X-arq *Webification*

Isaac Macedo Vargas

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors: Prof. José Luís Brinquete Borbinha
Eng. Bruno Filipe Conceição Ferreira

Examination Committee

Chairperson: Prof. Paolo Romano
Supervisor: Prof. José Luís Brinquete Borbinha
Member of the Committee: Prof. João Fernando Peixoto Ferreira

October 2020

Acknowledgments

I would like to thank my parents and brothers for their support and help during all these years and for having kept pushing me forward.

To everyone at Mind for receiving me in their midst as one of them, a special thanks to Rui Casteleiro for proposing the thesis, and to the GI department for being my colleagues during this time.

I would also like to acknowledge my thesis supervisors Prof. José Borbinha, and Bruno Ferreira for allowing me to work at my own rhythm and as well as my colleague Miguel for their insight, support, and sharing of knowledge that has made this thesis possible.

Last but not least to my Instituto Superior Técnico (IST) colleagues and all my friends, specially Rodolfo, Gustavo, Karina, Diogo, and Luís that helped me grow as a person and were always there for me.

To every one of you – Thank you.

Abstract

In today's world, the Internet and the World Wide Web (WWW) is pervasive in society, an example of this is the Internet of Things (IoT), but many applications still don't take advantage of the prevalence of them.

A web application was developed to make use of this new reality, this application is X-arq Description, a module of X-arq. X-arq is a standardised archive solution created by Mind, that is used by several archives in Portugal to manage their archival collections.

The development of X-arq Description was the main goal of this project. It's a web application that was built using Vue.js, a JavaScript (JS) framework, for the frontend, and ASP.NET Core, a C# framework, for the backend.

Keywords

X-arq *Webification*; Archival Management Software; Records System.

Resumo

No mundo de hoje, a Internet e a WWW estão difundidas pela socidade, um exemplo é a IoT, mas muitas aplicações ainda não tiram vantagem da prevalência delas.

Uma aplicação web foi desenvolvida para fazer uso desta nova realidade, esta aplicação é X-arq Description, um módulo do X-arq. X-arq é uma solução normalizada de arquivo criada pela Mind, usada por vários arquivos em Portugal para gerir as suas coleções de arquivos.

O desenvolvimento de X-arq Description foi o objetivo principal desta tese. É um aplicação web que foi construída usando Vue.js, uma framework JS, e ASP.NET Core, uma framework C#.

Palavras Chave

X-arq *Webificação*; *Software* de gestão de arquivos; Sistema de registos.

Contents

List of Figures

List of Tables

Acronyms

API Application Program Interface

AtoM Access to Memory

CGD Caixa Geral de Depósitos

CSS Cascading Style Sheets

DGALB Direção-Geral do Livro, dos Arquivos e das Bibliotecas

DOM Document Object Model

DRE Diário da República Eletrónico

E2E End-to-End

EDM Entity Data Model

EF Entity Framework

EPAL Empresa Portuguesa de Águas Livres

ES ECMAScript

EU European Union

FR Functional Requirement

FRD Folha de Recolha de Dados

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

ICA International Council on Archives

IE Internet Explorer

IoT Internet of Things

ISO International Organization for Standardization

IIS Internet Information Services

IST Instituto Superior Técnico

JS JavaScript

JSON JavaScript Object Notation

MFC Microsoft Foundation Class Library

NFR Nonfunctional Requirement

ORM Object-Relational Mapping

OS Operating System

PBI Product Backlog Item

PHP PHP: Hypertext Preprocessor

PWA Progressive Web Application

REST Representational State Transfer

RDBMS Relational Database Management System

SEO Search Engine Optimization

SFC Single-File Component

SPA Single-Page Application

SQL Structured Query Language

TFS Team Foundation Server

TS TypeScript

UI User Interface

UX User Experience

W3C World Wide Web Consortium

WAI Web Accessibility Initiative

WCAG Web Content Accessibility Guidelines

WWW World Wide Web

1

Introduction

Contents

X-arq[1], the name being derived from ”extended archive”, is an archival management software application by Mind[2], a Portuguese company founded in 1997. It was created in 2001 and runs on the .NET Framework environment. It is a standardised archive solution that is used by several archives in Portugal to manage their archival collections.

It follows standards of the International Council on Archives (ICA)[3] among others.

Currently, X-arq is used by over 50 institutions, some of them are the Arquivo Municipal de Lisboa[4], Arquivo Municipal de Oeiras, Arquivo Municipal de Pombal, Arquivo Municipal de Cascais, Arquivo Municipal de Albufeira, Arquivo Municipal de Montemor-o-Novo, and Empresa Portuguesa de Águas Livres (EPAL).

1.1 Motivation and Objectives

X-arq is an older desktop application, started in 2001. Mind wanted to update it and so it was decided to reimplement X-arq using newer technology, to move away from Microsoft Visual C++ into C# and into web technology.

X-arq Description is to have the same functionalities as the old X-arq description module but with a revamped User Interface (UI) and User Experience (UX) as well as allowing for the system to be accessed remotely and a modern backend using C#.

1.2 Contributions

The current solution is at the stage of being a prototype with the core functionalities ready to be exhibited to customers, but it's not ready to be used in production and to replace the old X-arq module. A new feature that was introduced during the development was making it a Progressive Web Application (PWA) and not simply a web application.

X-arq Description being a PWA brings additional properties that are useful for X-arq Description, mainly the caching of the static parts of the UI which allows for a faster and responsive UX for successive uses of the application.

X-arq Description has a three-tier architecture.

The frontend is currently in a good shape, and it has been heavily tested, allowing for it to be easily modified and introduce new features if you already know the structure of Vue projects. It was all done from scratch, being a brand new UI.

[1] https://x-arq.mind.pt/en/
[2] https://mind.pt/mind/en/
[3] https://ica.org/en/public-resources/standards
[4] http://arquivomunicipal2.cm-lisboa.pt/sala/online/ui/searchbasic.aspx

The backend has a simple and linear structure with the responsibilities of each component clearly defined that should make it easy to understand and added upon.

The database queries used have to be refined, as currently they don't have the necessary performance.

1.3 Document Structure

This document is structured in 6 different chapters as follows:

1. **Introduction:** The current chapter, where a contextualisation of the project is exposed.
2. **Related Work:** An overview of what is archival management and the current state of X-arq.
3. **Analysis and Architecture:** The project objective and proposal are examined.
4. **Solution:** A detailed report of the work done during the project.
5. **Evaluation:** The technical tests and solution compliance with the requirements.
6. **Conclusions and Future Work:** The knowledge obtained during the project and future work.

2

Related Work

Contents

In this chapter, the context of this project is defined by having an overview of archival management software. Followed by examples of other solutions currently in the Portuguese market.

2.1 Archival Management Software

Archival management software is software that was created to manage archival collections. A management system for records is defined as ”management system to direct and control an organization with regard to records” [1]. A records system is defined as ”information system which captures, manages and provides access to records over time” [1, 2].

This software needs to have certain capabilities, the main three are: being able to capture [3]; do records management [1, 2]; and have access [1, 2]. It also has to ensure that each record is classified, retained, and disposed following the provided information long with other concepts [1, 2, 4].

Capture is defined as ”process of lodging a document or digital object into a records management system and assigning metadata to describe the record and place it in context”. [3]

Records management is defined as ”field of management responsible for the efficient and systematic control of the creation, receipt, maintenance, use and disposition of records, including processes for capturing and maintaining evidence of and information about business activities and transactions in the form of records”. [1, 2]

Access is defined as ”right, opportunity, means of finding, using or retrieving information”. [1, 2]

These concepts are all defined and standard by different standard authorities, by the previously mention ICA[1], Direção-Geral do Livro, dos Arquivos e das Bibliotecas (DGALB)[2], Open Archives[3], and International Organization for Standardization (ISO)[4].

Standards from ICA are such as the International Standard Archival Description (ISAD(G)) [**?**] and the International Standard Archival Authority Record (ISSAR(CPF)) [**?**].

2.2 Market Solutions

There are main products in the market but in the Portuguese market there are only two other main solutions, namely archeevo[5], by KEEP SOLUTIONS, and gead[6], by Libware. There’s also an open source international solution, Access to Memory (AtoM)[7] that is currently maintained by Artefactual Systems after being originally built with the support of ICA.

[1] https://ica.org/
[2] http://dglab.gov.pt/
[3] https://openarchives.org/
[4] https://iso.org/
[5] https://keep.pt/en/produts/archeevo-archival-management-software/
[6] http://libware.pt/en-gb/products/gead.aspx
[7] https://accesstomemory.org/

In respect to the functionality of X-arq Description, AtoM is web-based while gead is a desktop application and archeevo is mixed having several of the functionalities in a web-based approach and even offering a limited version of their software for Android devices, allowing the connection of records and their physical location while navigating the locale.

Both archeevo and AtoM have a demo available. For archeevo the demo is available online on their website[8] while AtoM has it available to be downloaded to be run locally. These two demos were used to experience the products and take the screenshots that are in the figures of this section.

2.2.1 archeevo

KEEP SOLUTIONS has a whitepaper[9] describing the characteristics of archeevo.

Taking a few details from it to give an overview of the architecture, there are nine functional modules that are on top of five application modules: one module for Windows; two for use in Web; one for Android; and an Application Program Interface (API).

These connect with the archeevo business logic and core service through SOAP or a REST service which runs on top of Microsoft products such as Internet Information Services (IIS), Microsoft SQL Server, Windows Server, etc.

Following there's a screenshot of archeevo that allows us to see records in visualization mode.

Figure 2.1: Screenshot of archeevo in visualization mode.

[8] https://demo.archeevo.pt/
[9] https://www.keep.pt/wp-content/uploads/2019/12/WP181116.1-whitepaper-archeevo-5_EN.pdf

To view the records in X-arq in this form of visualization the X-arq Web module is used, which can be open directly from a record in X-arq Description.

2.2.2 AtoM

AtoM is completely web-base, it also has multilingual support, including in Portuguese, and is open source. The AtoM code is released under the GNU Affero General Public License (A-GPL 3.0). Making it a copyleft licence and therefore it's not a permissive licence. It is mainly build with PHP: Hypertext Preprocessor (PHP).

Following there's a screenshot of AtoM in edition mode, which is the mode of X-arq Description.

