



UNIVERSIDADE D  
COIMBRA

Pedro Miguel da Cruz Santos

**MOSAIC EDITOR AND SHARING PLATFORM**  
BUILDING OF AN INTERACTIVE PLATFORM

Dissertation in the context of the Master's in Informatics Engineering, specialization in Software Engineering, advised by Professor Jorge C. S. Cardoso and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July of 2022





FACULDADE DE  
CIÊNCIAS E TECNOLOGIA  
UNIVERSIDADE DE  
**COIMBRA**

DEPARTMENT OF INFORMATICS ENGINEERING

Pedro Miguel da Cruz Santos

# Mosaic Editor and Sharing Platform

Building of an interactive platform

Dissertation in the context of the Master in Informatics Engineering,  
specialization in Software Engineering, advised by Prof. Jorge C. S. Cardoso and  
presented to the Department of Informatics Engineering of the Faculty of  
Sciences and Technology of the University of Coimbra.

July 2022



## **Acknowledgements**

I would like to thank my supervisor for the availability and open mindedness during this project. Without their help, the current state of this document and the project wouldn't have been possible.

I would also like to thank my family and friends for their continuous support and always believing in what I can accomplish.



## **Abstract**

MosaicoLab is an organization that actively seeks to bring attention to the cultural significance of mosaics. To better achieve this goal, several workshops are held each year to bring attention to this subject. However, during the pandemic context the world has been through, it is difficult to keep organizing these workshops. Therefore, a digital approach was taken that led to the creation of the mosaic editor. However, this tool was not perfect on its own, several tools could be added to the editor and the process of saving mosaics to showcase them was not intuitive.

Given these difficulties, this project was proposed. In this project, the objective was to create a Mosaic Sharing Platform where people can design and paint their mosaics as well as create mosaics from scratch or start from a previously built mosaic. Furthermore, several tools have been added to an existing mosaic editor so that users can easily perform actions that would take them much longer had these tools not been present.

During this project, several platforms were analyzed to better understand what can be expected from a sharing platform and how certain drawing tools work. Having the analysis performed, the requirements can be proposed, after which the architecture design and implementation phases took place.

During the implementation phase, the project was divided into two separate smaller projects. One of them is the creation of the Mosaic Sharing Platform, consisting in implementing all the features needed for users to easily choose mosaics to re-mix as well as publish their own mosaics for other users to re-mix from. The other consists of adding functionalities to an existing mosaic editor. This second part directly relates to the Mosaic Sharing Platform since the mosaics present in that platform are created in the mosaic editor.

In order to understand how well were the requirements accomplished, an evaluation of the work performed during this year will take place, analyzing not only the functional requirements but also the quality attributes.

With this project, not only can mosaics reach further in the sense of people that will have access, but also in the sense that through mutual collaboration, several new ideas may come up that will allow a spur of mosaic enthusiasts and new creative patterns to be used in the mosaicing community. Therefore, driving the importance of mosaics as a landmark further.

## **Keywords**

Tile, Mosaic, Re-Mixing, Sharing, Platform, JavaScript, PHP





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	MosaicoLab . . . . .	2
1.2	Motivation . . . . .	2
1.3	Scope . . . . .	3
1.4	Objectives . . . . .	5
1.5	Structure of this document . . . . .	6
<b>2</b>	<b>Background – Mosaic Editor</b>	<b>9</b>
2.1	Functionalities . . . . .	9
2.2	Software Architecture . . . . .	10
2.3	Code base . . . . .	15
<b>3</b>	<b>State of the Art</b>	<b>19</b>
3.1	Mosaic Editors . . . . .	19
3.1.1	Macropolis . . . . .	19
3.1.2	Learning Playground . . . . .	21
3.1.3	TileMaker . . . . .	22
3.1.4	Possible Features . . . . .	25
3.2	Drawing Software Applications . . . . .	25
3.2.1	Selection tools . . . . .	26
3.2.2	Painting tools . . . . .	27
3.2.3	Drawing tools . . . . .	29
3.2.4	File operations . . . . .	30
3.2.5	Image manipulation . . . . .	31
3.2.6	Possible Features . . . . .	32
3.3	Re-mixing platforms . . . . .	32
3.3.1	Scratch . . . . .	32
3.3.2	Snap! . . . . .	34
3.3.3	Glitch . . . . .	36
3.3.4	Possible Features . . . . .	38
<b>4</b>	<b>Methodology and Work Plan</b>	<b>41</b>
4.1	Methodology . . . . .	41
4.1.1	Scrum . . . . .	41
4.1.2	Waterfall . . . . .	42
4.1.3	Kanban . . . . .	44
4.1.4	Reflection . . . . .	45
4.1.5	Applying this methodology . . . . .	45
4.2	Work Plan . . . . .	46

