X-arq *Webification*

Isaac Macedo Vargas
isaac.vargas@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

September 2020

Abstract

In today's world, the Internet and the World Wide Web is pervasive in society, an example of this is the Internet of Things, 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 thesis. It's a web application that was built using Vue.js, a JavaScript framework.

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

1. Introduction

X-arq, the name being derived from "extended archive", is an archival management software application by Mind, 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 among others.

Currently, X-arq is used by over 50 institutions, some of them are the Arquivo Municipal de Lisboa, 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.

1.1. Motivation and Objectives

X-arq is an old desktop application, started in 2001, that is build with old technology with an user interface that looks outdated, and it has some quirks that degrade the user experience.

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 and user experience as well as allowing for the system to be accessed remotely.

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 and not simply a web application.

X-arq Description being a progressive web application brings additional properties that are useful for X-arq Description, mainly the caching of the static parts of the user interface which allows for a faster and responsive user experience for successive uses of the application.

The frontend is currently in very good shape, it has a good structure that is easy to understand, 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. It was all done from scratch, being a brand new user interface.

The backend has a very simple and linear structure with the responsibilities of each component clearly defined making it easy to understand and added upon. The database queries used have to be refined, currently they don't have the necessary performance.

1.3. Project Structure

This project 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 solution compliance with the requirements.

6. **Conclusions and Future Work:** The knowledge obtained during the project and future work.

2. Related Work

In this section, 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"[5]. A records system is defined as "information system which captures, manages and provides access to records over time"[5, 7].

This software needs to have certain capabilities, the main three are: being able to capture[8]; do records management[5, 7]; and have access[5, 7]. It also has to ensure that each record is classified, retained, and disposed following the provided information long with other concepts[5, 7, 4].

These concepts are all defined and standard by different standard authorities, by the previously mention International Council on Archives, Direção-Geral do Livro, dos Arquivos e das Bibliotecas, Open Archives, and International Organization for Standardization.

2.2. Market Solutions

In the current market there are two other main solutions, namely archeevo, by KEEP SOLUTIONS, and gead, by Libware. There's also an open source international solution, Access to Memory (AtoM) that is currently maintained by Artefactual Systems after being originally built with the support of International Council on Archives.

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 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.

3. Analysis and Architecture

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#. It 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 X-arq user interface is in Portuguese and no other languages are provided which means that some acronyms maybe be used in their Portuguese version.

The main view of X-arq has six zones. There are the menus. The actions shortcuts. The three navigation areas that the users have access. The zone with the hierarchy of records that the user can visualize. The zone where the records' data is exhibited.

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

For 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.

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

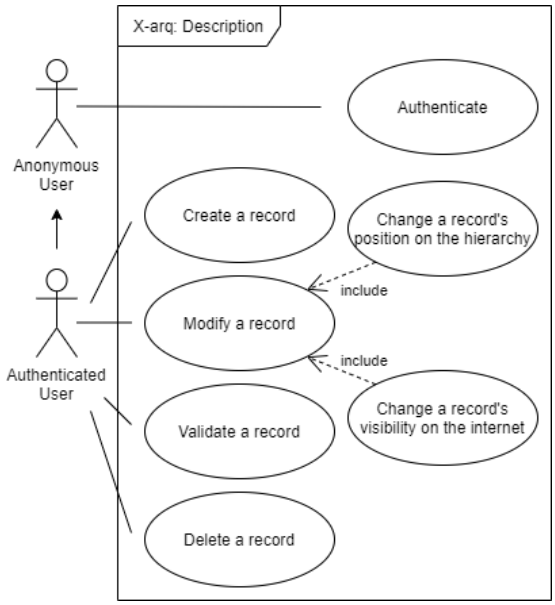

Figure 1: Use case diagram of X-arq Description.

be, for example, connecting X-arq Description with Kapture.

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[9].

