adapting interactions to diverse cognitive styles.

The evaluation relied on a questionnaire targeting perceived usability, GenderMag cognitive facets, and a direct comparison between the original interface and the new version with table actions. While the sample was relatively small, the results provide clear indications of the impact of table actions. The original version received moderate ratings across intuitiveness, task efficiency, error proneness, and overall satisfaction, with no substantial gender differences observed. In contrast, the new version with table actions was consistently preferred across all measured aspects, particularly among male participants. Analysis of the GenderMag cognitive facets essentially confirmed existing research patterns: women were generally more task-oriented, comprehensive in information processing, and cautious with unfamiliar options, while men were more exploratory and confident using new software. These findings underscore the importance of considering cognitive diversity when designing user interfaces.

While this work demonstrates meaningful improvements, it has limitations. The evaluation relied on a pre-seeded test database and an online questionnaire rather than real users interacting with the system, so usability gains are inferred rather than empirically measured. Moreover, the focus was limited to a subset of tasks in the MHP, leaving open questions about performance and efficiency when scaling to larger datasets or more complex tables.

Overall, the implementation of table actions successfully met all functional requirements, and the results of the questionnaire demonstrated a clear improvement in usability. Although the questionnaire sample was limited, the results indicate that table actions can enhance user experience by simplifying task execution, reducing interaction complexity, and improving overall satisfaction. This confirms that table actions are a valuable addition to DISME-generated applications.

6.2 Future work

There are multiple opportunities to extend and build upon this research. A natural next step would be to conduct empirical usability studies with real users of DISME generated applications. Such studies could measure the actual impact of table actions on usability, efficiency, and user satisfaction, providing evidence to confirm or refine the findings inferred in this work.

Incorporating more advanced GenderMag-informed design principles represents another promising direction. While this work introduced static support through tooltips and embedded actions, future iterations could explore dynamic adaptation. For instance, the system could adjust interface elements, such as button styles, information presented, colors, or interaction format, based on the user's GenderMag profile, further promoting inclusivity in the generated applications.

Technical extensions also deserve attention. Evaluating the scalability and performance of table actions when applied to larger datasets is crucial, as enterprise applications often operate on high volumes of data. Integration with a more customizable button and table components, extended icon libraries, or custom icon upload could further improve the flexibility of the applications.

For the MHP, specific improvements could include the two-step calendar-based time selection and removal of redundant inputs, such as the state input field, in certain operations.

Future studies could compare single-step and multi-step approaches, assessing how they interact with cognitive facets of GenderMag, such as Information Processing Styles, Risk Aversion, and Tinkering. In cases where multi-step forms are preferred, the inclusion of a Wizard component could be evaluated for its ability to guide users through complex processes without increasing cognitive load, as well as adding the possibility of users returning to previous steps without losing already filled information. Currently, if a user needs to change a field in the last step, they must restart the entire process and start from the first step again.

By addressing these research opportunities, the DISME platform can continue to evolve into a more inclusive, usable, and technically robust LCP, further closing the gap between rapid application development and user-centered design.

Bibliography

[1] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. URL: https://github.com/joaomlourenco/novathesis/raw/main/template.pdf (cit. on p. i).

[2] A. Kalbande. *What are the pros and cons of low-code development platforms?* URL: https://www.linkedin.com/pulse/what-pros-cons-low-code-development-platforms-ashutosh-kalbande/ (visited on 2025-02-10) (cit. on pp. 1, 6).

[3] J. Dias, D. Aveiro, and M. Goulão. "Development of a Uniform, Dynamic, and Inclusive GUI for a Low-Code Platform". Dissertation Plan for a master in computer science and engineering, at NOVA University of Lisbon, not yet published (cit. on pp. 1, 5, 6, 9, 10).

[4] D. Pinho, A. Aguiar, and V. Amaral. "What about the usability in low-code platforms? A systematic literature review". In: *Journal of Computer Languages* 74 (2023), p. 101185 (cit. on pp. 1, 5, 6).

[5] D. Aveiro et al. "Traditional vs. low-code development: comparing needed effort and system complexity in the NexusBRaNT experiment". In: *2023 IEEE 25th Conference on Business Informatics (CBI)*. 2023. DOI: 10.1109/CBI58679.2023.10187470 (cit. on pp. 1, 6, 9, 10).

[6] A. Alamin et al. "An Empirical Study of Developer Discussions on Low-Code Software Development Challenges". In: *IEEE Working Conference on Mining Software Repositories* (2021). DOI: 10.1109/msr52588.2021.00018 (cit. on pp. 1, 6, 12, 13).