4.2.1	First Semester . . . . .	46
4.2.2	Second Semester . . . . .	49
4.2.3	Achieved Planning . . . . .	50
4.2.4	Risk Analysis . . . . .	52
4.2.5	Materialized Risks . . . . .	54
4.2.6	Threshold of Success . . . . .	55
<b>5</b>	<b>Elicit Requirements</b>	<b>57</b>
5.1	Mosaic Sharing Platform . . . . .	57
5.2	Requirements . . . . .	58
5.3	Backend . . . . .	61
5.4	The Mosaic Editor tool . . . . .	62
5.5	The Overall Platform . . . . .	63
5.5.1	Admin . . . . .	64
5.5.2	Curator . . . . .	64
5.5.3	User . . . . .	65
5.6	Quality attributes . . . . .	65
5.6.1	Usability . . . . .	65
5.6.2	Maintainability . . . . .	66
<b>6</b>	<b>Architecture</b>	<b>69</b>
6.1	General Architecture . . . . .	69
6.2	Mosaic Sharing Platform . . . . .	70
6.2.1	Technologies . . . . .	70
6.2.2	Database Architecture . . . . .	72
6.2.3	User Operations . . . . .	74
6.3	Mosaic Editor . . . . .	75
6.3.1	Communication with the Mosaic Sharing Platform . . . . .	77
<b>7</b>	<b>Implementation</b>	<b>79</b>
7.1	Development Environment . . . . .	79
7.2	Mosaic Sharing Platform . . . . .	79
7.2.1	Project Structure . . . . .	80
7.2.2	Communication with the database . . . . .	83
7.2.3	Managing files from the server . . . . .	85
7.2.4	Platform behaviour . . . . .	88
7.3	Mosaic Editor . . . . .	93
7.3.1	Communication with the Mosaic Sharing Platform . . . . .	93
7.3.2	Added Functionalities . . . . .	95
<b>8</b>	<b>Evaluation</b>	<b>103</b>
8.1	Functional Tests . . . . .	103
8.2	Usability testing . . . . .	104
8.2.1	Procedure . . . . .	105
8.2.2	Results . . . . .	106
8.2.3	Discussion . . . . .	114
8.3	Maintainability . . . . .	115
8.3.1	Mosaic Sharing Platform . . . . .	115
8.3.2	Mosaic Editor . . . . .	115

8.4	Accomplished requirements . . . . .	116
<b>9</b>	<b>Conclusion</b>	<b>119</b>
9.1	Work Done . . . . .	119
9.2	Challenges . . . . .	120
9.3	Suggestions . . . . .	121
9.4	Final Remarks . . . . .	121
<b>Appendix A</b>	<b>Mosaic Editor Software Architecture</b>	<b>127</b>



# Acronyms

**ACID** Atomicity, Consistency, Isolation and Durability.

**API** Application Programming Interface.

**CRUD** Create, read, update and delete.

**CSS** Cascading Style Sheets.

**DJ** Disk Jokeys.

**HTML** HyperText Markup Language.

**IDE** Integrated Development Environment.

**JSON** JavaScript Object Notation.

**OO** Object Oriented.

**PHP** PHP: Hypertext Preprocessor.

**PNG** Portable Network Graphics.

**SQL** Structured Query Language.

**SVG** Scalable Vector Graphics.

**TIFF** Tagged Image File Format.

**URL** Uniform Resource Locator.

**XML** Extensible Markup Language.

**ZIP** Zip Format.



# List of Figures

1.1	Interface of MosaicoLab’s mosaic editor . . . . .	3
1.2	Structure of the currently implemented editor . . . . .	4
2.1	Class diagram representing the MosaicEditor class . . . . .	10
2.2	Class diagram representing the Cmd class . . . . .	11
2.3	Class diagram representing the TilePainter and associated classes .	12
2.4	Class diagram representing the GestureDetector class . . . . .	13
2.5	Class diagram representing the MosaicLayout and the MosaicView classes . . . . .	14
2.6	Class diagram representing the Mosaic and Tile classes . . . . .	15
2.7	Class diagram representing the Observer class . . . . .	15
2.8	Diagram representing the Observer behavior . . . . .	16
2.9	Structure of the classes on the existing project . . . . .	16
3.1	Interface of the Macropolis Mosaic Editor tool . . . . .	20
3.2	Macropolis editor when picking the chevron pattern . . . . .	20
3.3	Interface of the Learning Playground Mosaic Editor tool . . . . .	21
3.4	Interface of the TileMaker editor tool . . . . .	23
3.5	Rectangular selection . . . . .	26
3.6	Free hand area selection . . . . .	27
3.7	Color selection tool . . . . .	27
3.8	Fill-in tool icons for Paint, Gimp and Photoshop . . . . .	28
3.9	Smudge tool icon . . . . .	29
3.10	Draw path functionality icon . . . . .	30
3.11	Layer functionality icon . . . . .	30
3.12	Color Picker functionality icon . . . . .	30
3.13	Interface of Scratch platform . . . . .	33
3.14	Scratch’s creating interface . . . . .	34
3.15	Snap’s main page . . . . .	35
3.16	Glitch’s Discover page . . . . .	37
3.17	Featured Projects on Glitch . . . . .	37
4.1	Waterfall life cycle . . . . .	43
4.2	Life cycle of the project using the waterfall methodology . . . . .	46
4.3	Gantt diagram of this project . . . . .	48
4.4	Work Plan for the 2nd Semester . . . . .	49
4.5	New planning for the second semester . . . . .	51
4.6	Risk Matrix of the risks associated with the project . . . . .	52