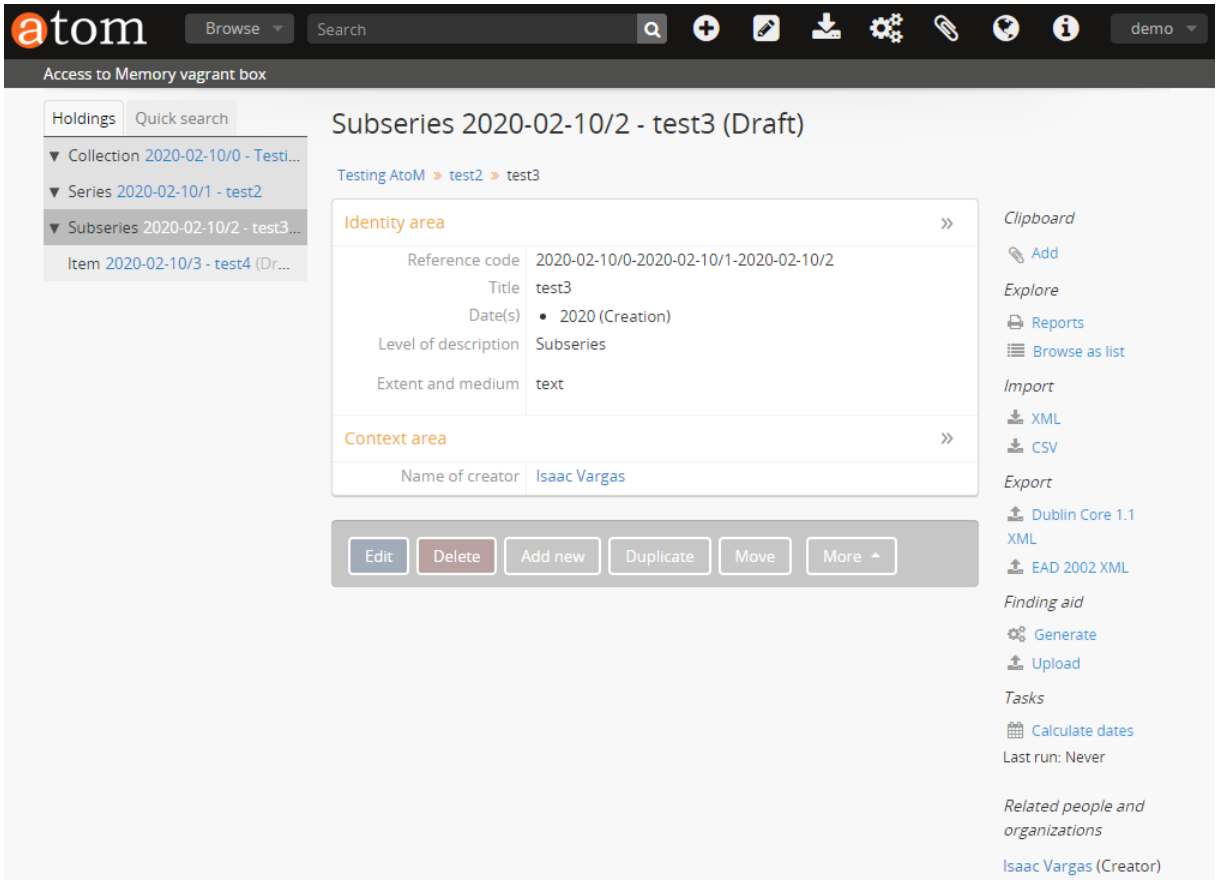


Figure 2.2: Screenshot of Access to Memory (AtoM) in edition mode.

3

Analysis and Architecture

Contents

This chapter presents an overview of the current state of the X-arq description module, analysis and requirements of the system are presented, and the architecture of the new solution is introduced.

3.1 Current Application

Most of the X-arq code is currently in Microsoft Visual C++ only the X-arq Web module code is currently on C#. X-arq has four modules: the description module, X-arq; the configuration module, X-arq Config; and the two search modules, X-arq Search and X-arq Web.

The goal of this project is to reimplement the description module. It's in this module that the records management happens. Where the users create, modify, and delete records.

The X-arq UI is in Portuguese and no other languages are provided which means that some acronyms maybe be used in their Portuguese version and that all text in screenshots of the application will be in Portuguese such as the following one.

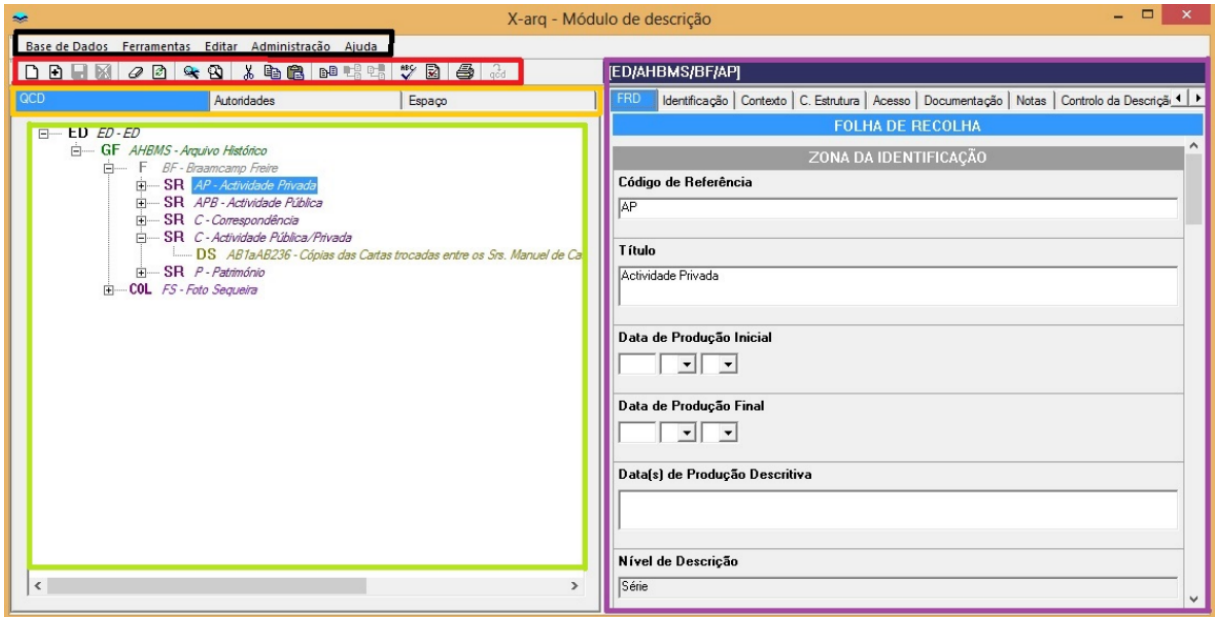


Figure 3.1: A screenshot of the description module of X-arq with the six zones delimited.

In the screenshot in fig. 3.1 the main view of X-arq can be seen and its six zones are delimited. In the black zone there are the menus. In the red zone the actions shortcuts. In the orange there are the three navigation areas that this particular user has access. In the green zone there is the hierarchy of records this user can visualize. In the violet zone the selected record's data is exhibited.

Looking at the violet zone it's noticeable four different types of behaviour from the six visible fields. Each field can have associated different rules that modify the behaviour of each field. This will be the data management section of the requirements.

3.2 Analysis

In this section the system will be analysed by using use cases, exploring the system's domain model, and documenting the software requirements.

Analysing the current X-arq module of description, the features can be grouped into three loose categories: core; complementary; and add-ons. Due to time constrains only the core functionalities are in the scope of this project and as such are here specified.

3.2.1 Use Cases

Looking at fig. 3.2 which has the core use cases of X-arq Description. There are two actors: the anonymous user which only has access to the home page and other static pages like it and one feature, authentication; and the authenticated user which can perform operations on the records according to the permissions it has.

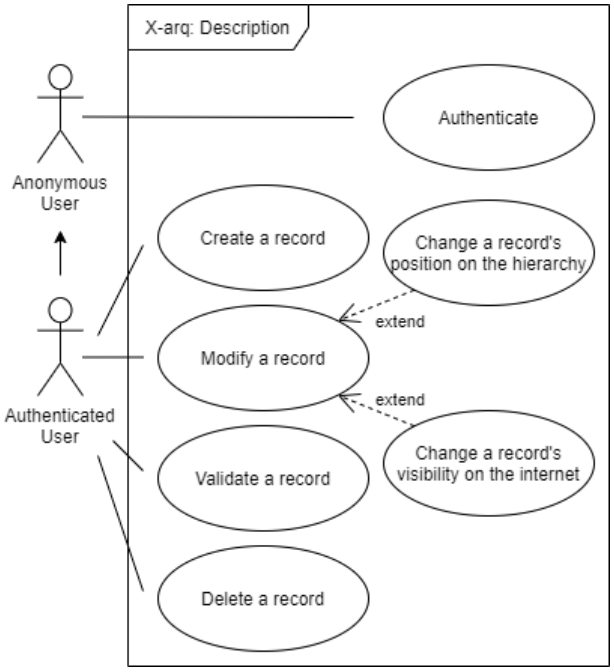


Figure 3.2: Use case diagram of X-arq Description.

In fig. 3.2 use cases relating to complementary and add-ons features weren't included, as previously stated only the core functionalities are the focus of the project. An example of a complementary use case would be a "managing users" subgroup that is a generalization of creating new users and setting user permissions. An add-on use case would be, for example, connecting X-arq Description with Kapture[1].

[1] https://kapture.mind.pt/indexNew_en.aspx

3.2.2 Requirements

A software requirement is the software capability needed by a user to solve a problem or to achieve an objective or the software capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document [5].

In this section the Functional Requirements (FRs) and Nonfunctional Requirements (NFRs) for X-arq Description are detailed.

3.2.2.A Functional

A Functional Requirement (FR) is a statement that identifies what results a product or process shall produce or a requirement that specifies a function that a system or system component shall perform [5]. A FR defines the behaviour of the system.

X-arq functionalities can be grouped into three loose categories: core; complementary; and add-ons. The core requirements can be further subdivided into three subgroups: permissions; record operations; and data management.

