Web-based platforms for management of museological assets

Duarte Miguel Ferreira Dias
duarte.ferreira.dias@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2019

Abstract

Asset management in museums has seen continuous improvements. With the advancements made in technology collection museums have evolved from using paper based registries to using digital systems that facilitate a collection manager's life as well the efficiency of performing modification to the asset base and registering these modifications.

Despite the currently available systems for asset management within museums, there is not a well defined best solution since these assets contain very diverse properties and requirements when cataloguing and storing their information.

The motivation for this work comes from the desire and necessity of creating a single highly adaptable and scalable solution capable of giving response to most museums.

This work has two main objectives: provide a long-term term solution that can be adapted for all museums to use in order to deal with their requirements when managing their collection of assets, and a short-term solution of providing a system capable of serving as a tool to solve IST's immediate need to catalogue its collection of assets from its multiple collections.

The motivation for the long-term objective comes with the current state of asset management solution, with most platforms providing a closed solution which is not meant to be adapted.

For the short-term objective, we are faced with IST, a university with a vast collection of assets from multiple areas which are not currently catalogued.

This work also provides insights into some of the currently used procedures when it comes to asset management within museums and how they differ from each other.

Keywords: Museum; Database; Asset Management; Extensibility; Web; Vue.js; Node.js; MongoDB

1. Introduction

Asset management is a crucial part of any museum. For a museum manager it is imperative to have the capability of inserting, viewing, editing and removing any asset at any given moment.

In the past, the main way of managing these assets was through paper registries, but with the emergence and development of information technologies we are now able to perform these tasks more efficiently and without as big of a risk of losing the information.

Despite the technological progress, there is yet to be defined an ideal way of dealing with museological assets mainly due to their complexity and diversity. There exists already some software for performing these tasks but they are generally closed in the sense that the client is unable to adapt the platform to fit its specific needs.

The motivation for the development of this work comes with the anaylsis of a particular asset management case, more concretely, the one at Instituto Superior Técnico (IST). IST is a portuguese engineering college with a wide range of museological assets to manage, despite the lack of an estabilished system required to perform these necessary tasks.

As previously mentioned, one of the biggest challenges comes with the diversity of assets. IST is a perfect example of this scenario, having assets that range from rock and minerals all the way to motors and electronic devices. Another challenge comes with the lack of procedures which are currently not defined for IST.

Also, having an immediate need for such a management system, IST is the ideal case study to focus this work upon.

There already exist other systems in the market, such as closed-code products developed and sold by private companies, as well as opensourced solutions developed by the community. But still, with all the available solutions, none of them gives the necessary response to an institution such as IST.

2. Background

One of the difficulties of cataloguing the assets of museums comes with the vast amount of specific locations an asset can be in. Museums such as the

museum of Geology and Mineralogy have most of their assets within cabinets, drawers and sections. This requires the documentation of the asset to be very precise in stating its location.

Another difficulty is related to the amount of properties associated to assets of different categories. The properties used to catalog and identify a painting and a mineral will be very different.

The thesis [1] developed by Miriam resulted on a Microsoft Access document with information on some of the assets of the museum of Geology and Mineralogy at IST. The document includes 3 different tables: identification, characteristics and history. These are the fields used to describe assets of this museum:

Identification

- Inventory number
- Inventory A
- Inventory B
- Inventory C
- Designation 1
- Designation 2
- Designation 3
- Brand
- Collection
- Location

Characteristics

- Inventory number
- Height
- Depth
- Diameter
- Weight
- Wood material
- Glass material
- Metal material
- Plastic material
- Other material
- Price
- Price assigner

History

- Inventory number
- Acquisition date
- Acquisition method
- Acquisition price
- Supplier
- Conservation
- Repair
- Bibliography
- Exhibitions
- Sites 1
- Sites 2
- Sites 3
- Sites 4
- Documentation

Spectrum

Spectrum [2] is a collection management standard developed in the UK that is also used in other places around the world with the latest version having been published in September of 2017. According to its creators, Spectrum gives "tried-and-tested advice on the things most museums do when managing their collections".

This standard is activity-oriented, it focuses on how to register and execute certain activities, such as moving an object or updating location records. These activities are called procedures and Spectrum has a total of 21 defined procedures.

Each procedure has the following elements:

- A definition that sums up the procedure into a single sentence.
- An explanation on the scope, explaining when the procedure should be used.
- The spectrum standard, that consists on the goal that we plan on achieving.
- A suggested workflow, this can be a diagram or a text written explanation.