6.1	Context Diagram representing the basic functionality of the platform . . . . .	70
6.2	Database model for the new platform . . . . .	72
6.3	Container Diagram with the different stacks for user operations . .	74
6.4	Class diagram representing the Tool class . . . . .	76
6.5	Activity Diagram representing the flow of information and the user tasks when using the platform to draw mosaics. . . . .	77
7.1	Project Structure of the Mosaic Sharing Platform . . . . .	80
7.2	Language files available for the platform . . . . .	81
7.3	Comparison between an excerpt of the English language file and the Portuguese language file . . . . .	82
7.4	Folder that contains all the Mosaics and their project and image files	82
7.5	Admin Navbar . . . . .	90
7.6	Not logged in user navbar . . . . .	90
7.7	Regular navbar . . . . .	90
7.8	Admin collection info page, for regular users, the add mosaic button is locked as well as the edit collection. For curators, the add mosaic button is visible but not the edit collection. . . . .	90
7.9	Logic behind how a user can save a mosaic depending on the mosaic and user information . . . . .	94
7.10	Activity Diagram for the Chain of Responsibility applied to the mosaic editor . . . . .	96
7.11	Button used to initiate the draw shape operation . . . . .	97
7.12	Button used to initiate the switch color operation . . . . .	97
7.13	Button used to initiate the area selection operation . . . . .	98
7.14	Mosaic without a predefined layout where the first technique used to paint the tiles doesn't work . . . . .	99
7.15	Button used to initiate the fill in operation . . . . .	100
7.16	Button used to toggle the guidelines . . . . .	101
8.1	Mosaic on the left is the mosaic to be remixed into the mosaic on the right . . . . .	105
8.2	Collection containing all the house mosaics created from the usability testing . . . . .	107
8.3	Gender of participants in the usability testing . . . . .	107
8.4	Ages of participants in the usability testing . . . . .	108
8.5	Answer to the first question of the form regarding the project . . .	108
8.6	Answer to the second question of the form regarding the project . .	109
8.7	Answer to the third question of the form regarding the project . . .	109
8.8	Answer to the fourth question of the form regarding the project . . .	110
8.9	Answer to the fifth question of the form regarding the project . . .	110
8.10	Answer to the sixth question of the form regarding the project . . .	111
8.11	Answer to the seventh question of the form regarding the project .	111
8.12	Answer to the eighth question of the form regarding the project . .	112
8.13	Answer to the ninth question of the form regarding the project . . .	112
8.14	Answer to the tenth question of the form regarding the project . . .	113
A.1	Mosaic editor class diagram . . . . .	128



# List of Tables

5.1	MoSCoW table for priority of different requirements . . . . .	59
5.2	Usability scenario . . . . .	66
8.1	Requirements accomplished from the already existing requirements table . . . . .	117



# Chapter 1

## Introduction

Roman mosaics were a form of art in ancient Rome that was used in walls, ceilings, and floors mostly to represent gods, quotidian events, and natural elements. The richest of the society would have the privilege of possessing mosaics in their homes while the working class could enjoy them on their walkways, and commerce areas [Stephan, 2016]. The tradition of mosaics was spread throughout the entire Roman empire, including Lusitania which is now Portugal. The tradition of mosaicing did not stop with the Romans however, we made it our own and built the famous Calçada Portuguesa with patterns that were introduced from roman mosaics [Balmelle et al., 2018]. Portugal has a great number of Roman mosaics in Conimbriga – one of the largest Roman settlements excavated in Portugal.

This heritage is very important to the Portuguese people and carries a lot of cultural weight. The tradition of Roman mosaics dates back to the 5th century BC. Since it is an old tradition it's natural that as the times change our focus is directed into other areas. To keep-sake the Roman Mosaics tradition, but also to promote modern creative work inspired by mosaicing, the MosaicoLab Association [Figueiredo, 2020] (<https://mosaicolab.pt>) hosts events (workshops) in Conimbriga and elsewhere with the intent of letting the participants get a feel for what this activity has to offer and how it is done, as well as shining a light on the culture behind mosaics.

To help mosaicing keep up with the digital age, MosaicoLab has developed a digital mosaic editor. This allows the participants of these events to design their mosaics and experiment with different patterns, colors, sizes, etc. before setting them on stone. Given the pandemic situation of these recent years, the mosaic editor has been used also in digital mosaic creation workshops, as the central tool for mosaic creation. Usage of this tool has revealed some shortcomings that prevent it from being used to an even wider extent. Functionalities such as being able to store and share digital mosaics on a server, or adding more advanced editing tools, are examples of things that would make the tool more relevant for MosaicoLab.

The objective of this work is, thus, to expand the mosaic editor's capabilities. By implementing backend functions that allow the registration and logging in of users, as well as saving and sharing mosaics online. The final tool is expected to

allow users to work on mosaic designs on their own or over a design that was already built by another user. We also want to implement more advanced editing functionalities such as using tools that allow the user to draw certain shapes on the mosaic easily. With these new functionalities, we can expect a growth of activity in the platform since the tool will become more useful.

## 1.1 MosaicoLab

The client of this project is MosaicoLab, an organization that seeks to promote creative tourism and build new job offers through the cultural weight that mosaics carry. With that objective in mind, some steps need to be taken in that direction, namely allowing different people from different backgrounds to explore and learn about mosaics. This way the relevance of mosaics will become more noticeable and allow for the objective to be near completion. Some of the projects that are done to achieve this are the "Festa do Mosaico" events which are held every year as an event that allows participants to develop their own mosaic. However, through the pandemic, this events could no longer be held due to the safety measures applied by the Portuguese government. This led to the creation of a digital platform that allows users to draw their own mosaics. In the year 2021, the event was held once again, this time, using the newly created platform.

## 1.2 Motivation

Throughout the pandemic, MosaicoLab's team kept up with their workshops to bring attention to the existence of mosaics and to incentivize young people to experiment with mosaicing. However, during these times, it is very difficult to gather people together and have face-to-face experiences, therefore, they resorted to virtual events.

Trying to still convey the excitement of creating their mosaic, they used the digital tool that they had developed to draw mosaics and, as is tradition, in the end, the digital works that the participants developed get showcased in the MosaicoLab's website. With the current implementation of this platform, this process was very difficult, in particular the showcasing of the mosaics each participant designed, since they needed to download the image and send it to the staff members in charge of the workshop. The design process was not perfect either, sometimes the participants might have wanted to fill in a whole region of the same color or draw a shape and since the only possible way of doing so currently is by filling each tile one by one, a lot of otherwise unnecessary work had to be done.

Therefore, the need for an improved tool arose and this dissertation came to be.