- **G1: Permissions**
  - **G1.1: Authentication -** users must be authenticated to access the system.
  - **G1.2: Component -** users can only access the components they have permission.
  - **G1.3: Record -** users can only access the records they have permission.
  - **G1.4: Operation -** users can only perform operations that they have permission.
  - **G1.5: Lock -** users can only modify a record if they have its lock (i.e. no one is modifying it).
- **G2: Record Operations**
  - **G2.1: Creation -** a new record can be created.
  - **G2.2: Validation -** a record can be validated.
  - **G2.3: Change Level -** a record can be moved on the hierarchy of records.
  - **G2.4: Allow Display Content on the Internet -** a record's visibility can be set.
  - **G2.5: Deletion -** a record can be set to deleted.
- **G3: Data Management**
  - **G3.1: Visualisation -** present each field and subfield correctly.
  - **G3.2: Modification -** allow each field and subfield to be modified correctly.

In general these are the three subgroups of the core totalling 12 general FRs. These were the expected requirements to be completed in the period of this project.

3.2.2.B Nonfunctional

A Nonfunctional Requirement (NFR) is a software requirement that describes not what the software will do but how the software will do it [5].

Examples of NFRs are quality attributes that end with "ility" such as availability, interoperability, etc.

Several of NFRs that are considered important for X-arq Description are the following:

- **Accessibility** is the degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use [6].
- **Performance**, in particular time behaviour, which is defined as the degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements [6].
- **Testability** is the degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met [6].
- **Usability** is the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use [6].
- **Maintainability** is the degree of effectiveness and efficiency with which a product or system can be modified by the intended maintainers [6]. Testability can be considered a subpart of this quality.
- **Security** is the degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization [6].

These six NFRs are considered important for X-arq Description each for their reason.

The main target audience for this project's software is government employees, which mean it's software that will be bought by the government and there are guidelines to follow. The European Union (EU) publishes directives[2] that each government in the union has to implement in their way.

The Directive (EU) 2016/2102 was one of such directives and it was about the accessibility of the websites and mobile applications of public sector bodies [7]. This directive was implemented by the Portuguese Government with Decreto-Lei n.º 83/2018 [8].

The Decree Law sets the accessibility guidelines to follow as the AA level of the Web Content Accessibility Guidelines (WCAG) 2.1 created by the World Wide Web Consortium (W3C) [9].

Testability and maintainability are important for the code quality of the product so while it doesn't have much direct impact on the product it has a considerable indirect impact. A system must be testable

[2] https://europa.eu/european-union/eu-law/legal-acts_en

to ensure it's correct behaviour and have high maintainability so that the developers have an easier time understanding the system and being able to modify it.

The cost is also a big factor, studies have shown that over the entire life-cycle of software maintenance can cost over 40% and that has been increasing up to 80% [10, 11].

Usability has a part of subjectivity because in part it is determined by user satisfaction, which is a major point. If a user starts getting frustrated with the software then the desire to use it and the productivity of using it decreases. X-arq Description should have slightly higher usability than the old version it's trying to replace since the UX was one of the reasons for the change.

Performance is a basic requirement in most systems. For X-arq Description the part that isn't trivial is the hierarchy of the records. To achieve good performance the most important part is to have good database indexes, good Structured Query Language (SQL) queries in the server, and a good tree component to display and control the records in their hierarchy form in the client. The component used for the tree in the client is an open-source package that was imported as a dependency. Its name is LiquorTree[3] and it was created by Kostiantyn.

X-arq Description being a web application instead of a desktop application means that there is now additional importance on the security. So there are two important sides now. The external security that only authorized agents can use the system, and the internal security that each user can only perform tasks and access the parts of the system that they have permissions to do so.

The first four NFRs, accessibility, performance, testability, and usability, will be evaluated in chapter 5. The last two NFRs, maintainability and security, won't be evaluated due to the increased complexity in evaluating them combined with the project's time constraint.

3.3 Architecture

X-arq software architecture is a client-server architecture, more specifically a three-tier architecture: presentation tier; application tier; and data tier. As seen in fig. 3.3. This is a common architecture in the web environment.

In this section the architecture deployment diagram and technology are detailed as well as a screenshot of the current X-arq Description UI is presented.

3.3.1 Diagrams

X-arq Description follows a three tier architecture and is deployed as seen in fig. 3.4.

On fig. 3.4 on the left side we have the first tier, the presentation tier that is client-side and can be accessed using a web browser.

[3] https://amsik.github.io/liquor-tree/

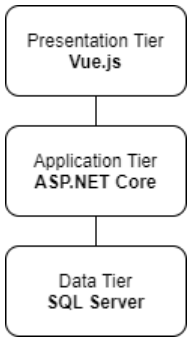


Figure 3.3: Three tier architecture diagram of X-arq Description.

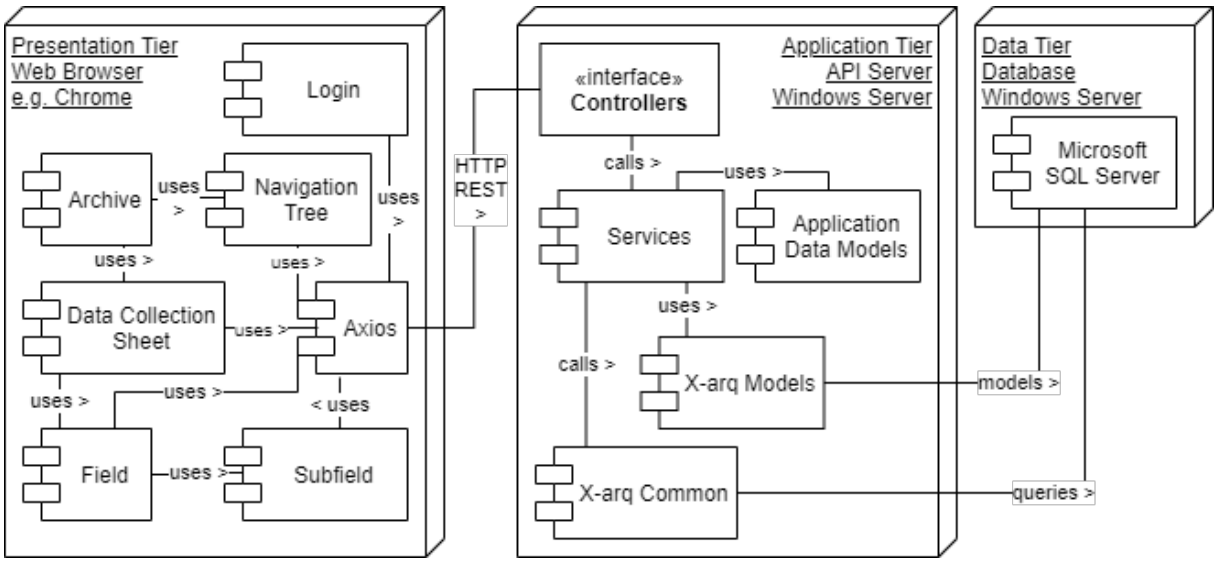


Figure 3.4: Deployment diagram of X-arq Description.

Three modules on the presentation tier to noticed are: `login`; `archive`; and `axios`. The `axios` module is a dependency, it is a Hypertext Transfer Protocol (HTTP) client, and is used to make all requests to the backend, the API. The `login` and `archive` modules are Vue views. The `login` view is used for authentication while `archive` is the main view. All other modules can be seen to make a tree with `archive` as the root. This is how Vue works, the components are hierarchical and make a tree, more details will be shown in chapter 4.

In the middle of fig. 3.3 there is the application tier, this is the API and lives in a server running Windows Server.

The interaction with the API server is done through Representational State Transfer (REST) HTTP requests using the JavaScript Object Notation (JSON) format. These requests are received by the controllers that then call the services. The services are where the business logic is implemented.

The services use classes that represent data formatted in different ways. The `X-arq models` are Entity Data Models (EDMs), the models of the database, that are used by Entity Framework (EF), an

Object-Relational Mapping (ORM), to communicate with the database. The `application data models` are other representations of data, such as the data in a tree model, or in other models that are used to send as a response to a request.

The final module in the application tier worth mentioning is `X-arq common`, this is a module that already existed, it is also used by X-arq Web. It implements common functionalities that aren't X-arq Description specific such as authentication. It also communicates with the database on its own right instead of going through EF using the `X-arq models`.

The third tier, on the right of fig. 3.3, is the data tier that consists of the database that is a Microsoft SQL Server database that runs on Windows Server.

On fig. 3.5 there is a simplified version of the domain model.