A functional requirement 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[9]. A functional requirement 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 only access the components they have permission.
  - **G1.3: Record -** users can 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, meaning that no one else is accessing that record.
- **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 to be a child of a record that accepts that type of record.
  - **G2.4: Allow Display Content on the Internet -** a record can be set to be viewable through the internet.
  - **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 functional requirements. These were the expected requirements to be completed in the period of this project.

A nonfunctional requirement is a software requirement that describes not what the software will do but how the software will do it[9].

Examples of nonfunctional requirements are quality attributes that end with ”ility” such as availability, interoperability, etc.

Several of nonfunctional requirements 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 nonfunctional requirements 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 publishes directives 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[1]. This directive was implemented by the Portuguese Government with Decreto-Lei n.º 83/2018[2].

The Decree Law sets the accessibility guidelines to follow as the AA level of the Web Content Accessibility Guidelines 2.1 created by the World Wide Web Consortium[12].

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 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 user experience 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 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 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.

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.

3.3.1 Deployment Diagram

X-arq Description follows a three tier architecture.

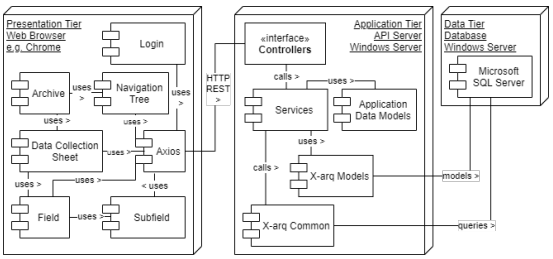

Figure 2: Deployment diagram of X-arq Description.

The first tier, the presentation layer that is client-side and can be accessed using a web browser. The main modules on the presentation tier are: `login`; `archive`; and `axios`. The `axios` module is a dependency, it is a HTTP client, and is used to make all requests to the backend, the API. The `login` and `archive` modules are Vue views. The `login` vie is used for authentication while `archive` is the main view. The other modules make a tree with `archive` as the root.

In the middle layer 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 REST HTTP requests using the JSON format. These requests are received by the controllers that then call the services. The services are where the business logic is implemented.

A 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.

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

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** is a JavaScript 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** is a progressive JavaScript framework for building user interfaces and single-page applications.
  - **TypeScript** is a strict syntactical superset of JavaScript which adds optional static typing to the language. It is then transcompiled into JavaScript.
- **.NET** 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 object-relational mapping to ease communication with the database and the manipulation of the data by use of the Entity Data Models.
- **Microsoft SQL Server** is a relational database management system 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 that serves has the basis for the web and the database server.
  - **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.

The ones used in the frontend, the Node (Node.js) group are all new as well as ASP.NET Core and Entity Framework 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 object-relational mapping and it was build using Microsoft Foundation Class Library which is now being replaced by ASP.NET Core.

3.3.3 User Interface

A mock up of how the new user interface 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.

Development of X-arq Description was made following the mock-up as the guide.

4. Solution

In this section, 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 user interface Design
- Implementation Process and Technical Testing
- Functional Testing

This project looks at all four phases but in this section, 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 to manage their projects as well as for version control. Team Foundation Server has several functionalities, it's possible to create sprints and Product Backlog Items and associate them with a developer, etc.

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[3].

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[3].

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

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.

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 Entity Framework 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 Entity Data Models 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

While the server-side code, the backend, is in the root of the C# project, the important source code is in three folders: Controllers; ModelsApp; Services.

4.3.1 Frontend

On the frontend 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 that guarantees that all the frontend code in the project follows the same style guide.

In the `public` folder, files in this folder won't go through webpack, 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 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.

The main folder of the frontend is `src`, it contains six folders as well as two files. The `main.ts` file that is the entry point of the application, it will 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, these files have three root tags: `template`; `script`; and `style`. In the interior of each of these respective tags, there is the HTML markup, a JavaScript dialect in this project being TypeScript, and the CSS classes.

4.3.2 Backend

The backend source code structure 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 how a record's data should be structured in `RecordData.cs`.