Since mosaicing is a big part of Portuguese inheritance and history I found this project to be of bigger importance than other people might have thought. The currently implemented tool can be improved by adding some features. Such features could enable us to bring other users into the mosaic community and could

get some more engagement between user and tool and users between themselves. With the project at hand, the goal is to solve some of these issues and develop a tool that is easy to use and offers collaboration possibilities. The more people join the platform the safer we can feel about the maintenance of our heritage and the progression of the art of mosaicing.

## 1.3 Scope

This project consists of two parts: the frontend, which is the client-side part (UI) of the mosaic platform to be developed; and the backend, which is the server-side part that provides services such as storing the digital mosaic creations, managing users, etc.

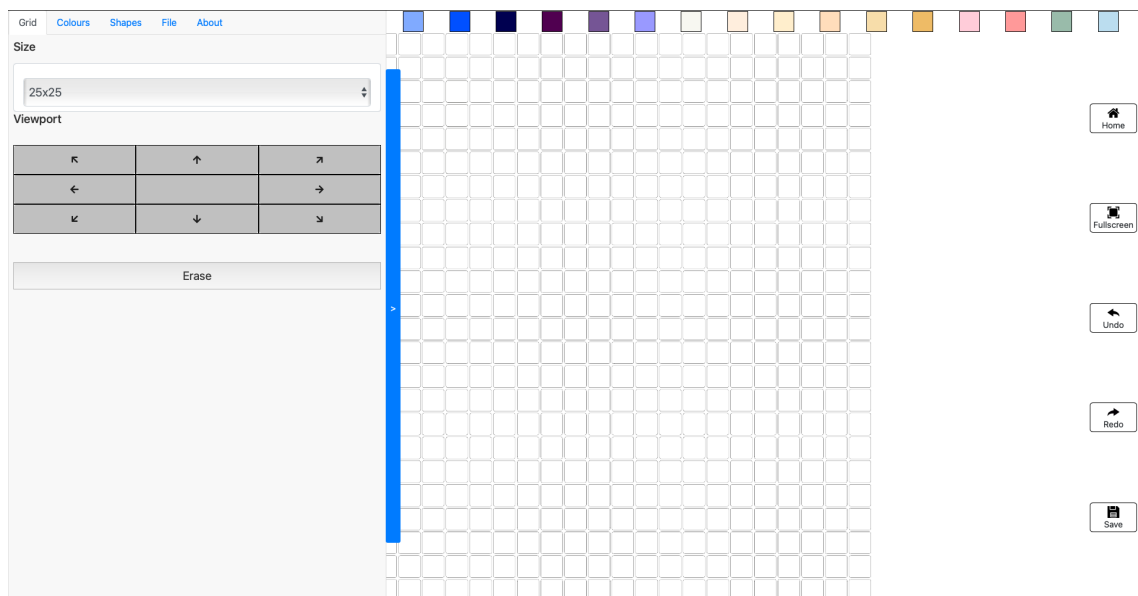


Figure 1.1: Interface of MosaicoLab's mosaic editor

The client-side already exists. As seen in figure 1.1 the interface where the user can produce mosaic is already implemented and working on MosaicoLabs' server (<https://mosaicolab.pt/editor>). The frontend part of this project is where the implementation of other mosaic drawing features will take place. These features can be drawing shapes by clicking a button and then dragging the mouse, area selection and guidelines. These features mainly deal with the editor itself and what the user sees and interacts with while designing their mosaic.

The underlying software for the existing client-side mosaic editor is currently structured in a way that tries to take advantage of well-known software design patterns. It is implemented in JavaScript and structured in an Object Oriented (OO) fashion. We can clearly understand what each class does, as seen in figure 1.2<sup>1</sup>, there are several classes that communicate with each other, allowing us to easily pinpoint where we'll have to work when implementing certain features.

<sup>1</sup>A bigger image can be seen in appendix A

All the functionalities to be added to this tool will be added into the existing code base and should, thus, also follow the design patterns that are already in use. They'll be implemented using JavaScript (as well as using HyperText Markup Language (HTML) and Cascading Style Sheets (CSS)).

The server-side does not exist yet, it will need to be developed from scratch. Here the definition and implementation of a database will occur as well as sharing mosaics between users, or publishing mosaics into the server. The development of the server-side will also need to consider the implementation of a new website that supports things such as the ability to register, login, manage users, etc., and also that provides access to the published listings of existing digital mosaics such as the ones created during specific workshops.