When it comes to concrete procedures, these are the ones that Spectrum considers to be its primary procedures:

- Object entry - Registering incoming assets taking in to account the entry method (acquisition, loan, ...).
- Acquisition and accessioning - Taking legal ownership of objects in order to add them into the museum's collection.
- Location and movement control - Keeping track of the locations of all assets and updating them each time the assets are moved.
- Inventory - Making sure you have the basic information in order to identify your objects.
- Cataloguing - Managing the assets' information.
- Object exit - Recording when, the assets you are responsible for, exit your collection.
- Loans in - Managing objects you borrow for a fixed period of time and for a specific purpose.
- Loans out - Managing the requests for you to lend assets as well as the lending process until assets are returned to you.
- Documentation planning - Ongoing improvement of your documentation systems.

Taking this into account, Spectrum is not one single document but the composition of multiple documents, being that each one of these is associated to a specific procedure.

Spectrum is not a software itself but a set of rules that you should follow in order to create a digital of paper system that complies this management standard. There is not a single way of implementing this standard as we can see in the documents associated to the procedures, but as long as the base rules are followed we can consider a system to be compliant with this standard.

3. System Description

The short term objective of this work is to provide a solution for IST's immediate need of being able to know what assets it currently owns. In order to do so, we require some minimum information such as the asset's identification/title, location and the collection it belongs to.

In the medium to long term, the objective of this work is to provide an easily scalable asset management platform where developers are capable of, without much effort or prior knowledge, adding new information specific to new groups of assets as well as adding other new features to the system. Another long term objective is to then integrate the platform with other applications besides the FenixEdu system.

The proposed platform will follow a web architecture as shown in figure 1.

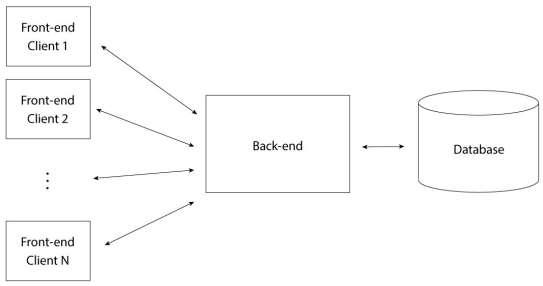

Figure 1: Commonly used web architecture that will serve as template for the platform.

Data Model

The main focus of this system is the way in which we will model our data. The most intuitive way of doing it would be having, for each asset, a node of information with all possible properties that the asset can have and only fill in the relevant properties for that specific asset. This would simplify our data structure since we would have the same structure for every single asset and would not have to create different structures for different assets. The problem here is that we would end up with vast amounts of irrelevant data fields as well as many properties without an associated value.

A possible way of tackling this problem would be of creating a solution like the one presented in figure 2.

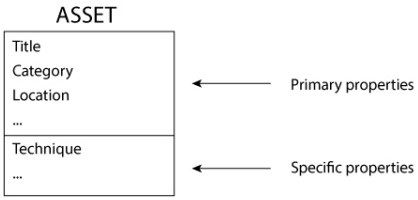

Figure 2: Initial data structure

In this data structure, we would define primary properties that would appear in all assets as well as a couple properties specific to the current asset. The specific properties would be conditional of the primary ones. This way, certain specific properties would be associated main properties' values. For example: an asset with the category of painting would only have, besides the primary properties associated to every assets, properties associated to paintings.

In this solution, an asset would have different sections (primary and specific ones) that are independent of each other. When relating this with the Spectrum standard [2] described in the previous chapter, we can divide each asset into multiple procedures where each procedure contains its properties. By doing this we are following the Spectrum standard as well and making the information associated to each procedure independent from each other.

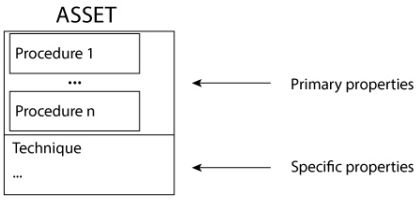

Figure 3: Spectrum procedure oriented data structure

Due to the limited timespan of this work, only a few key properties will be implemented. The chosen properties will be associated to the following groups and will provide IST with the minimum necessary information for cataloguing it's current asset collection:

- Asset Identification
- Collection

- Asset Description
- Asset Location
- Asset History

Each one of these groups will include the minimum necessary information to identify an asset. These groups can be easily expanded to include more features as described in the full document of this work.