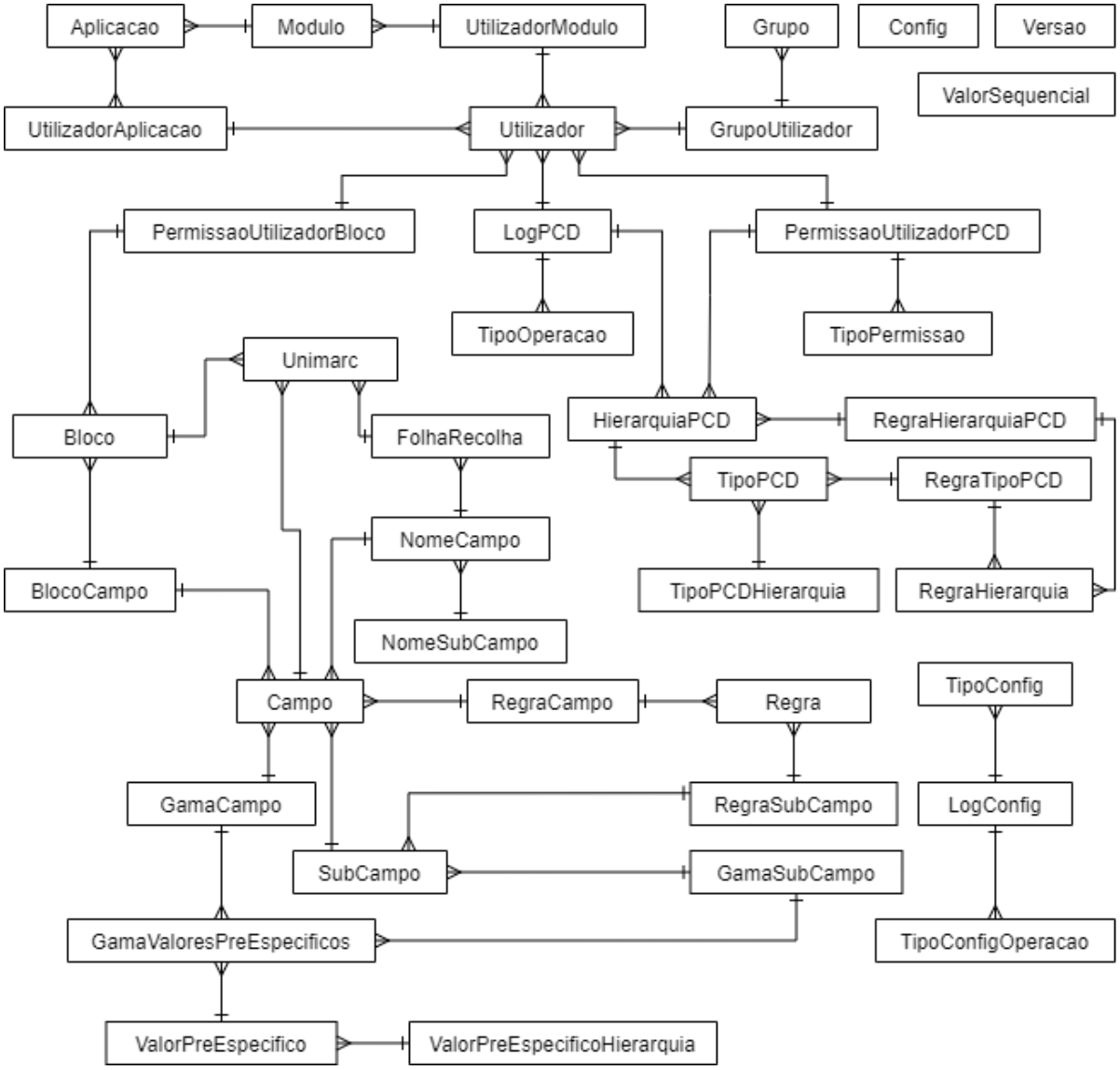


Figure 3.5: Domain model of X-arq Description.

All the classes that have `PCD` in the name also exist but with `Espaco` and `Autoridade` in the name instead of `PCD` and all the classes that have `Utilizador` in the name except for `Utilizador` and `GrupoUtilizador` also exist but with `Grupo` in the name.

3.3.2 Technology

The purpose of this project was to reimplement X-arq using newer technology.

These technologies can be grouped into four: the frontend group; the backend group; the database technology; and the server group that the backend and the database run on top of.

- **Node.js**[4] is a JavaScript (JS) runtime environment that it is used for development. An important part is npm (originally short for Node Package Manager) which allows for the use of over 1.2 million packages as dependencies in the project.
  - **Vue.js**[5] is a progressive JS framework for building UIs and Single-Page Applications (SPAs).
  - **TypeScript (TS)**[6] is a strict syntactical superset of JS which adds optional static typing to the language. It is then transcompiled into JS.
- **.NET**[7] is an open-source developer platform, created by Microsoft, made up of tools, programming languages, and libraries for building many different types of applications. Included is NuGet, the environment package manager.
  - **.NET Framework** is the original implementation of .NET.
  - **ASP.NET Core** is a open-source web framework that will mainly serve as the backend.
  - **Entity Framework Core** is an ORM to ease communication with the database and the manipulation of the data by use of the EDMs.
- **Microsoft SQL Server** is a Relational Database Management System (RDBMS) that is used to store the persistent data used by all the X-arq modules mainly X-arq Description.
- **Windows Server** is a server Operating System (OS) that serves has the basis for the web and the database server.
  - **Internet Information Services (IIS)** is a web server to run the ASP.NET Core application.

Not all of the previously listed technologies are new in the X-arq technology stack.

[4] https://nodejs.org/
[5] https://vuejs.org/
[6] https://www.typescriptlang.org/
[7] https://dotnet.microsoft.com/

The ones used in the frontend, the Node (Node.js) group are all new as well as ASP.NET Core and EF Core from the .NET group.

While the old implementation of X-arq also ran in .NET Framework and used Microsoft SQL Server and Windows Server it didn't use an ORM and it was build using Microsoft Foundation Class Library (MFC) which is now being replaced by ASP.NET Core.

3.3.3 User Interface

A mock up of how the new UI should look for X-arq Description was created by a Mind's designer. Changing the old windows application look for a modern web flat design and moving to a more vibrant color by using orange instead of blue.

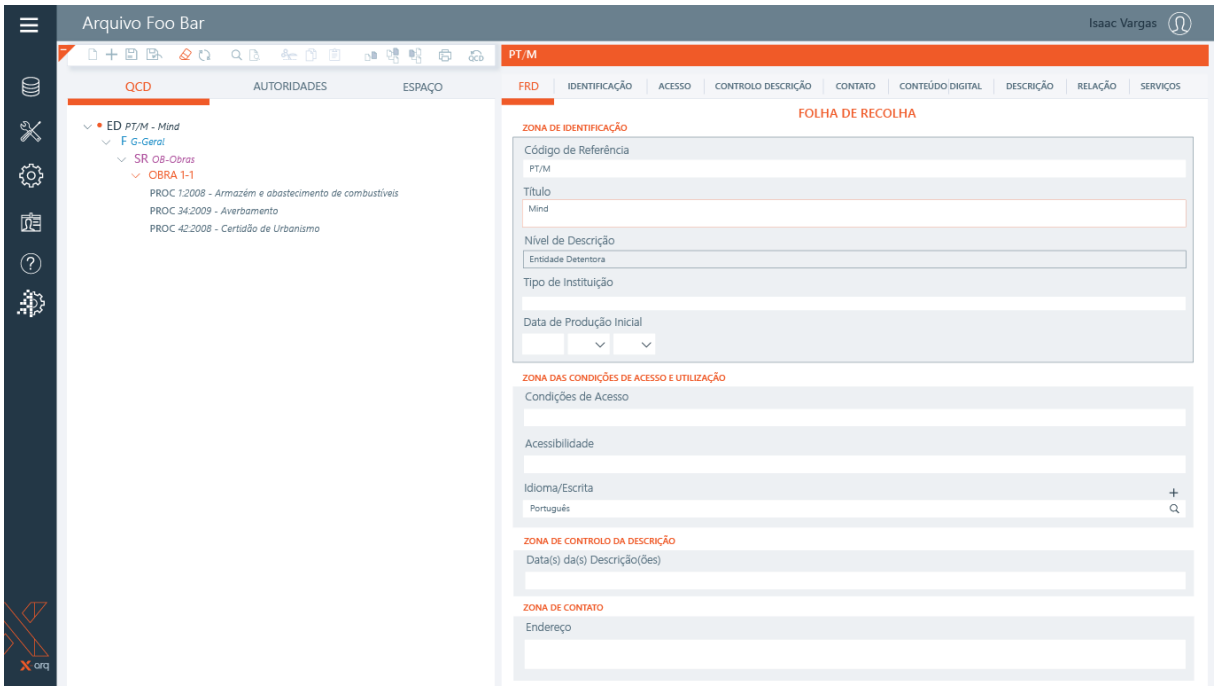


Figure 3.6: Mock up of the new UI for X-arq Description.

Development of X-arq Description was made following fig. 3.6 as the guide.

4

Solution

Contents

In this chapter, a detailed description of the new solution is presented. The difficulties encountered are discussed as well as the differences between the planning and the execution are examined.

4.1 Development Process

This project has gone through several phases:

- Technology Research and Related Works
- Requirements Gathering and UI Design
- Implementation Process and Technical Testing
- Functional Testing

This project looks at all four phases but in this chapter, the focus is on the third point.

For the third point, Implementation Process and Technical Testing, agile methodology principles were adopted to manage the development process. Mind uses Team Foundation Server (TFS) to manage their projects as well as for version control. TFS has several functionalities, it's possible to create sprints and Product Backlog Items (PBIs) and associate them with a developer, etc. In fig. 4.1 a sprint of X-arq TFS is presented.

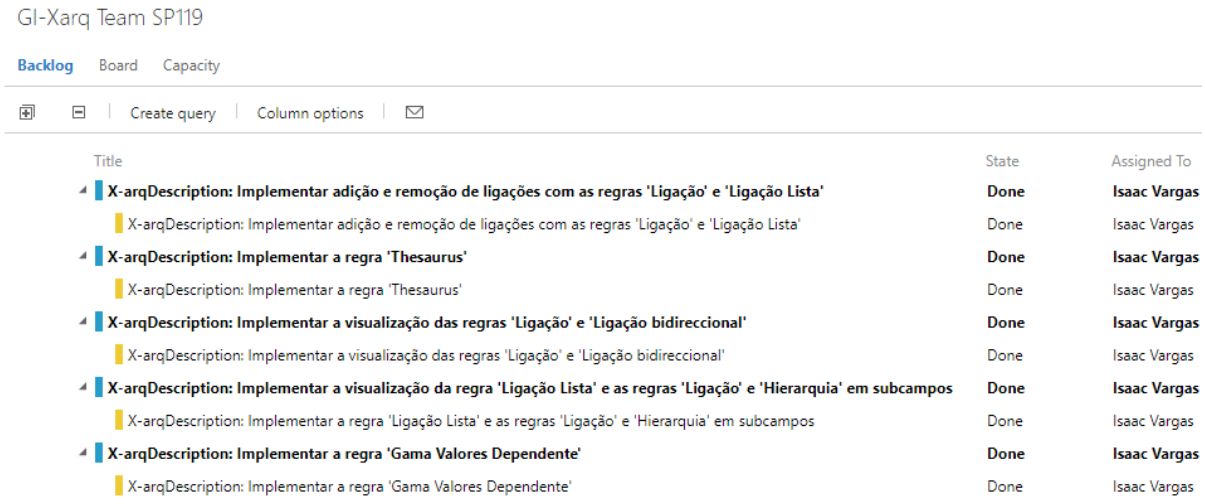


Figure 4.1: View of a X-arq sprint in TFS.

After setting up a very bare-bones project without tests, the unit and integration testing was done in concurrency with the implementation of each functionality. When the project had a few basic functionalities implemented it was then deployed onto a local server allowing the access of it through Mind's intranet to allow easy access to other stakeholders in the project.