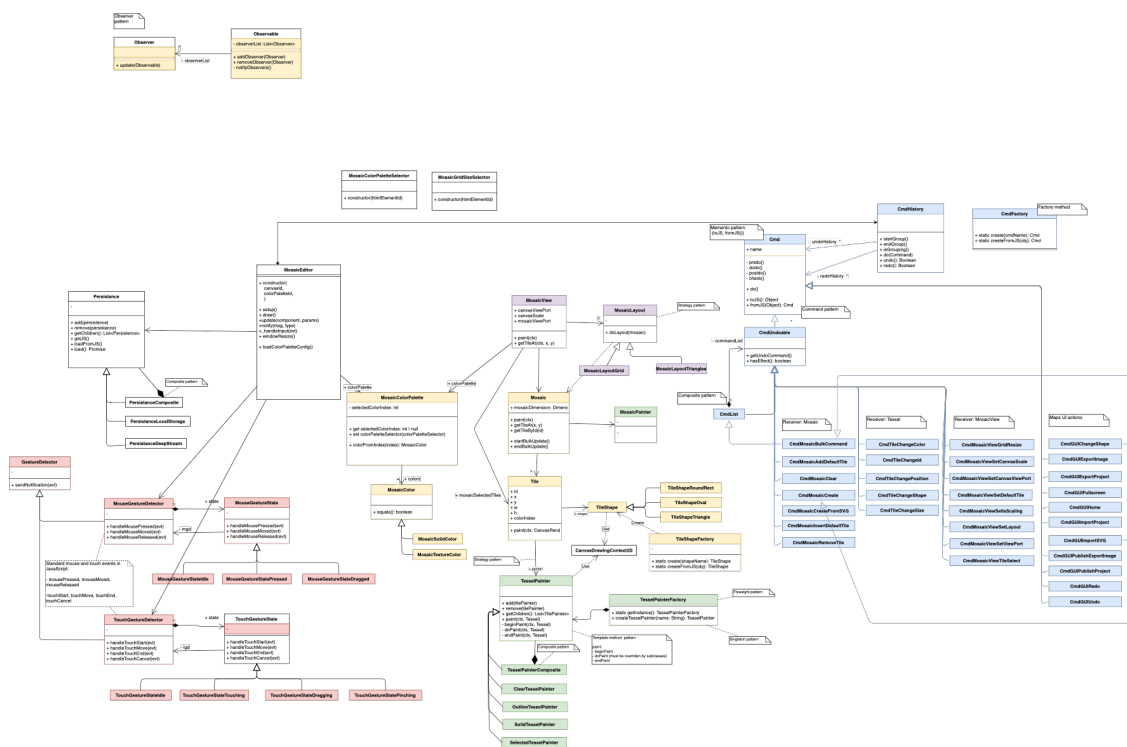


Figure 1.2: Structure of the currently implemented editor

For the time being, there are no backend capabilities to this platform. However, the fact that we can export project files from the platform already is a great advantage to being able to store those files in a server file system. From which we will be able to implement functionalities such as users working on projects that were started from different users. Since there aren't any server functionalities, this has to be done from scratch. As a client requirement, given the low-cost web service that the MosaicoLab has, implementation of the server-side should be done in PHP: Hypertext Preprocessor (PHP), without the use of complex frameworks that could run into dependency problems. The use of PHP becomes apparent when taking into consideration the simplicity and how it is platform independent. Furthermore, the process of connecting to databases is fast and easy, making it a good choice for the backend portion of the project. Even though several alternatives exist that provide great support and security, when taking into consideration the learning process, setting up the development environment and the dependencies,

PHP can overcome those issues quickly by being a simple language that doesn't need external software to compile.

## 1.4 Objectives

For this project, as mentioned previously, the objective is to further develop the currently implemented platform in a way that allows users to collaborate in the sense they would easily start drawing mosaics by remixing existing projects and share their mosaics so that other users can get inspiration or, start their mosaics from those projects. As well as easily draw their mosaics with new drawing tools. To achieve this, the current plan can be divided into two main categories, the backend server and the front end, interface, and editor. For the backend server, we'll need to implement a database and create registration and login forms as well as implement a safe way to save projects into a server file system. Furthermore, in this first category, we also have to develop the feature of allowing users to start their projects from other users' projects. Although the users can already do this by asking for the JavaScript Object Notation (JSON) file that represents the project from other users, this isn't an effective way and could take a long time to get a response. As such, the plan is to save the mosaics on the server so that users can click on the mosaic they want to start from and get the project file right away.

For the front end, we will develop features that are considered desirable and useful in our platform in terms of tools the user can use while designing their mosaics. Such features can be guidelines, shapes, or even automatic mosaic generation from an image. The concrete features to develop will be iteratively defined through interviews with the representative from the MosaicoLab Association as well as with the original author of the mosaic editor.

When these objectives are completed, we will then perform a usability evaluation of the new platform. Concretely, the objectives that were defined from the beginning for this work are to:

- Develop a server-side application for saving and sharing mosaics created with the web-based mosaic editor. This will entail functionality such as:
  - User registration and management (possibly managing roles and permissions)
  - Saving and loading projects from the editor directly on the server
  - Creating and managing collections of mosaics for organizing the public view on the webpage and for easier browsing.
- Extend the current functionality of the mosaic editor with new drawing tools. Examples of tools that might be developed (still subject to discussion) are:
  - New drawing shapes (rectangles, ellipses, etc.)

- Selecting regions (for erasing, painting, etc)
- Area painting tools
- Addition of guidelines to help drawing

Some requirements proposed by the student are to:

- In the server-side application:
  - Allow users to create private mosaics (mosaics that can't be shown publicly but the user has access to)
  - Allow users to create personal collections (where they would be able to insert their private mosaics and build a collection that follows a certain topic that the user enjoys)
  - Change the language of the platform (between Portuguese and English)
- In the editor:
  - Save the mosaics from the editor into the server-side application

Some possible changes to the original editor were pointed out, namely certain buttons weren't correctly styled. Meaning they appeared as plain text and not necessarily as a button.

Furthermore, changes to the tab were proposed, as to provide a easier understanding of how the platform is going to work.

## 1.5 Structure of this document

The rest of this document is structured in the following way:

The Background chapter presents in detail the existing mosaic editor tool, from the perspective of the functionality available to an end-user but also the underlying technical software architecture.

The State of the Art chapter will explore some other tools and explain the way they work and how we can learn from them to further develop our project. Then the methodologies and the work plan designed for the work done so far and the work to come will be presented. In this chapter the risks associated with the project will also be presented as well as a reflection on the risks that occurred from the ones presented.

Succeeding the Methodologies and work plan chapter the requirements for the new functionalities will be presented, explaining what they are and why they are needed. Subsequently, the achieved architecture for this project will be presented for the Mosaic editor and for the Mosaic Sharing Platform.



The Implementation chapter will clarify the process of implementing this project, taking into consideration the most important aspects that are needed to accomplish the requirements and the quality attributes proposed.

Following the Implementation chapter, the evaluation of the project will take place, taking into consideration the achieved elicited requirements, the achieved quality attributes and an overall view of the project.