A possible solution for implementing the previously described data structure is JSON. JSON is a data format that allows for nesting variables, objects and arrays within each other, creating a simple to implement and read structure such as the tree associated to the specific properties. Another advantage of using JSON is the incorporation of this data format to other languages such as Javascript where a JSON element directly maps to a Javascript object [3]. In listing 1 an example of a possible asset JSON data structure is shown.

Listing 1: Example of an asset's JSON data structure stored in the database.

```
{
    "_id" :
    ObjectId("5d4d734bd1fa0"),
    "ObjectIdentification" : {
        "title" :
        "Quadro da nascente",
        "optionalIds" : [
            {
                "N de inventorio B" :
                "435"
            }
        ]
    },
    "ObjectDescription" : {
        "category" : [
            "Pinturas",
            "Gravuras"
        ]
    },
    "pinturas" : {
        "author" : "Eduardo Lopes",
        "year" : "2007",
        "material" : [
            "Madeira"
        ],
        "suporte" : "Madeira",
        "tecnica" : "Aguarela"
    },
    "gravuras" : {
        "amountOfCopies" : "252",
        "copyNumber" : "8"
    }
}
```

Specific modules can be associated to an asset depending on its category. In the previous listed example, we can observe that the data structure contains 2 objects which are conditionally inserted to the asset upon insertion: "pinturas" and "gravuras". The properties exist in the data structure since in the "ObjectDescription" both these properties are included. In this specific case "gravuras" is a sub-category of "pinturas", meaning that for an asset to have the property "gravuras" it must also have the property "pinturas".

Data Persistence

Now that the data structure is defined, we must ensure its persistance. This means that our data will survive even after the process that created it ends or if the system shuts down, to do so, a database is needed.

When thinking of a database choice in computer engineering the most popular option is using an SQL based system such as MySQL. A MySQL database is made up of multiple tables, with each table having a specific pre-defined data structure schema. In our museum example this means that we could have, for example, a table which contained all information of assets that were painting such as the author, technique, and so on. Each table then has rows, being that each row corresponds to a single item such as an asset.

When looking into other database systems, another solution is using a document based database system such as MongoDB. In MongoDB, documents are grouped into collections, meaning that the database might have a collection for the assets, one for the users, and so on. When relating to MySQL, we can look at collections as being SQL tables and documents as being table rows. Documents follow a dynamic JSON structure, in this structure we can nest properties inside other properties [4]. When comparing to the previously described SQL structure, in MongoDB collections do not have an associated data schema, meaning that they may contain assets with different properties. This solves the previously presented problems of using an SQL based system for managing museum assets.

In addition to this, the data model introduced in the previous section can be easily implemented by subdividing the content of each asset's documents into multiple independent modules.

Back-end

Having defined the data structure as well as the database the next step is to define how the necessary computing power will be implemented in order to perform operations on the database. Hav-

ing users directly interacting with the database may bring problems such as data duplication or insertion of invalid data structures, which will compromise the whole system.

In order to prevent such problems a back-end infrastructure is defined. In this case, the back-end infrastructure will consist of the previously mentioned MongoDB database as well as a machine running processes capable of performing the standard database operations as well as ensuring the system's integrity.

An example such solution is using Node.js: an open-source, cross-platform JavaScript run-time environment that executes Javascript code outside of a browser in order to enable server-side scripting [5]. Since JSON and MongoDB documents perfectly map into Javascript objects, performing the necessary data conversions will be a seamless process. Throughout the rest of the work, this Node.js environment will be refered to as the back-end.

The back-end is based on routes which are identified by a unique URL according to the REST methodology [6]. This means that in order for a user to interact with the system a request is sent, according to the REST API to the corresponding back-end route. So for example, in order to get the list of users in the database we will perform an HTTP GET request to a route similar to this one:

GET http://www.myServer.com/api/users

Our back-end server will receive this request, access the database, insert all of the users into a javascript array and send that array to the front-end.

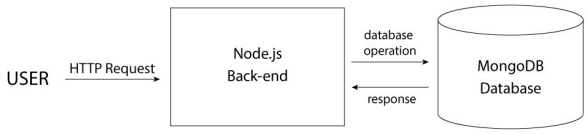

Figure 4: User performing a request to the back-end.

This is a very basic example which does not include authentication. Security and user authentication will be discussed further on.