4.2 Dependencies

Being in the .NET and Node ecosystems they possess package managers, NuGet and npm, respectively. Both have a huge registry of packages, NuGet has over 190 thousand, which is good enough for fifth on the list of package managers with the most packages in their registry, but this falls short of the number of packages accessible through npm with over 1.2 million, being the package manager with the most of packages in their registry [12].

To put in perspective the number of available packages in npm, if you sum the following five package managers after npm which are the only other package managers that have over 100 thousand packages it comes to around 1.15 million packages which is still short of npm. This lead will only continue to increase due to npm having the biggest growth rate of all package managers and also bigger than the following five package managers combined [12].

All the previously stated numbers were taken from this website[1] that tracks the number of packages in each package manager registry.

4.2.1 Frontend

In the `package.json` file of the client source code there are three different types of dependencies: dependencies; devDependencies; and optionalDependencies. The client has 13 dependencies, 22 devDependencies, and two optionalDependencies, for a total of 37 direct dependencies. But these are just the tip of the dependencies tree of the project, in total there are 1781 packages.

For comparison, a newly created Vue project with the same features selected as X-arq Description when it was scaffolded, has five dependencies, `core-js` and four relating to Vue, 18 devDependencies, and zero optionalDependencies, for 23 direct dependencies and a total of 1613 packages. This shows that the amount of packages that X-arq Description has isn't abnormal. Both it and the new project have a small difference in the number of packages between them. This is just to show that in the JS ecosystem it's not abnormal to have a big tree of dependencies.

The dependencies packages of X-arq Description are:

- **axios**[2] - a promise based HTTP client, used to make the HTTP requests to the backend.
- **bootstrap**[3] - a Cascading Style Sheets (CSS) library used to build responsive websites.
- **bootstrap-vue**[4] - a implementation of `bootstrap` in Vue that makes it even easier to use.

[1] http://modulecounts.com/
[2] https://github.com/axios/axios
[3] https://github.com/twbs/bootstrap
[4] https://github.com/bootstrap-vue/bootstrap-vue

- **core-js**[5] - a polyfill library so that it's possible to use newer versions of ECMAScript (ES), such as ES2020, that implement new features that aren't supported by the targeted browsers, namely Internet Explorer (IE) 11.
- **liquor-tree**[6] - a tree component based on Vue that implements functionalities such a drag and drop.
- **splitpanes**[7] - a Vue panes splitter/resizer.
- **vue**[8] - and six others relating to it, such as `vue-router`[9] and `vuex`[10].

Three examples of packages in the devDependencies are `@typescript-eslint/eslint-plugin`[11], `@typescript-eslint/parser`[12] and `typescript`[13] which has been previously mentioned.

4.2.1.A License

The `typescript` package has the Apache License, Version 2.0[14]. The `@typescript-eslint/eslint-plugin` and `@typescript-eslint/parser` packages have the 2-Clause BSD License[15], that is also known as the Simplified BSD License or FreeBSD License. The remaining 34 direct dependencies have the MIT License[16].

These three licenses used by the 37 direct dependencies are all permissive licenses, which unlike copyleft licenses, don't require that code licensed under them continue to be open-source and continue to be licensed under the same license. This means that code that is licensed under permissive license can be used in proprietary products, as is the case with X-arq Description.

4.2.1.B Contributions

Two important packages used in X-arq Description are `splitpanes` and `liquor-tree`. The former package allows to easily create resizable panes in a Vue project while the latter allows creating tree components in Vue. These two packages only have one developer working on them and while following these two packages some contributions were made to them: such as fixing the former not working in IE 11[17];

[5] https://github.com/zloirock/core-js
[6] https://github.com/amsik/liquor-tree
[7] https://github.com/antoniandre/splitpanes
[8] https://github.com/vuejs/vue
[9] https://github.com/vuejs/vue-router
[10] https://github.com/vuejs/vuex
[11] https://github.com/typescript-eslint/typescript-eslint
[12] https://github.com/typescript-eslint/typescript-eslint/tree/master/packages/parser
[13] https://github.com/microsoft/TypeScript
[14] https://apache.org/licenses/LICENSE-2.0
[15] https://opensource.org/licenses/BSD-2-Clause
[16] https://opensource.org/licenses/MIT
[17] https://github.com/antoniandre/splitpanes/pull/68

and when it was decided to do some small accessibility improvements to the latter an issue was created reporting that it wasn't building[18]; after that issue was dealt with the small accessibility improvements were made[19].

4.2.2 Backend

The backend naturally also has dependencies, these are installed through NuGet, the .NET environment package manager. It contains 12 packages: six are the `Microsoft.AspNetCore` package and sub packages; one is `Microsoft.EntityFrameworkCore.SqlServer` that allows the use of EF Core with SQL Server, four are `Microsoft.Extensions` sub packages, and the last is `Microsoft.TypeScript.MSBuild`.

Along with packages dependencies, X-arq Description also has project dependencies on other Mind projects, these amount to four. Three previously existing projects and one that was refactored from X-arq Description. The latter is the `X-arqDatabaseModels` project that holds the EDMs of the X-arq database.

4.3 Source Code Structure

X-arq Description is structured in a single C# project, in that project, there's a folder named `client` that contains all of the frontend source code. The directory structure of that folder can be seen in fig. 4.2(a).

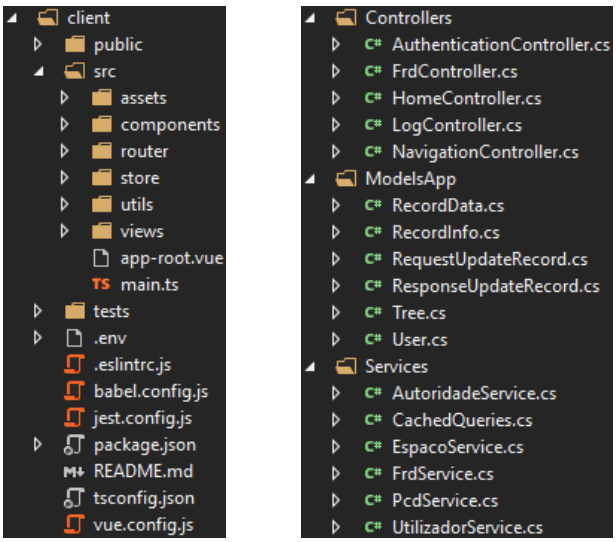
While the server-side code, the backend, is in the root of the C# project, the important source code can be seen in fig. 4.2(b), being in three folders.

4.3.1 Frontend

Starting at the bottom of fig. 4.2(a) there are several loose files, these are all configuration files for different tools. For example, the `package.json` file has one propriety, `browserslist`, that lists what browsers are being targeted, so tools that need that information will grab it from there. One final example is the `.eslintrc.js` file that is the configuration file for ESLint[20] that guarantees that all the frontend code in the project follows the same style guide.

Moving back to the top of fig. 4.2(a), looking at the `public` folder, files in this folder won't go through webpack[21], a module bundler, and will instead simply be copied. The files in it are simple things like `favicon.ico`, and `index.html` that is the base Hypertext Markup Language (HTML) file on which the web application is inserted dynamically. This file consists of a `head` tag with the necessary information defined and a `body` tag that has a `noscript` tag and a empty `div` tag with an id on which the vue web application is then connected to.

[18] https://github.com/amsik/liquor-tree/issues/181
[19] https://github.com/amsik/liquor-tree/pull/182
[20] https://eslint.org/
[21] https://webpack.js.org/



(a) Frontend **(b)** Backend

Figure 4.2: Source code directory.

Now the main folder of the frontend is `src`, it contains six folders as well as two files. Looking at the `main.ts` file that is the entry point of the application, it will create create the base Vue instance and connect it to the previously mentioned `div` in the base HTML file. The Vue object that is created is the base one that is defined in `app-root.vue`, the other file contained in `src` folder. This is a vue file, all the files ending in '`.vue`' are Single-File Components (SFCs)[22], these files have three root tags: `template`; `script`; and `style`. In the interior of each of these respectively tag there is the HTML markup, a JS dialect in this project being TS, and the CSS classes. In fig. 4.3 the code in `app-root.vue` is shown as an example.

Looking at the content of the `template` tag there is the `div` with the 'app' id property mirroring what is on the HTML file in the `public` folder. Inside that is a `router-view` tag. This is a custom Vue tag that will render the contents of the component that has the current route that the user is viewing associated in the `vue-router` configuration. For example, if a user is in the '/' route, and in the `vue-router` configuration that route is linked to the home component, then the custom `router-view` will be replaced with the content of the `template` base tag from the home component file.

In the following line the `router-view` tag has a `class` propriety defined, but notice the colon before which tells Vue to bind it dynamically instead of just analysing it statically. In it, it has a static `class`, which is `view`, and a dynamic one that depends on a boolean variable, `cursor-wait`.

On line 5 there is an event listener which has the '@' character to mark it as so. It listen to events named `viewEmitCursorWait` and when it receives one it calls the `setCursorWait` method.

[22] https://vuejs.org/v2/guide/single-file-components.html

Figure 4.3: Partial view of the contents of the `app-root.vue` file.

The rest of the HTML is defined in each view, this is the base markup that is in every view.

As previously mentioned, the markup in the `template` tag references code that is inside the `script` tag, in this case it has one propriety, `lang`, defining it as being TS code instead of vanilla JS code. This is one big object that can have several proprieties but being a simple base class, only three are present in the example: `name`; `data`; and `methods`. The `name` holds the component's name. The `data` holds the variables and data needed for the component and is reactive, in this case, it's simply a variable holding a boolean to set the cursor to `wait` mode or default. The `methods` that holds all the methods like the name indicates, in this case only having a setter for the only variable in `data`.

The last root tag in the example is a minimized `style` tag that contains several basic CSS classes that will be used by the component itself, such as the `view` and `cursor-wait` classes but as well by other components rendered in `router-view` due to the `style` tag not having the `scoped` propriety, setting it would isolate the classes only to the single component it is declared on, not letting it be accessible from other components and not polluting these other components CSS namespace.

The component system is central to Vue. In Vue, an application is a tree of components and these components can be easily reused.

The basic structure of a Vue file having been explained, it's now easier to understand the rest of the source code structure.

The `views` folder holds all the Vue components that have a route registered in `vue-router`, this configuration can be found in the, `router` folder, these components have the bare-bones structure and

functionality that is unique to each view.