To finalize this document, a conclusion chapter will be presented where a reflection on the work done, the challenges that were faced as well as some suggestions for future work with these project will be given.



# Chapter 2

## Background – Mosaic Editor

In this chapter, we describe the currently implemented mosaic editor tool and provide details about its functionalities as well as about the underlying software architecture. This will allow for a better understanding of how the current project fits into what is already developed.

### 2.1 Functionalities

The currently existing platform is presented as seen in figure 1.1. We can notice the grid of squares that occupies the central portion of the screen. This is where we are going to build our mosaic by filling each square with a certain color. As seen on the left side of the image, we have a tab we can open that has several categories.

The first of which allows us to define the grid size. By altering the grid size, the component of the grid changes and it will take over different dimensions on the screen. We are also able to use a viewport which lets us specify in which part of the mosaic we want to focus our screen using the arrows.

In this side panel there is also another tab that alters the grid, that tab is the “Shapes” tab, here we have a dropdown menu with the different shapes we can choose as tile shapes, the options are circles, squares and triangles.

The colors available are displayed in a row on the top of the page. To change between these colors the user has to click or touch the color that they wish to use next. In the side panel we also have a tab available for the colors, when opening that tab we are presented with a dropdown menu and the colors we have currently. The user can change the color palette in the side panel as well. Currently the palettes are solid colors and textures.

The other tab that has yet to be covered is the “File” tab. This tab has the options to import a project, export a project, import SVG and one to export an image of the project. When exporting a project, a download of a JSON file that represents the entirety of our project occurs. This is very helpful if the user doesn’t want to

lose a project and intends on working on it later or on another day, from which they would press the Import project button to do so. The export image which would download an image of their mosaic in case they want to show it to other people or print it.

There is also the option of returning to the main page of the editor that shows some of the different mosaicing styles and patterns the user can get to know and try. The full screen allows the user to focus solely on the editor by having it occupy the entire screen. The undo and redo options allow the user to go back and forth between actions they have taken on the mosaic (filling squares, clearing squares). Lastly, if the user doesn't want to open the side panel and choose from the export project and image options, they can use the save option that will automatically download images of the mosaic.

## 2.2 Software Architecture

As mentioned previously, this project has a defined architecture which can be seen in figure 1.2. To better explain how it all works together we'll be looking at the class diagram step by step and get a deeper look into the behavior of some of the main groups of classes.

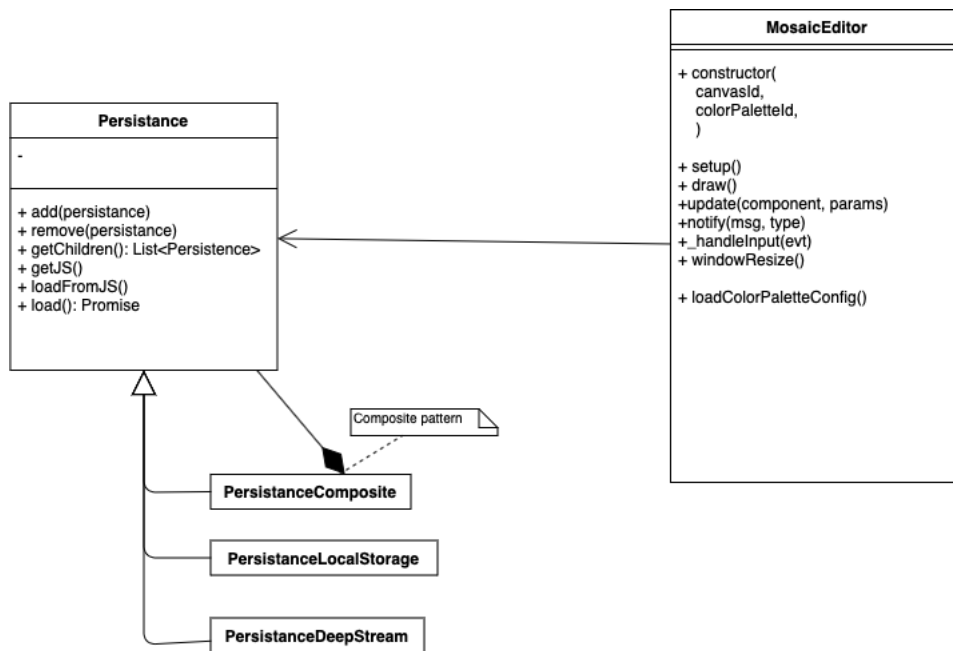


Figure 2.1: Class diagram representing the MosaicEditor class

Firstly we have the **MosaicEditor** class, seen in figure 2.1, this is the central point of our project in the sense that all other classes are called and initialized from here. Here is where we set up the editor, load the color palette, and create the canvas which is the portion of the editor represented by the mosaic (grid). Furthermore, the update method is present in this class, this method updates a component depending on the command executed. There are several event listeners as well as handlers for each event.

From here we also create the Persistence class which follows the Composite pattern [Gamma et al., 2009]. What this means is that we treat each component and the collective in the same way. For example, in our case, we have tiles, which are components of the mosaic. On altering the tiles, we are altering the mosaic. Throughout this project, several components have their state altered through changes in smaller components. Therefore, it is required to initialize the “Persistence” class right as we load the MosaicEditor class.