Front-end

Performing an HTTP request to a web server requires some computer networking knowledge, leaving the system as it is now would only allow a reduced number of people to use it. It is possible to abstract all of the networking logic into something such as a browser interface with interactive buttons and data visualizations. This is called the front-end.

The front-end corresponds to the user interface with which the user will interact without the need of knowing any prior knowledge about programming. This is a case of code on demand, in order for the user to be able to use the platform we will simply need to open a web browser and type in the URL corresponding to the server directory in which the front-end files are stored. By doing this, the browser will send the necessary HTTP request to the server in order to get the interface code and will display it on the browser for the user to interact with.

In order to implement the front-end a framework called Vue.js (or simply Vue) can be used. Vue is a web development framework created by Evan You in 2013 after having worked for Google using the framework Angular.js in various projects. This framework allows us to separate the various parts of the platform into components. A component is a file which can contain HTML, CSS and Javascript code. We use components in order to better structure our code, it would be a mess if we had all the elements of our platform into the same HTML, CSS and Javascript files. Having the possibility of dividing the platform into components will prove to be very important in order to create a scalable code base where each element works independently from each other [7].

The code structure used for our front-end client is quite similar to the one seen in most Vue projects, the big difference is the way in which we use modules to create a truly dynamic and scalable platform. We define views that serve as the containers for other components but you can also look at views as if they correspond to each of our platform's different pages.

The big challenge with storing museological assets lies in their massive diversity of properties. It makes sense to specify what type of motor a car has but it does not make sense to specify what type of a motor a painting has. This is where our modules come in.

The main operations of the platform are the following:

- Insert - Add a new asset to the database.
- Display - Observe an asset's details.
- Edit - Change an asset's details.
- Search - Search for assets with certain caracteristics.
- Config - Configure the values that a user can select for a specific module.

By defining these key operations we can now separate our assets' components into different modules. We have, for example, the object identification module which includes an asset's title and optional ids. For this module we will have insert, display, edit, search and config files.

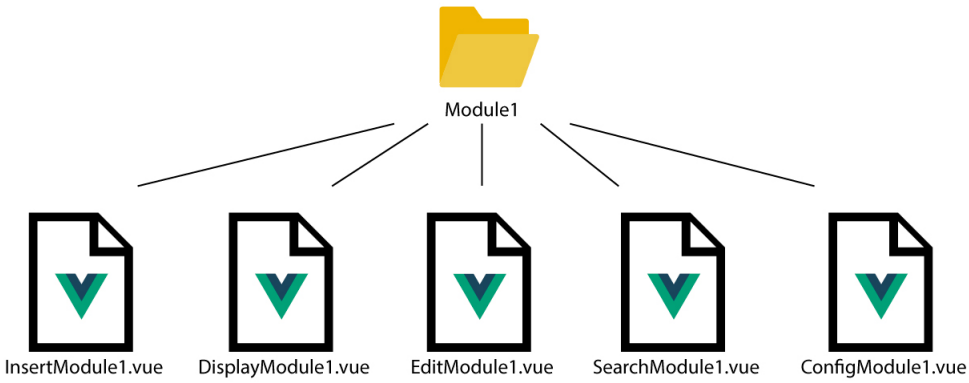

Figure 5: Module structure in front-end.

This means that when displaying an asset we will render in our web page all of the modules relevant to the asset, thus creating a single information page that results from many Vue components. We can now also decide which modules we show to the user depending for each case. When displaying an asset we will view its categories and render all of modules corresponding to those categories if they exist, this is done dynamically in Vue by conditionally rendering Vue components.

In order to further illustrate the relation between a module and other components we can observe figure 6.

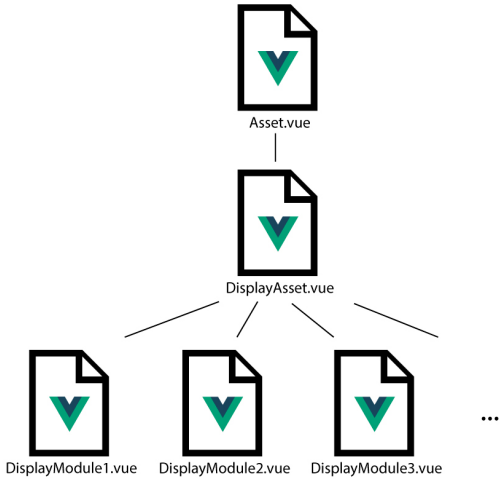

Figure 6: Example of how display modules are connected to parent components.