The `components` folder holds reusable Vue components, such as the header and footer components among other components used in the views.

The `store` folder holds the Vue store from the `vuex` package. This allows to save data in a more permanent and central way, each component can access the store directly without needing the parent or children components to pass them the data. The data in a component only lives as long as the component lives, if a component is deleted the data associated with it is lost, even if that component is then recreated. The store has four main parts: state; mutations; getters; actions.

The state is the stored information, for X-arq Description this is the user information, general information such as the API URL, the currently selected record, etc.

The getters allow us to read something from the store and compute it and cache it. The state can be read directly but if the same operation is being done to one piece of information from the store in multiple places it's better to refactor that into a getter, so that it's only done once and the result will be cached until the dependency of the getter suffers a change.

The mutations are the only way to change the state of the store. This operations are done synchronously and they can't be called directly.

The actions is were the mutations are normally called using the `commit` function. Actions can be asynchronous.

As an example of the store use in X-arq Description is a user logging in. When a user tries to login on the login view of X-arq Description an asynchronous action is called. This action will send the authentication request to the API, if the request is successful it then commits the login mutation. Finally the action returns the request status code to the login view which then depending on the code will either redirect the user to the home page or show an error message.

The `utils` folder holds TS files that define enumerated types, interfaces, etc.

Finally, there is the `assets` folder, this folder holds assets such as images, fonts, that will go through webpack in opposition to the previously mentioned `public` folder that holds assets that won't pass through webpack.

Moving outside of the `src` folder, there's also the `tests` folder. This folder holds the source code of the unit and integration tests done for source code in the `src` folder. Its structure is a mirror `src`, it also contains `components`, `router`, `store`, and `views` subfolders.

4.3.2 Backend

The backend source code structure, has can be seen in fig. 4.2(b), is mainly divided in three folders: `Controllers`; `ModelsApp`; `Services`.

The `Controllers` folder holds the API endpoints, all API requests are received by controllers that are located here. All controllers except for the `HomeController.cs` are for different parts of the API.

The `HomeController.cs` receives the requests for the frontend. The frontend is served by the same server that serves the API. It was decided to not separate the frontend so there's no need for an extra server in production only serving the frontend.

All controllers have a more or less explanatory name. The `AuthenticationController.cs` deals with authentication requests; the `FrdController.cs` deals with requests concerning each record's data, Folha de Recolha de Dados (FRD) means 'Data Collection Sheet'; the `LogController.cs` deals with log requests; and `NavigationController.cs` deals with requests concerning the hierarchy of records, the tree of records.

The controllers then call the necessary functions to respond to the requests, this normally implies a call to a services function. These services are located in the `Services` folder.

The `Services` folder contain most of the business logic of the backend. In this folder there's also a `CachedQueries.cs` file. This file holds queries that are expected to not change, for example, the list of rules. The cache has a 12 hours lifespan.

The `ModelsApp` folder holds the format of the complex types for the requests and responses from and to the frontend, such as `RecordData.cs` that describes how a record's data should be structured for it to be used on the frontend.

4.4 Goal Completeness

Looking back at section 3.2.2.A, there are three groups with a total of 12 well-defined functionalities in the core requirements of X-arq Description. Out of the 12 core requirements, 10 have been fulfilled, one (G2.1 - Records Operations: Creation) has been partially completed, and the only goal that wasn't reached was G2.3 (Records Operations: Change Level).

The Creation (G2.1) functionality has two variants, the basic one, creation of an empty record was fulfilled, while the second variant which consists on copying an existing record and pasting it, thereby creating a copy in another location, wasn't.

The Change Level (G2.3) functionality was supposed to have been implemented using the drag and drop feature directly on top of the navigation tree. The Vue tree component that was selected to be used in the project, LiquorTree[23], has a drag and drop functionality but it hasn't been activated. This functionality has database implications, so simply activating the functionality on the frontend without implementing it on the backend to update the database would only be a detriment as it would put the frontend on an inconsistent state with the state saved on the database. One other reason is that there

[23] https://amsik.github.io/liquor-tree/

are rules to which records can be associated with other records so even in the frontend it's not as simple as just activating the feature on LiquorTree due to moving the record needing to be validated if such change was permitted to happen.

4.4.1 User Interface

Another of the project's goals was to have a new UI, which can be seen in fig. 4.4.

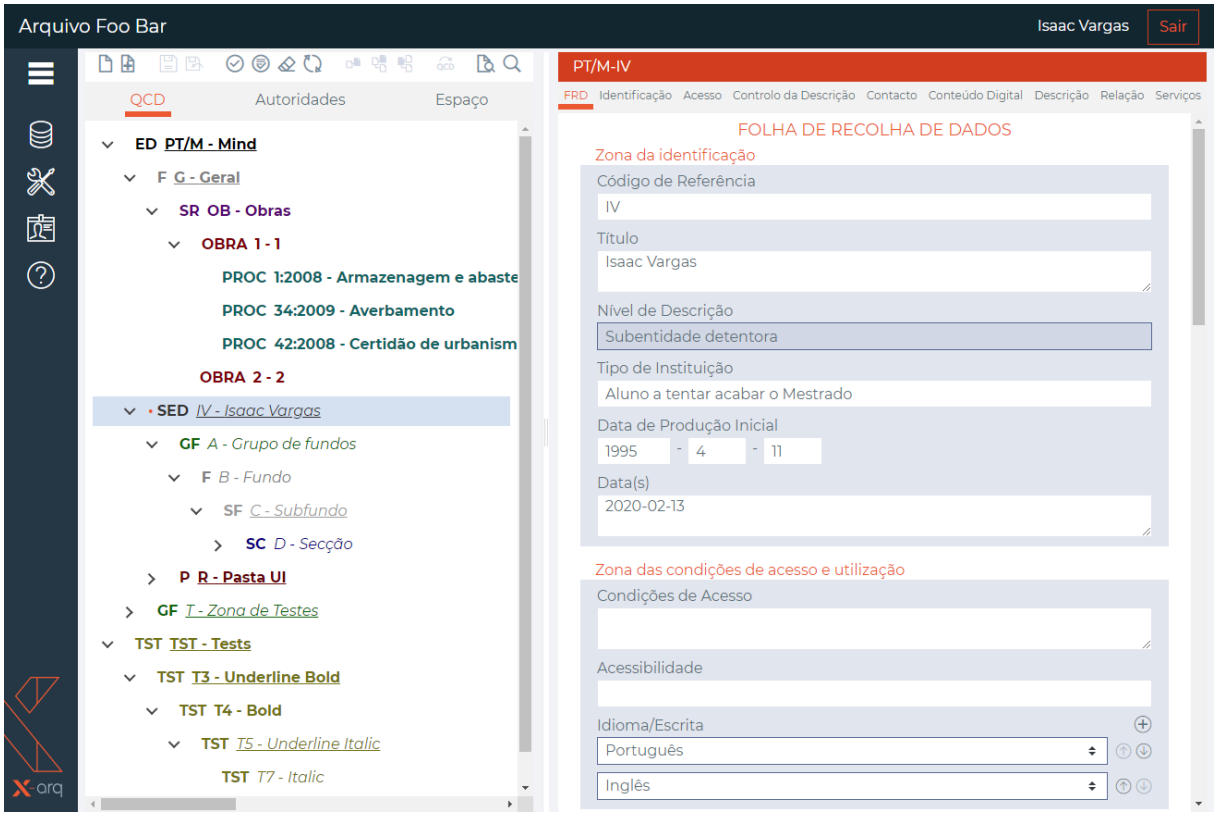


Figure 4.4: Screenshot of X-arq Description.

Back in fig. 3.6, the mock-up for the new UI was presented and while it looks similar to fig. 4.4 there are a few differences which should be mentioned.

Some of these differences between the two are the font, font size, and general colours. Some of these changes were done to improve the website accessibility, as the mock-up wasn't created following accessibility guidelines.

The shortcut bar also changed, from 16 shortcuts down to 14. Four shortcuts were removed and two new ones were added, the validation and hierarchy validation shortcut buttons. Another change was making the two major components, the navigation tree and the FRD component resizable, which can be seen by the addition of the splitter between the two components.

4.5 Implementation

X-arq Description currently has four views but two of these are just empty bare-bones, the `home` and `user` views.

One view that was fully implemented was the `login` view. It has a `header` and `footer` components. The header component can be seen in fig. 4.4. The footer component is simply the copyright symbol, the current year, and the text `Mind` with a hyperlink to Mind's website. The login form is shown in fig. 4.5.

Figure 4.5: Partial screenshot of the login view.

Figure 4.5 is a screenshot of the form after there has been an unsuccessful attempt at authentication. A bootstrap alert in the warning variant appears explaining why the attempt was unsuccessful.

The final implemented view of X-arq Description is the `archive` view which can be viewed in fig. 4.4. This view also has the `header` component like the other three views but unlike the other views it does not have the `footer` component.

On the left edge of fig. 4.4 there are icons for the menus due to them being important for the structure of the view. The menus and their functionalities aren't implemented because they weren't inside the scope of the project.

Moving to the main part of the `archive` view, it is split into two parts that are resizable. This was a feature that was introduced after the original design of the UI. It was implemented by using Splitpanes[24], an open-source Vue package that implements this behaviour and is very simple to use.

On the left side of the main, the tree side, there are three sections: the shortcuts bar; the area tabs; and the navigation tree section.

While the shortcut bar is in the tree side, not all of the operations on the bar only affect the navigation tree. The second grouping of shortcut buttons holds two buttons, a save and another to cancel the current unsaved changes, these two buttons interact with the right side of the main, the FRD side. Another more extreme example, the last two buttons on the shortcut bar deal with X-arq Web, which is

[24] https://antoniandre.github.io/splitpanes/

external to X-arq Description.

The second to last button in the shortcut bar opens a modal that shows the selected record in X-arq Web in visualization mode, this mode is similar to the one seen in archeevo in fig. 2.1. This modal is shown in fig. 4.6. The last button opens the X-arq Web homepage in a new tab.