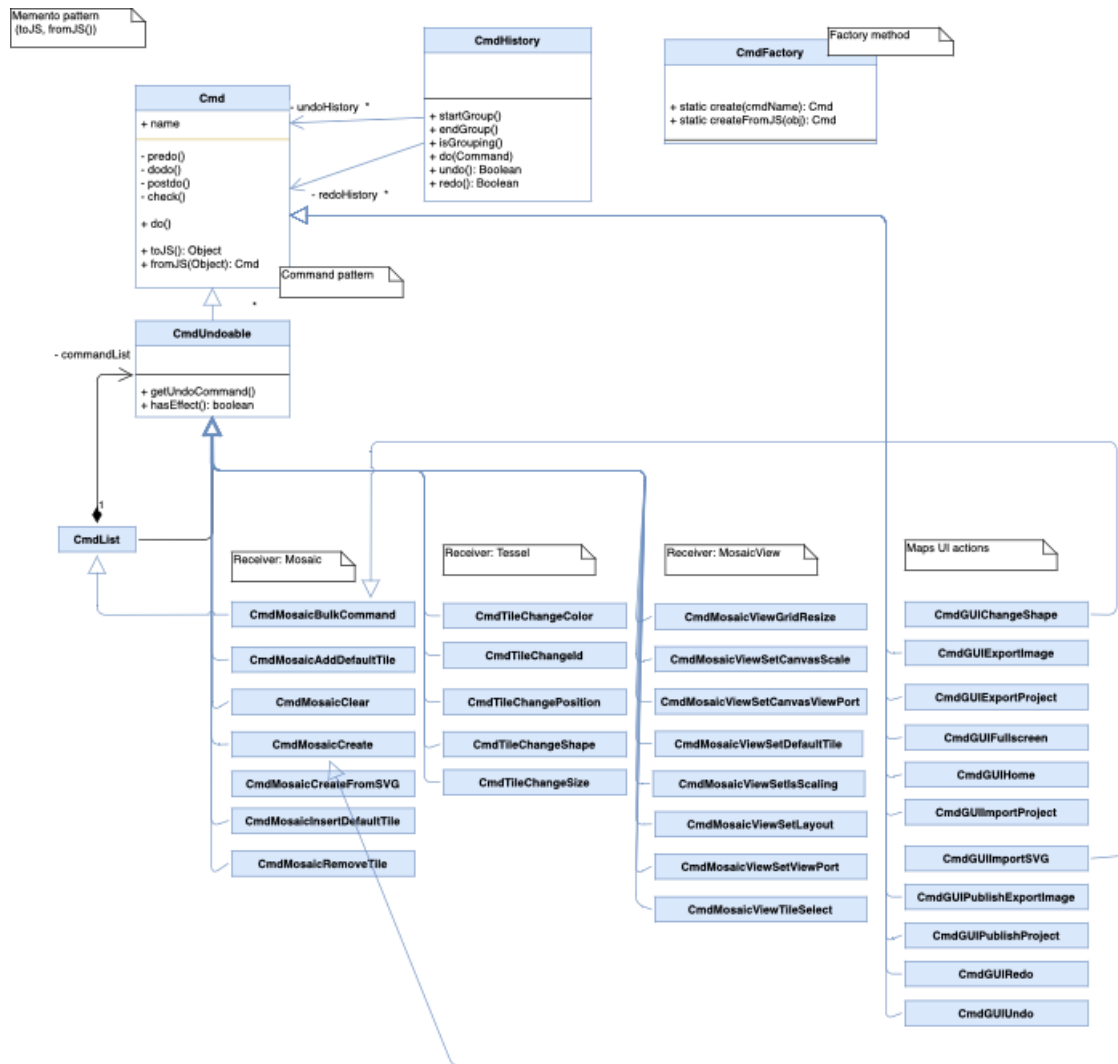


Figure 2.2: Class diagram representing the Cmd class

In the class diagram presented in figure 2.2, we can see the Command group of classes. Commands (**Cmd**) represent actions that can be performed in the editor. This group of classes follows the Command design pattern [Gamma et al., 2009]. Commands are executed through the **CmdHistory**, which provides undo/redo functionality to the mosaic editor. The **CmdHistory** class keeps a list of executed commands and is thus able to undo the previous command(s). Only undoable commands (**CmdUndoable**) are recorded by the **CmdHistory**. Commands such as saving the project are not undoable and, thus, are not kept in the **CmdHistory**. Commands are also structured around the composite pattern: basic commands may be composed into more complex commands by grouping them in a **CmdList**.

There are several concrete commands for performing different kinds of actions in the application:

- Commands that affect the mosaic structure or contents (e.g., CmdMosaicClear, which clears the colors of all tiles in the mosaic.)
- Commands that affect a single tile
- Commands that affect how the mosaic is displayed to the user (e.g. CmdMosaicViewSetLayout, which changes the layout of the mosaic)
- Commands that are directly invoked from the UI

Commands that can be undone are commands that directly change the canvas, such as painting one or multiple tiles and clearing one or multiple tiles. In this class, we have a subclass named “CmdFactory”, this class is responsible for verifying what command was requested. Several commands alter classes outside of the “Cmd” class. We can alter the Graphical User Interface with commands such as toggling fullscreen. We also have commands that change the Mosaic class, such as adding tiles, clearing the mosaic, or creating a new mosaic.