The front-end is tightly coupled to the back-end, meaning that it won't function properly without a running back-end server. Data that we view on the interface, such as the name of the assets, has to be fetched from the database. In order to get such data we perform HTTP requests to the back-end server. But using simple HTTP request is not enough since we want to load the data on the page dynamically and not have to refresh the page each time we make a request. For that we use AJAX in order to perform asynchronous requests to the server. An asynchronous operation allows us to perform other tasks while waiting for this operation to complete [8]. After fetching the data, we can then process it and display it in our Vue component. Our system architecture will now resemble the one in figure 7.

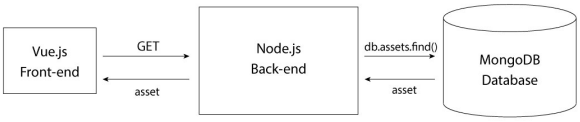

Figure 7: Example of a client request for an asset.

Security and Authentication

Security is of key importance to our web platform since we have multiple users acessing common data on the database. We need to have a way of defining specific permissions for each user in order to control what they can and cannot do. We start by defining 3 types of possible user profiles:

- Viewers - Can only view the assets in the database, they cannot change or delete anything.
- Editors - Can insert, edit and delete assets from the categories they have been assigned.
- Admins - Can do everything. This includes giving permissions to other users, configuring modules' data and inserting, editing and deleting any assets from the database.

To implement this security system we use JWT as well as the FenixEdu authentication system [9] since we only want to allow users from IST to use our application for now.

JWT is used for passing tokens between machines in order to identify and validate who is sending the request [10]. In our platform, the front-end will append a token to requests that require one. For example, editing is a higher permission feature which viewers must be unable to use. To make sure the person editing the asset can indeed do it we append a token that confirms the person's identity. Most requests such as a search query do not require the front-end to append a token to the message since every type of user can perform this action. Once the back-end receives a request, it will decode the token in order to identify the person performing the action. For this, it will need to know the secret with which the token was encrypted with, but that is no problem since the only entity that generates tokens is the back-end itself.

This is the authentication process:

1. The front-end redirects the user to the FenixEdu login page.
2. If the login is successfull, FenixEdu will redirect the user back to the front-end and append a code to the URL.
3. The front-end sends a POST request to FenixEdu with the received code.
4. FenixEdu responds with a token corresponding to the user's session.
5. The front-end sends that token to the back-end
6. The back-end gets the user's basic information from FenixEdu.
7. The back-end checks if the user is already in the database, if not, a new user is created. The user university id is used as the user identifier.
8. The back-end generates a jwt token and sends it as a response to the front-end.
9. The front-end stores the jwt token locally and appends it to future requests when needed.
10. Each time the back-end receives a jwt token, it decodes it and views if the user has the required permissions before continuing.

The front-end checks, each time before we change pages, if the token is already in the front-end, if it is, the user will continue to the desired page. If not, the authentication process that can be seen in figure 8 will be triggered in order to authenticate the current user.

Category Matching

In order to implement the dynamic modules previously described we require a way of identifying

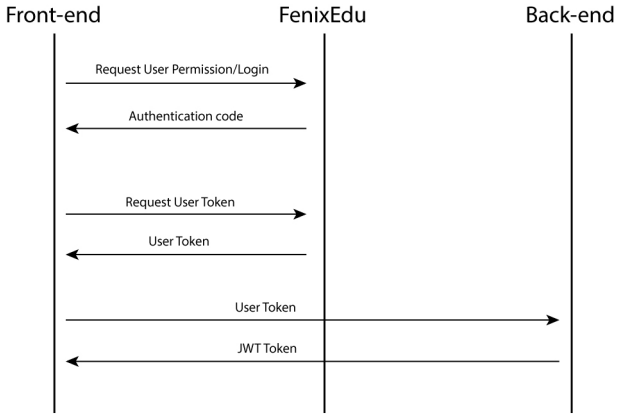

Figure 8: User authentication process.

the categories of the asset. This information is included in the asset's JSON, more specifically in the Object Description object. In this object is included the information that associates the asset to certain categories in order to then allow for other dynamic modules to work. These categories follow a tree structure. This allows for the user to specify, within a category, the asset's sub categories in order to further identify the asset. An example of a category tree is shown in figure 9. In this example we can set an asset's category as being a painting and then drill down even further and state, for example, that it is also of the category acrylic.