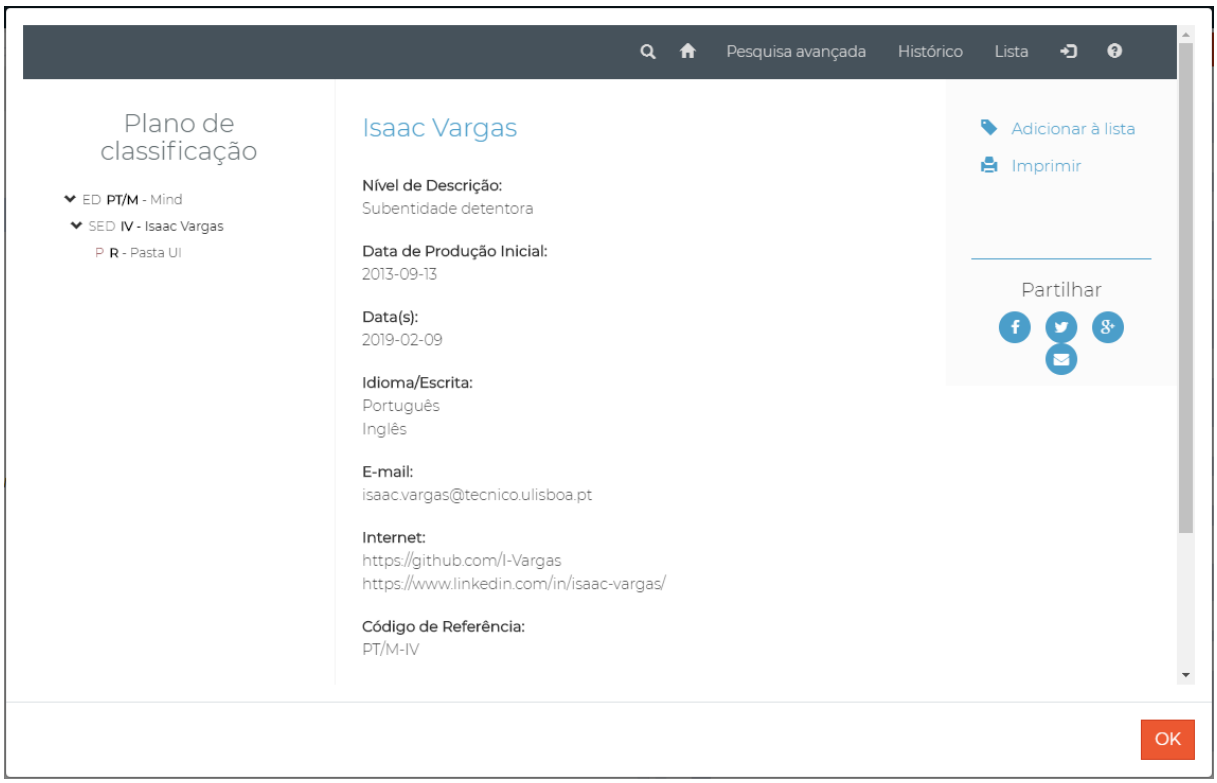


Figure 4.6: Partial screenshot of the X-arq Web modal in X-arq Description.

4.5.1 Goal 1: Permissions

This goal was implemented on the backend for all sub-goals and a few also on the frontend. For example, Lock (G1.5), if the user requests a locked record that information is transmitted to the frontend and the data is also locked so that the user can't update it. This is also checked when trying to save the record. If by some reason a request to update a record is received but the user doesn't hold the lock then the update is rejected by the backend.

For Authentication (G1.1), the implementation used was the already existing one in the original X-arq.

For Component (G1.2) and Record(G1.3), when an authenticated user requests the list of records there's always a check so that only the records and components the user has access to return.

For Operation (G1.4) just like for Lock (G1.5), the frontend will disable the operations that the user doesn't have permission to execute. When a user logs in the list of operations the user can do is included

in the authentication response and is stored by the frontend.

For Permissions, outside Authentication, it was mainly about making sure the SQL queries include checking the permissions the user has and only selecting that which they have access to.

4.5.2 Goal 2: Record Operations

This goal as previously mentioned wasn't completed in its entirety. Goal Change Level (G2.3) wasn't done and Creation (G2.1) was only partially done. This was due to running out of time during the project's duration.

The half that was implemented for Creation was creating a new record from scratch, instead of making a copy of another record. Creating a new record from scratch can be accomplished in two different ways. Creating a child of the selected record or creating a sibling of the selected record, when each case is possible, it's not always possible to create children or siblings for some types of records. On creating a new record a blank FRD is shown that then the user fills in and when the user tries to save the record the backend checks that such operation can occur and that the user has the permission to create it before saving it to the database.

The remaining goals, Validation (G2.2), Allow Display Content on the Internet (G2.3), and Deletion (G2.5), were again implemented both on the backend and on the frontend as mentioned in the previous section. When a request for one of these operations is received by the backend it's checked if the user truly has the permission to do such operation.

4.5.3 Goal 3: Data Management

For the Data Management goal, unlike the previous two goals, this is a frontend-only goal. All the data is saved in the same way in the backend but the data should be visualized and modified differently depending on the rules associated with each field and subfield.

On section 4.3.1 it was explained how the source code for the frontend was set up, how it's hierarchical. To expand on that, at the root is the `app-root.vue`, which has one child `archive` view, which has three components: `header`; `navigation` (tree); `FRD`.

The `FRD` component is the starting point for this goal. Inside the `FRD` component there is the `field` component which has the `subfield` component inside.

A FRD is divided into zones. These zones group fields of the same context so this only has Visualisation (G3.1) implications. It was on the `FRD` component that this was implemented.

Inside each zone of the FRD, there are some fields. Each field may have rules that are associated with them. The rules can be visual-only, such as a rule stating the number of lines of text that should be visible. They can apply constraints, such as the mandatory rule. They can be both at the same time

such as the selection rule, that states the field should be shown with a dropdown and the only acceptable values are the ones supplied as options.

Inside each field, the same thing happens with the subfields that can also have rules associated with them. The list of rules that can be applied to fields and subfields aren't equal.

Once a record is selected, the type of FRD of that record is passed to the `FRD` component that will then fetch the definition of that FRD and show the zones and create the fields. The fields then look at the received rules and show the correct visualisation and create the subfields. The subfields then look at the received rules and present the correct visualisation.

Once the `FRD` component has the visualization information, it passes the data of each field down to it, which then passes the pertaining data to each subfield down to the correct subfield.

When a subfield is modified it triggers an event that is captured by the field with the modification which is propagated upwards. The data is passed downwards directly but it moves upwards by events.

5

Evaluation

Contents

With creating software there's also a need to test it. In this chapter the testing and evaluation of the work done is exposed. The technical tests were carried out in conjunction with the development of the software while its evaluation was done on the last weeks of the development.

5.1 Technical

For the technical tests, several types can be done: unit; integration; functional; End-to-End (E2E); load; penetration; etc.

On the frontend two types were done, unit and integration. These tests were done with Vue Test Utils[1], a testing utility library, on top of Jest[2], a JS testing framework.

At the end of the project it was using version 1.0.0-beta.31 of Vue Test Utils but it started on version 1.0.0-beta.29. This version had a major bug, it had a feature, sync, that tried to make asynchronous code run synchronously but it wasn't working correctly and was then removed. This is documented in issue #1137[3].

The project suffered with this bug and quite a lot of time was lost trying to deal with it as well as trying to see when a test failed if it was because of the bug or the underlying code was wrong.

On the backend there were tests planned to have been done with NUnit[4] but testing the frontend was given priority. After working on the frontend tests and dealing with the difficulties that came up and seeing the time to complete the project keep going down it was decided to be left for future work as the software is currently still only a prototype with the core functionalities, to keep adding those core functionalities was deemed higher priority than to test the backend.

Jest using Istanbul[5] can be configured to report the test coverage of the project. Istanbul reports the coverage on each file and directory, as well as giving a summary and the stats on the number of tests suites, tests, and snapshots that passed, skipped, or failed, and the total.

The following three tables are reporting the coverage of the project when running both the unit and the integration tests.

The current coverage is almost 100% in each category, only missing the branches. The branches category isn't 100% due to the tests loading five variables from the environment process. To achieve 100% the tests would need to run twice. Once loading the variables from the environment process and the other using the fallback branches.

[1] https://vue-test-utils.vuejs.org/
[2] https://jestjs.io/
[3] https://github.com/vuejs/vue-test-utils/issues/1137
[4] https://nunit.org/
[5] https://istanbul.js.org/

Files in directory	% Statements	% Branches	% Functions	% Lines
All files	100	99.17	100	100
src	100	100	100	100
src\components	100	100	100	100
src\router	100	100	100	100
src\store	100	91.07	100	100
src\views	100	100	100	100

Table 5.1: Combined unit and integration testing coverage by directory.

Type	% Coverage	# Covered	# Total
Statements	100	1142	1142
Branches	99.17	595	600
Functions	100	303	303
Lines	100	1138	1138

Table 5.2: Combined unit and integration testing coverage summary.

5.1.1 Unit

For unit testing, the procedure was to mock everything not directly under testing. For example, testing a function that called a getter of the store and then made a request to the back end and then called a function that returned a value and then called another function that is void. The getter, the backend request, both the calls to other functions and the returned value of the called function were mocked. Just testing that one function without it propagating to other elements.

No major obstacles appeared during unit testing. Following there are two tables with the coverage summary and stats.

5.1.2 Integration

For integration testing the procedure was to only mock the backend requests and nothing else, allowing for all the frontend components to interact and change information.

On integration testing, there were two issues, one minor and one major.

The minor issue is due to trying to use and modify the ValidityState[6] of the HTML Document Object

[6] https://developer.mozilla.org/en-US/docs/Web/API/ValidityState

Type	Passed	Skipped	Failed	Total
Suites	24	2	0	26
Tests	656	68	0	724
Snapshots	131	0	0	131

Table 5.3: Combined unit and integration testing coverage stats.

Type	% Coverage	# Covered	# Total
Statements	99.47	1136	1142
Branches	98.50	591	600
Functions	99.34	301	303
Lines	99.47	1132	1138

Table 5.4: Unit testing coverage summary.

Type	Passed	Skipped	Failed	Total
Suites	14	0	0	14
Tests	499	0	0	499
Snapshots	70	0	0	70

Table 5.5: Unit testing coverage stats.

Model (DOM) form elements such as input, textarea, etc. The reason to modify the ValidityState of the form elements was to set the error messages in Portuguese instead of using the default English ones.