4.4. Goal Completeness

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 consists on copying an existing record and pasting it, creating a copy in another location, wasn't.

The Change Level (G2.3) functionality was supposed to have been implemented using drag and drop directly on top of the navigation tree. The Vue tree component that was selected to be used in the project, LiquorTree, 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 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.

5. Evaluation

With creating software there's also a need to test it. In this chapter, the testing and evaluation of the work done are 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; load; etc.

On the frontend two types were done, unit and integration. These tests were done with Vue Test Utils on top of Jest.

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.

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 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 can be configured to report the test coverage of the project. 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.

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

Table 1: Combined unit and integration testing coverage summary. (Covg is Coverage)

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

Table 2: Combined unit and integration testing coverage stats. (Snaps is Snapshots)

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 is the table with the coverage summary.

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

Table 3: Unit testing coverage summary. (Covg is Coverage)

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 of the HTML Document Object Model 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 table they were set to be skipped over.

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

Table 4: Integration testing coverage summary. (Covg is Coverage)

5.2. functional requirements

As mentioned in several other sections, this project looked at 12 functional requirements. At the end of the project, only 10 were totally completed with another one partially completed and one that wasn't done.

These functional requirements 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 in order for an analysis to be done to find the fault that caused the error and afterwards implement a bug fix to resolve it.

5.3. Nonfunctional Requirements

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

Four categories of nonfunctional requirements 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 nonfunctional requirements.

5.3.1 Accessibility

The accessibility of the application was tested in two different tools: AccessMonitor, which is provided by the Portuguese government; and Lighthouse, which can be used by accessing Google Chrome DevTools.

The AccessMonitor tool verifies that the website follows the Web Content Accessibility Guidelines 2.1 directive, checking the accessibility of it. Lighthouse is broader and not only checks accessibility but also audits performance and more.

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.

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 a 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.

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, 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 1.0.0-beta versions, but since then version 1.0.0 has been released and work has already started with alphas on version 2.0.0 that will be used to target Vue 3 that is also being developed 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 an 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 functional requirements 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 at then taking time to evaluate X-arq Description wasn't a priority.

6. Conclusions and Future Work

In this final section, 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 also had 12 objectives, but only 10 were completely fulfilled with another one being partially done. So progress hasn't been fast.

Having reflected on some of the shortcomings during this project there are also some positives, such as Mind seeming to like how X-arq Description was turning out.

X-arq Description was developed to update X-arq to the reality of today's world.

Using Vue.js, a vanguard in the JavaScript 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 several of the previous sections, there are still several actions needed.

The two major actions are finishing the implementation of the last two core functionalities that still need to be completed, as well as rewriting 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 Nonfunctional Requirements 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.

References

[1] 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.

[2] Diário da República. Decreto-Lei n.º 83/2018, October 2018. Accessed on 2020-02-17.

[3] Erik DeBill. Amount of packages in package manager registries. Accessed on 2020-02-24.

[4] International Organization for Standardization. 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, December 2010. Accessed on 2020-01-24.

[5] International Organization for Standardization. ISO 30300:2011: Information and documentation — Management systems for records — Fundamentals and vocabulary, November 2011. Accessed on 2020-01-24.

[6] International Organization for Standardization. ISO/IEC 25010:2011: Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models, March 2011. Accessed on 2020-01-27.

[7] International Organization for Standardization. ISO 15489-1:2016: Information and documentation — Records management — Part 1: Concepts and principles, April 2016. Accessed on 2020-01-24.

[8] International Organization for Standardization. ISO 23081-1:2017: Information and documentation – Records management processes – Metadata for records – Part 1: Principles, October 2017. Accessed on 2020-01-24.

[9] International Organization for Standardization. ISO/IEC/IEEE 24765:2017: Systems and software engineering — Vocabulary, September 2017. Accessed on 2020-01-27.

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

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

[12] World Wide Web Consortium. Web Content Accessibility Guidelines 2.1, June 2018. Accessed on 2020-02-17.