[7] D. Liu et al. "What's Wrong With Low-Code Development Platforms? An Empirical Study of Low-Code Development Platform Bugs". In: *IEEE Transactions on Reliability* 73.1 (2024), pp. 695–709. DOI: 10.1109/TR.2023.3295009 (cit. on pp. 1, 6, 12, 13).

[8] P. Gomes and M. Brito. "Low-Code Development Platforms: A Descriptive Study". In: *Iberian Conference on Information Systems and Technologies* (2022). DOI: 10.23919/cisti54924.2022.9820354 (cit. on pp. 1, 6, 13).

[9] V. Freitas et al. "The DISME low-code platform - from simple diagram creation to system execution". In: *Forum/CIAO! DC@ EEWC*. 2022 (cit. on pp. 1, 8–11).

[10] V. H. S. Freitas. "Extension of action rule grammar and implementation of processing engine of a DEMO based low-code platform". MA thesis. University of Madeira, 2023 (cit. on pp. 2, 5, 6, 8, 9).

[11] M. Burnett et al. "Finding Gender-Inclusiveness Software Issues with GenderMag: A Field Investigation". In: *International Conference on Human Factors in Computing Systems* (2016). DOI: 10.1145/2858036.2858274 (cit. on pp. 2, 16–18).

[12] J. Silva et al. "Low-code and No-code Technologies Adoption: A Gray Literature Review". In: *Proceedings of the XIX Brazilian Symposium on Information Systems*. 2023, pp. 388–395. DOI: 10.1145/3592813.3592929. URL: https://doi.org/10.1145/3592813.3592929 (cit. on p. 2).

[13] D. Nagy and B. Kővári. "Improving Designer-Developer Workflow for Better User Experience". In: *10th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics (CINTI)*. 2009 (cit. on p. 2).

[14] N. Forsgren. *Quantifying the impact of developer experience*. URL: https://azure.microsoft.com/en-us/blog/quantifying-the-impact-of-developer-experience/ (visited on 2025-02-10) (cit. on p. 2).

[15] F. Fagerholm and J. Münch. "Developer experience: Concept and definition". In: *2012 International Conference on Software and System Process (ICSSP)*. 2012, pp. 73–77. DOI: 10.1109/ICSSP.2012.6225984 (cit. on p. 2).

[16] A. Razzaq et al. "A Systematic Literature Review on the Influence of Enhanced Developer Experience on Developers' Productivity: Factors, Practices, and Recommendations". In: *ACM Comput. Surv.* 57.1 (2024). DOI: 10.1145/3687299. URL: https://doi.org/10.1145/3687299 (cit. on p. 2).

[17] P. Nimje. "The Rise of Low-Code/No-Code Development Platforms". In: *International Journal of Advanced Research in Science, Communication and Technology* (2024). DOI: 10.48175/ijarsct-18974 (cit. on p. 5).

[18] T. Lethbridge. "Low-code is often high-code, so we must design low-code platforms to enable proper software engineering". In: *Leveraging Applications of Formal Methods, Verification and Validation: 10th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2021, Rhodes, Greece, October 17–29, 2021, Proceedings 10*. Springer. 2021, pp. 202–212 (cit. on pp. 6, 12).

[19] S. Shridhar and S. Bose. "Analysis of Low Code-No Code Development Platforms in comparison with Traditional Development Methodologies". In: *International Journal for Research in Applied Science and Engineering Technology* 9.12 (2021), pp. 508–513 (cit. on pp. 6, 13).

[20] Mendix. *Extending Your Application with Custom Java*. URL: https://docs.mendix.com/refguide/extending-your-application-with-custom-java/ (visited on 2025-02-10) (cit. on pp. 6, 7, 13).

[21] Budibase. *Custom component*. URL: https://docs.budibase.com/docs/custom-component (visited on 2025-02-10) (cit. on pp. 6, 13).

[22] OutSystems. *Extending with Custom Code*. URL: https://www.outsystems.com/evaluation-guide/extending-with-custom-code/ (visited on 2025-02-10) (cit. on pp. 6, 13).

[23] OutSystems. *OutSystems Official Website*. URL: https://www.outsystems.com (visited on 2025-01-22) (cit. on p. 6).

[24] OutSystems. *Artificial intelligence overview*. URL: https://www.outsystems.com/evaluation-guide/ai/ (visited on 2025-01-22) (cit. on p. 7).

[25] Microsoft. *PowerApps Official Website*. URL: https://powerapps.microsoft.com/ (visited on 2025-01-22) (cit. on p. 7).

[26] Microsoft. *Overview of AI Builder*. URL: https://learn.microsoft.com/ai-builder/overview (visited on 2025-01-22) (cit. on p. 7).

[27] Microsoft. *Microsoft Power Fx overview*. URL: https://learn.microsoft.com/power-platform/power-fx/overview (visited on 2025-01-22) (cit. on p. 7).