The major issue appears to be due to component used for the tree visualisation, LiquorTree.

On the integration testing for the `navigation-section.vue` component and the view that uses it, `archive-page.vue`, when running all the tests some of them will fail and rerunning the tests without changes will sometimes make other tests fail instead of the original ones. Running each individual test alone will show that the tested code is correct as the test will pass.

The true cause and a fix for this problem hasn't been found and as such these components while they have tests written for them that pass if run individually for the run used to obtain the data in the following two tables they were set to be skipped over.

Type	% Coverage	# Covered	# Total
Statements	52.89	604	1142
Branches	52.17	313	600
Functions	57.76	175	303
Lines	52.72	600	1138

Table 5.6: Integration testing coverage summary.

As mentioned the two suites of tests that use the LiquorTree component and the tests contained in them were skipped over giving us only about 50% integration coverage.

5.2 Functional Requirements

As mentioned in several other sections, such as section 3.2.2.A and section 4.4, this project looked at 12 FRs. At the end of the project, only 10 were completed with another one partially completed and one that wasn't done.

Type	Passed	Skipped	Failed	Total
Suites	10	2	0	12
Tests	157	68	0	225
Snapshots	61	0	0	61

Table 5.7: Integration testing coverage stats.

These FRs were tested manually by an area colleague inside Mind in the last weeks of the project. While doing these tests naturally some bugs were discovered and reported which then were fixed and retested again.

A spreadsheet that specified each functionality and several operations under that functionality was used to track these manual functional tests. Each operation was then tested one by one and evaluated. When a failure was discovered it was reported for an analysis to be done to find the fault that cause the error and afterwards implement a bug fix to resolve it.

5.3 Nonfunctional Requirements

Like the FRs, the NFRs were only tested at the end of the project, but unlike the FRs, there weren't many actions derived from their results, the focus was on the FRs.

Four categories of NFRs were planned with a focus on three different areas: the user, with accessibility and usability; the application, with performance; and the source code with testability.

They were chosen due to various reasons and according to different criteria but make for a basic rounded overview of X-arq Description NFRs.

5.3.1 Accessibility

The accessibility of the application was tested in two different tools: AccessMonitor[7], which is provided by the Portuguese Government; and Lighthouse[8], from the Google Chrome DevTools.

The AccessMonitor tool verifies that the website follows the WCAG 2.1 directive, checking the accessibility of the websote. Lighthouse is broader and not only checks accessibility but also audits performance, Search Engine Optimization (SEO), and best practices.

X-arq Description scored a 6.3 out of 10 in AccessMonitor and a 88 out of 100 in Lighthouse. Which means there is still room for improvement on this, taking in special care the report from AccessMonitor.

For the sake of comparison and providing a baseline, here are the scores of three more websites: Portuguese Government[9], 7.9 and 80; Diário da República Eletrónico (DRE)[10], 8.5 and 95; Caixa Geral

[7] http://accessmonitor.acessibilidade.gov.pt/amp/
[8] https://developers.google.com/web/tools/lighthouse/
[9] https://portugal.gov.pt/
[10] https://dre.pt/

de Depósitos (CGD)[11], 9.0 and 83. These results were taken in February 2020.

The DRE has a webpage[12] about accessibility where you can see it follows the W3C WAI-AA (WCAG 2.0) and the CGD website has a AccessMonitor AAA level seal at the bottom.

5.3.2 Performance

One failure of this project that needs to be rectified before production are the SQL queries.

Most queries used by X-arq Description were new queries. Near the end of the project, X-arq Description was connected to production-like database, which has a huge amount of data instead of using the development database that is nearly empty only having a few records of each type. The API response time then increased hugely rendering the application near unusable. The original X-arq doesn't suffer from this problem using the production-like database.

An investigation needs to be done to determine the root cause of this degradation of performance. Optimize the queries, check that they are using the existing indexes, and if not create new indexes for them, etc.

No measurements were taken to quantify the performance of either application, due to it being very clear that the new queries were significantly worse, there was no need to even measure it.

On a whole there is still a lot of X-arq Description parts that need to be analysed under the performance optic but during development the only one that stood out even without doing a serious analysis were the queries.

5.3.3 Testability

At the end of the project, only half of the system had been tested.

The backend was planned to be tested using the NUnit framework but due to priority decisions, it got pushed back. The backend is a simple API, in that it has very few controllers and only two major ones: `FrdController`; and `NavigationController`.

The controllers themselves only have a few endpoints that only received around three arguments. While there doesn't seem to be any major issues that should arise when testing it, the frontend also seemed to be straight-forwarded but then indeed there were some problems. So the testability of the backend can't be determined.

The frontend has been heavily tested as discussed in the previous section 5.1, which was where the most time was invested while testing X-arq Description.

While Vue Test Utils can still be improved to make tests with more reliability and testing Vue applications smoother, it already provides a good base. The project testes were done with the 1.0.0-beta

[11] https://www.cgd.pt/
[12] https://dre.pt/acessibilidade

versions, but since then version 1.0.0 has been released and work has already started on version 2.0.0 with alphas that will be used to target Vue 3 that is also in developement right now.

While the frontend has mostly high coverage, the major issue in the integration tests means that it can't be said that it is highly testable.

The original X-arq didn't have tests, so the current state is already a big improvement over it.

5.3.4 Usability

For usability, the original proposed plan was for X-arq Description to be deployed as a prototype in real Mind clients using the original X-arq. This plan did not work for a multitude of reasons, one being that the clients that use X-arq are government organizations and therefore are quite bureaucratic and it was deemed that it was not feasible in the time frame of the project due to its short duration.

A backup plan was then for colleagues in Mind that had knowledge of the original X-arq to do this usability analysis after all the core functionalities had been implemented and the FRs evaluation had been done. In the end, this plan also wasn't realised due to several reasons, the major one being that not all core functionalities had been completed by the end and that the colleagues with knowledge of the original X-arq all had high workload and then taking time to evaluate an incomplete X-arq Description wasn't a priority.

6

Conclusions and Future Work

Contents

In this final chapter, a retrospective about the project is presented and conclusions are drawn from it as well as some thoughts about the future of it.

6.1 Conclusions

Several challenges appeared during the development of X-arq Description. Some were external such as the sync bug in Vue Test Utils, while others were internal such as the bad performing queries. Some have been resolved, some have not.

The project had 12 objectives, but only 10 were completely fulfilled with another one being partially done. So progress hasn't been fast.

Having reflected on the occurred shortcomings during this project there are also some positives, such as Mind liking how X-arq Description was turning out even with the previously mentioned setbacks.

X-arq Description was developed to update X-arq to the reality of today's world. Using Vue.js, a vanguard in the JS framework area, for the frontend brought considerable aesthetic improvements as well as testability and maintainability among others.

Using C# and ASP.NET Core also brought many of the same improvements such as testability and maintainability. The backend is now simpler than the original X-arq due to several reasons, one being that there is an actual separation of the backend and frontend now, among others.

All of the source code is now cleaner and has clear and divided responsibilities. This makes it easy to understand and maintain as well as easy to build upon it and add new features or modify current behaviours.

So while not all goals were reached and there was a misstep with the performance of the queries, the project's structure is in an excellent state making it easy for future work on X-arq Description.

6.2 Future Work

As mention in previous sections, there are still actions needed to improve X-arq Description.

The two major actions are finishing the implementation of the last two core functionalities that still need to be completed, as well as finishing the analysis and improvement of the database queries that simply don't have enough performance to deal with the amount of data that the archives hold.

Some medium priority actions should be taken, mainly dealing with NFRs such as improving accessibility and conducting the usability analysis as well as implementing the backend API tests.

This future work was just about the core of X-arq Description. Afterwards, there are still more functionalities that need to be added, the previously mentioned that were categorized as complementary and add-on functionalities in section 3.2.

Bibliography

[1] International Organization for Standardization (ISO). (2011, November) ISO 30300:2011: Information and documentation — Management systems for records — Fundamentals and vocabulary. Accessed on 2020-01-24. [Online]. Available: https://iso.org/standard/53732.html

[2] ——. (2016, April) ISO 15489-1:2016: Information and documentation — Records management — Part 1: Concepts and principles. Accessed on 2020-01-24. [Online]. Available: https://iso.org/standard/62542.html

[3] ——. (2017, October) ISO 23081-1:2017: Information and documentation – Records management processes – Metadata for records – Part 1: Principles. Accessed on 2020-01-24. [Online]. Available: https://iso.org/standard/73172.html

[4] ——. (2010, December) ISO 16175-3:2010: Information and documentation — Principles and functional requirements for records in electronic office environments — Part 3: Guidelines and functional requirements for records in business systems. Accessed on 2020-01-24. [Online]. Available: https://iso.org/standard/55792.html

[5] ——. (2017, September) ISO/IEC/IEEE 24765:2017: Systems and software engineering — Vocabulary. Accessed on 2020-01-27. [Online]. Available: https://iso.org/standard/71952.html

[6] ——. (2011, March) ISO/IEC 25010:2011: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. Accessed on 2020-01-27. [Online]. Available: https://iso.org/standard/35733.html

[7] Directive (EU) 2016/2102 of the European Parliament and of the Council of 26 October 2016 on the accessibility of the websites and mobile applications of public sector bodies. Accessed on 2020-02-17. [Online]. Available: https://eur-lex.europa.eu/eli/dir/2016/2102/oj

[8] Diário da República. (2018, October) Decreto-Lei n.º 83/2018. Accessed on 2020-02-17. [Online]. Available: https://data.dre.pt/eli/dec-lei/83/2018/10/19/p/dre/pt/html

[9] World Wide Web Consortium (W3C). (2018, June) Web Content Accessibility Guidelines (WCAG) 2.1. Accessed on 2020-02-17. [Online]. Available: https://w3.org/TR/2018/REC-WCAG21-20180605/

[10] Penny Grubb, Armstrong A. Takang, *Software Maintenance: Concepts and Practice*, 2nd ed. World Scientific Publishing Company, 2003.

[11] Shari Lawrence Pfleeger, Joanne M. Atlee, *Software Engineering: Theory and Practice*, 4th ed. Prentice Hall, 2009.

[12] Erik DeBill. Amount of packages in package manager registries. Accessed on 2020-02-24. [Online]. Available: http://www.modulecounts.com/