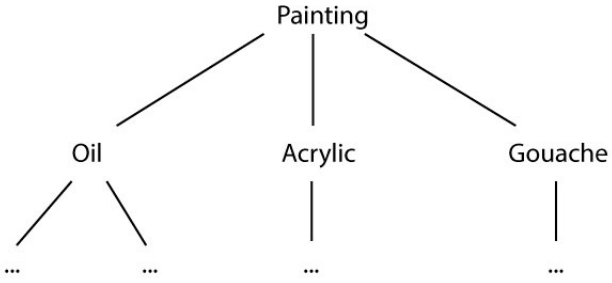

Figure 9: Example data structure of a category.

Web Platform

The work developed is based on a web architecture. This implies communication between machines, more concretely between: the front-end, back-end and database. Another option for this platform would be using other technologies to develop a solution with all of these 3 elements working locally in the same machine. Having a local solution would have its benefits when it comes to, for example, security since we would know exactly who could access the platform. The major drawback would be that the platform could only be accessed

from a specific machine and only the people with access to that machine would be able to interact with our platform.

Right from the beginning it was known that this platform would have to be very convenient to use in order for museum collectors to make use of it. That is why the platform is based on a web architecture. This allows for collectors to manage their collections from any device and any place as long as they have an internet connection. Another advantage of having a web platform is that we can give the collectors the option of allowing the general public to view their collections without the ability to perform the same actions as the collection managers. But having these benefits comes with a price, and that is the need for more complex security systems in order to avoid that people who do not have permissions to edit the collections do so.

Another advantage of using a web platforms with these specific technologies is the ease of developing new features since javascript is currently one of the most popular programming languages amongst developers as well as being a higher-level programming language when comparing to others such as C or C++ in which the developers have to deal with concepts such as memory allocation and pointers which are abstracted in javascript.

When it comes to the choice of technologies for developing our web platform solution another choice of programming language would be using Python with, for example, the framework Django for the front-end and the framework Flask for the back-end [11]. This would wield a very similar result, despite this observation, javascript was chosen for its popularity when it comes to web development since anyone with minimum knowledge of javascript would be able to quickly learn and implement small features in the current platform.

4. System Validation

In order to achieve the short-term objective of this work of cataloguing the current assets at IST we will now look into how the current existing asset repositories can be included into the developed system.

These are the properties developed during the timeframe of this work:

- Asset Identification
  - Title
  - Inventory numbers
- Collection
  - Collection name
- Asset Description
  - Category name
- Asset Location
  - IST location (current and usual)
  - Coordinates (current and usual)
  - Address (current and usual)
- Asset History
  - Event name
  - Event description
  - Event date

IST general inventory sheet

When looking into assets that follow the structure of a common IST physical asset information sheet presented in this work's full document, it is possible to create a direct mapping between some of the properties. These are the properties lost when inserting the asset into the platform:

- Description
- Brand
- Date
- Image id
- Dimensions
- Material
- Price evaluation

All of these properties can be stored within the platform by extending it with the implementations of new modules.

Paintings sheets

When looking into assets that follow the structure of a painting asset sheet at IST presented in this work's full document, it is also possible to establish a mapping between the properties of the inventory sheet and the platform. In this case all of the properties can be included into the platform since specific modules were developed for these categories of asset, more specifically the modules "Pintura" and "Gravura".

This is a good example of how, by extending the platform, as seen in the previous chapters, we are able to store all information relative to any type of asset category.

Museum of geology and mineralogy

As previously mentioned, the museum of geology and mineralogy has a Microsoft Access with information of some of its assets. These assets have properties divided into 3 groups:

- Identification
- Characteristics
- History

In the Identification group all properties are covered except the asset brand. In the Characteristics and History groups most properties are not covered by the platform, in order to incorporate them the platform needs to be extended with new modules similarly to what was done with the paintings.

5. Conclusions

Extensibility

In order to evaluate the extensibility of the platform, the code base was provided to a developer in order for him to develop a new module for the system. The developed module corresponds to the historical events associated to an asset where the user is capable of inserting the start and end time of the event, the event name and the event description. This means that if, for example, there happens to be an asset that was in exhibition at some specific place and time the platform user is capable of recording this information.

The developer assigned to developing this new module had never seen the platform before and was just given the technical guide in chapter 3 of this work's full document. The developer, who is called Artur, was at the time of this work a computer science student in his first year of his masters degree at IST with some experience in Vue, Node.js and Javascript.