[28] Mendix. *Mendix Official Website*. URL: https://www.mendix.com/ (visited on 2025-01-22) (cit. on p. 7).

[29] Mendix. *Mendix Platform*. URL: https://www.mendix.com/platform (visited on 2025-02-03) (cit. on p. 7).

[30] J. Yoder and R. Johnson. "The adaptive object-model architectural style". In: *Software Architecture: System Design, Development and Maintenance* (2002), pp. 3–27 (cit. on p. 8).

[31] *ISO 9241-11:2018 Ergonomics of human-system interaction – Part 11: Usability: Definitions and concepts*. International Organization for Standardization. 2018 (cit. on pp. 12, 14).

[32] I. D. Foundation. *Usability*. 2016-06. URL: https://www.interaction-design.org/literature/topics/usability (visited on 2024-09-29) (cit. on p. 12).

[33] A. Abran et al. "Usability Meanings and Interpretations in ISO Standards". In: *Software Quality Journal* 11 (2003-11), pp. 325–338. DOI: 10.1023/A:1025869312943 (cit. on p. 12).

[34] H. Henriques et al. "Improving the Developer Experience with a Low-Code Process Modelling Language". In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 2018, pp. 200–210. DOI: 10.1145/3239372.3239387. URL: https://doi.org/10.1145/3239372.3239387 (cit. on p. 12).

[35] R. Sanchis et al. "Low-code as enabler of digital transformation in manufacturing industry". In: *Applied Sciences* 10.1 (2019), p. 12 (cit. on p. 12).

[36] D. Dahlberg. *Developer experience of a low-code platform: An exploratory study*. 2020 (cit. on p. 12).

[37] A. Alamin et al. "Developer discussion topics on the adoption and barriers of low code software development platforms". In: *Empirical Software Engineering* 28 (2022). DOI: 10.1007/s10664-022-10244-0 (cit. on p. 12).

[38] C. Silva et al. "Development and validation of a descriptive cognitive model for predicting usability issues in a low-code development platform". In: *Human factors* 63.6 (2021), pp. 1012–1032 (cit. on p. 12).

[39] F. Fagerholm and J. Münch. "Developer experience: Concept and definition". In: *2012 International Conference on Software and System Process (ICSSP)*. 2012, pp. 73–77. DOI: 10.1109/ICSSP.2012.6225984 (cit. on p. 12).

[40] K. Rokis and M. Kirikova. "Exploring Low-Code Development: A Comprehensive Literature Review". In: *Complex Systems Informatics and Modeling Quarterly* (2023). DOI: 10.7250/csimq.2023-36.04 (cit. on p. 13).

[41] J. Pacheco, S. Garbatov, and M. Goulão. "Improving collaboration efficiency between ux/ui designers and developers in a low-code platform". In: *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. 2021, pp. 138–147 (cit. on p. 13).

[42] P. Agarwal et al. "Designing for inclusive national digital ID platform: A MOSIP case study". In: *Proceedings of the International Conference on Trustworthy Digital ID*. 2023, pp. 1–9 (cit. on p. 13).

[43] A. Fallatah. "Inclusive Software Design and Its Methods for Beyond-WIMP User Interfaces". MA thesis. Oregon State University, 2023 (cit. on p. 13).

[44] C. Mendez et al. "From GenderMag to InclusiveMag: An inclusive design meta-method". In: *2019 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE. 2019, pp. 97–106 (cit. on pp. 13, 18).

[45] F. Momotaz et al. "Understanding the Usages, Lifecycle, and Opportunities of Screen Readers' Plugins". In: *ACM Transactions on Accessible Computing* 16 (2023), pp. 1–35. DOI: 10.1145/3582697 (cit. on p. 13).

[46] M. Schrepp. "On the efficiency of keyboard navigation in Web sites". In: *Universal Access in the Information Society* 5 (2006), pp. 180–188. DOI: 10.1007/s10209-006-0036-x (cit. on p. 13).

[47] D. Weld and K. Gajos. "SUPPLE: automatically generating user interfaces". In: *Proceedings of the 9th International Conference on Intelligent User Interfaces*. Association for Computing Machinery, 2004, pp. 93–100. ISBN: 1581138156. DOI: 10.1145/964442.964461 (cit. on pp. 13, 14).

[48] Z. Huang and M. Benyoucef. "A systematic literature review of mobile application usability: addressing the design perspective". In: *Universal Access in the Information Society* 22 (2022-08), pp. 1–21. DOI: 10.1007/s10209-022-00903-w (cit. on p. 14).

[49] M. Andersson and O. Lang. *Users perceptions about the usability of a LCDP mobile application*. 2021 (cit. on p. 14).

[50] M. Burnett et al. "GenderMag: A method for evaluating software's gender inclusiveness". In: *Interacting with computers* 28.6 (2016), pp. 760–787 (cit. on pp. 15, 16, 18).

[51] E. Murphy-Hill et al. "GenderMag Improves Discoverability in the Field, Especially for Women: An Multi-Year Case Study of Suggest Edit, a Code Review Feature". In: *International Conference on Software Engineering* (2024). DOI: 10.1145/3597503.3639097 (cit. on p. 15).

[52] S. Stumpf et al. "Gender-inclusive HCI research and design: A conceptual review". In: *Foundations and Trends® in Human–Computer Interaction* 13.1 (2020), pp. 1–69 (cit. on p. 15).

[53] M. Hamid et al. "How to Measure Diversity Actionably in Technology". In: *Equity, Diversity, and Inclusion in Software Engineering: Best Practices and Insights*. Berkeley, CA: Apress, 2024, pp. 469–485. ISBN: 978-1-4842-9651-6. DOI: 10.1007/978-1-4842-9651-6_27. URL: https://doi.org/10.1007/978-1-4842-9651-6_27 (cit. on p. 16).

[54] M. Vorvoreanu et al. "From Gender Biases to Gender-Inclusive Design: An Empirical Investigation". In: *International Conference on Human Factors in Computing Systems* (2019). DOI: 10.1145/3290605.3300283 (cit. on p. 16).

[55] M. Burnett et al. *The GenderMag Kit: Gender-Inclusiveness Methods for Software and Systems*. 2020. URL: https://gendermag.org/Docs/GenderMagHandout-2020-0106-1649.pdf (visited on 2024-10-17) (cit. on p. 16).

[56] C. Hill et al. "Gender-inclusiveness personas vs. stereotyping: Can we have it both ways?" In: *Proceedings of the 2017 chi conference on human factors in computing systems*. 2017, pp. 6658–6671 (cit. on p. 17).

[57] C. Gralha, M. Goulao, and J. Araujo. "Are there gender differences when interacting with social goal models? A quasi-experiment". In: *Empirical Software Engineering* 25.6 (2020), pp. 5416–5453 (cit. on p. 19).

[58] D. Aveiro et al. "Extending DEMO Action Rule Specifications' Syntax in a Low Code Platform Based Municipality Hearing System Implementation". In: *16th International Conference on Knowledge Engineering and Ontology Development*. 2024-01, pp. 243–251. DOI: 10.5220/0013068800003838 (cit. on p. 19).

[59] R. Ramdoyal, A. Cleve, and J.-L. Hainaut. "Reverse Engineering User Interfaces for Interactive Database Conceptual Analysis". In: *Advanced Information Systems Engineering*. Ed. by B. Pernici. Springer. 2010, pp. 332–347. ISBN: 978-3-642-13094-6 (cit. on p. 22).

[60] OutSystems. *Entities*. URL: https://success.outsystems.com/documentation/11/building_apps/data_management/data_modeling/entities/ (visited on 2025-01-23) (cit. on p. 24).

[61] OutSystems. *Aggregate*. URL: https://success.outsystems.com/documentation/11/reference/outsystems_language/data/handling_data/queries/aggregate/ (visited on 2025-01-23) (cit. on p. 26).

[62] Mendix. *The Mendix Atlas UI Framework*. URL: https://www.mendix.com/atlas/ (visited on 2025-04-07) (cit. on p. 27).

[63] OutSystems. *Implementing Logic*. URL: https://success.outsystems.com/documentation/11/reference/outsystems_language/logic/implementing_logic/ (visited on 2025-01-25) (cit. on p. 30).

[64] U. Patterns. *Wizard Design Pattern*. URL: https://ui-patterns.com/patterns/Wizard (visited on 2025-02-11) (cit. on p. 39).

[65] C. Navy. *The Ultimate Guide to Web Wizard Design*. URL: https://lab.interface-design.co.uk/the-ultimate-guide-to-web-wizard-design-5b6fb4201f94 (visited on 2025-02-11) (cit. on p. 39).

[66] O. Chinonyerem. "Design Implementation of a Web Application for Microfinance Institution Financial Ratio Reporting". In: 2020. URL: https://api.semanticscholar.org/CorpusID:237158340 (cit. on p. 48).

[67] L. Zhang. "Research and Application of Icon Fonts in Web Front-end Technology". In: *Proceedings of the 3rd International Conference on Computer Engineering, Information Science Application Technology (ICCIA 2019)* (2019). DOI: 10.2991/ICCIA-19.2019.41 (cit. on p. 48).

[68] Laravel. *Database: Migrations*. URL: https://laravel.com/docs/12.x/migrations (visited on 2025-09-17) (cit. on p. 51).