In these conditions, Artur was capable of fully developing this new module in 1 hour and 38 minutes. The module includes insertion, display, editing and search.

Given the short amount of time required for a developer with absolutely no knowledge of the platform to develop a new module, we can conclude that the platform allows for fast and easy development of new components.

Extensibility Automation

Being this a platform that provides developers with a straightforward and repeatable guide on how to implement new modules, it should be possible to develop software to, given the new module's description, create new components automatically without requiring any coding. This was not implemented in this work due to the limited time frame but by implementing a script that will follow the technical guidelines given in chapter 3, the museums owner is able to adapt the platform to its specific needs without the need of any coding or knowledge on how the platform is implemented.

Final Thoughts

This thesis had two main objectives, one is for the short-term and the other for the long-term.

The short-term objective was to provide an immediate solution for managing the assets at IST. To achieve this, a web platform was developed with the features needed to properly identify different assets from different collections as well as other important information associated to them such as their current location. The platform provides essential operations including: insertion of an asset, visualization of an asset's information, capability of editing an asset's information upon insertion, ability to search within the platform in order to find specific assets and configuring what values can be selected from specific asset properties upon asset insertion, editing and searching.

The long-term objective was to provide the foundations for an extensible asset management system so that anyone with minimal programming knowledge can develop further components for it in order to fit any necessary requirements for other museums. To achieve this, a web platform capable of managing assets was developed, the same one described in the short-term objective. The architecture of this platform was developed in a way that allows for an ease of extensibility by modularizing its components so that new developments remain independent from previously defined ones. To add to this architecture, a step by step technical process on how to develop new modules for the platform was written in order to allow developers with little knowledge of Javascript to develop these new modules.

The initial study also raised attention to the importance of the use of proper asset management systems for museums in order to maintain their collections visible and accessible for people to see. In an institution such as IST where the assets are not catalogued neither publicly available, having a digital system to manage the assets might lead to a better internal knowledge of what assets exists as well as facilitate the public sharing of information regarding its collections and exhibitions.

The developed platform is far from being a finished product, but it is the foundation where other developers can create more functionalities and features in order to possibly create multiple solutions for multiple different museums.

References

[1] Miriam Rute De Jesus Barros. *Materialidade da técnica: um diagnóstico do acervo museológico do IST*. PhD thesis, 2007.

[2] CollectionsTrust. *The UK Museum Collections Management Standard Cataloguing procedure*. 2009. URL https://collectionstrust.org.uk/wp-content/uploads/2016/11/spectrum_4_03.pdf.

[3] Ray Rischpater. *JavaScript JSON Cookbook*. 2015. ISBN 9788578110796. doi: 10.1017/CBO9781107415324.004. URL https://pepa.holla.cz/wp-content/uploads/2015/11/JavaScript-JSON-Cookbook.pdf.

[4] MongoDB. MongoDB Architecture Guide. 2016. URL https://jira.mongodb.org/secure/attachment/112939/MongoDB_Architecture_Guide.pdf.

[5] Marklogic Corporation. *Node.js Application Developer's Guide*. Number May. 2019. URL https://docs.marklogic.com/guide/node-dev.pdf.

[6] Joe Gregorio. Intro to REST. URL https://bitworking.org/images/2009/01/intro-to-rest.pdf.

[7] Flavio Copes. *Vue Handbook*. 2018.

[8] Cornelia Györödi, Robert Györödi, George Pecherle, Tamas Lorand, and Rosu Alin. Web 2.0 technologies with jQuery and Ajax. *Computer Technology and Computer Programming: New Research and Strategies*, pages 99–110, 2016. URL https://www.researchgate.net/publication/40422422_Web_20_Technologies_with_jQuery_and_Ajax.

[9] Fernando Mira da Silva. *The FenixEdu project*. 2002. URL https://ciist.ist.utl.pt/projectos/Fenix.pdf.

[10] Krishna Shingala. *JSON Web Token (JWT) based client authentication in Message Queuing Telemetry Transport (MQTT)*. PhD thesis, 2019. URL http://arxiv.org/abs/1903.02895.

[11] Hans Petter Langtangen and Anders E Johansen. *Using Web Frameworks for Scientific Applications*. PhD thesis, 2015. URL https://hplgit.github.io/web4sciapps/doc/pub/web4sa.pdf.