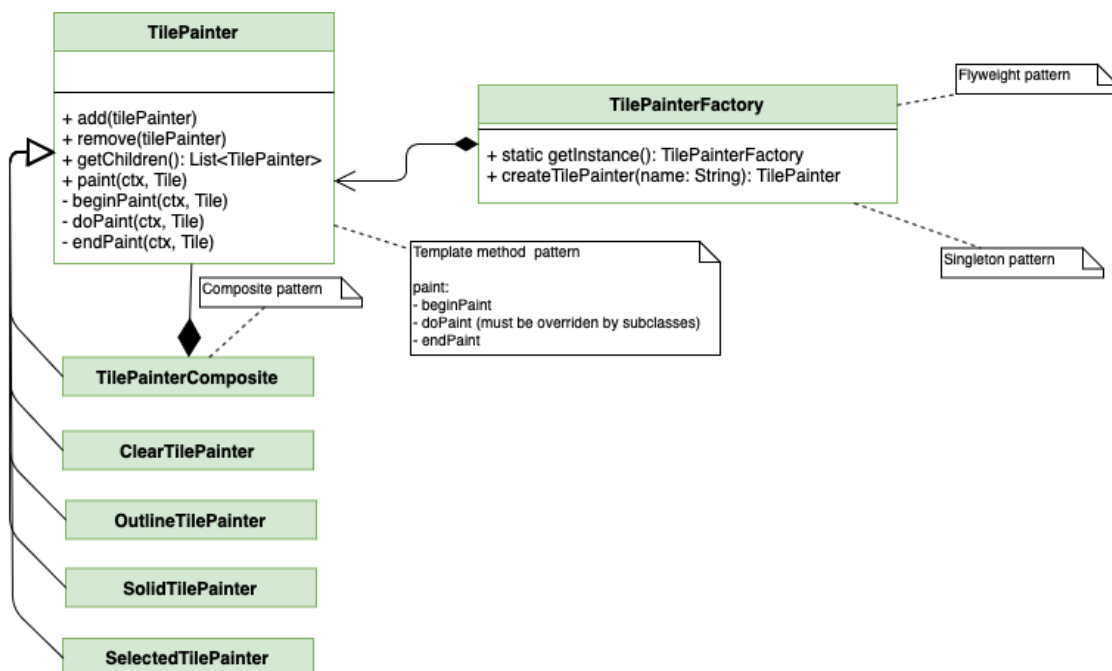


Figure 2.3: Class diagram representing the TilePainter and associated classes

As seen in figure 2.3 we also have the **TilePainter** class. This class is used to fill in the tiles with the color picked from the color palette. It receives the Solid colors or Texture colors chosen from the color palette and attributes those values to a painter class which will then be activated when clicking on tiles to fill them in. With the **ClearTilePainter** class, we can erase the color we had associated with the painter. This class uses the Template Method behavioral pattern [Gamma et al., 2009]. Which is mainly seen through the different methods involved in painting, these methods are:

- “beginPaint”, which prepares the color chosen by the user to be used to fill in the tiles
- “doPaint”, that uses the previously defined color and fills in the tiles selected by the user with that color
- “endPaint”, which restores the context of the painter to its original state

The composite pattern is once again visible since the CompositeTilePainter has several TilePainters. The singleton creational pattern [Gamma et al., 2009] is also visible in the tilePainterFactory class, meaning we only have one instance of the tilePainterFactory. This factory can then create several TilePainters that compose the CompositeTilePainter. Furthermore, the Flyweight structural pattern [Gamma et al., 2009] is also present in the tilePainterFactory. This pattern uses “. . . sharing to support large numbers of fine-grained objects efficiently.”. These fine-grained objects can be seen as tiles, and we have a TilePainter for each of them.

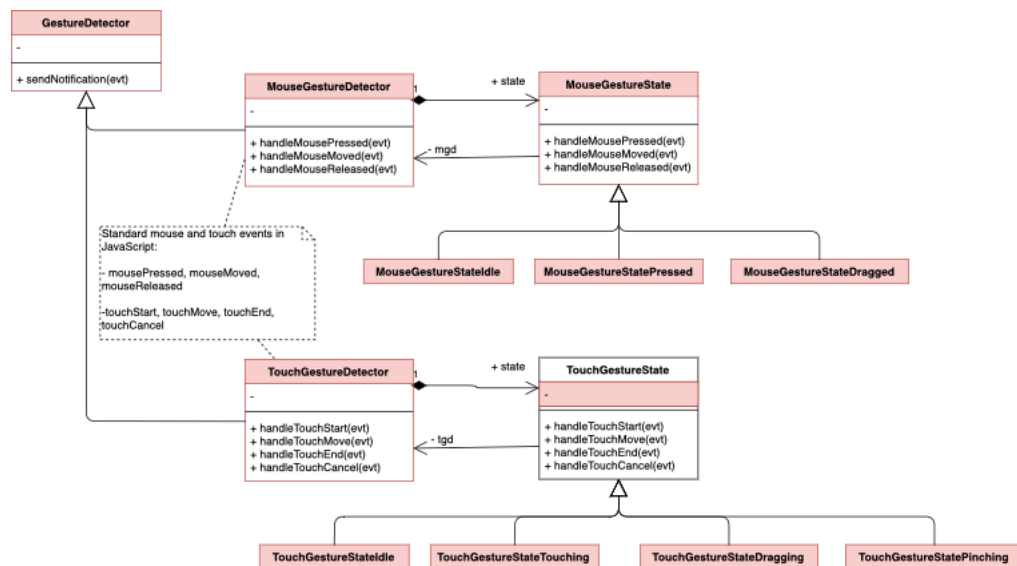


Figure 2.4: Class diagram representing the GestureDetector class

In the Gesture class, seen in figure 2.4 we decide what happens with certain gestures the user can use. We have the “GestureDetector” class that will detect what kind of gesture was used. Knowing the gesture, we are able to call the different methods that are needed to accomplish the action the user is trying to perform. For example, the gesture “pinch” is available on touch interfaces and is used to zoom in or out of the mosaic. In this class we also deal with zoom in and zoom out options from the computer “cmd + scroll”. In this category we handle those events as well as wheeling events.

All these actions are all dealt within this category and end up changing the interface in some form. Furthermore, this class uses the State pattern [Gamma et al., 2009], this means an object can change its behavior based on its internal state at runtime without resorting to conditional statements, therefore it improves maintainability of the platform.

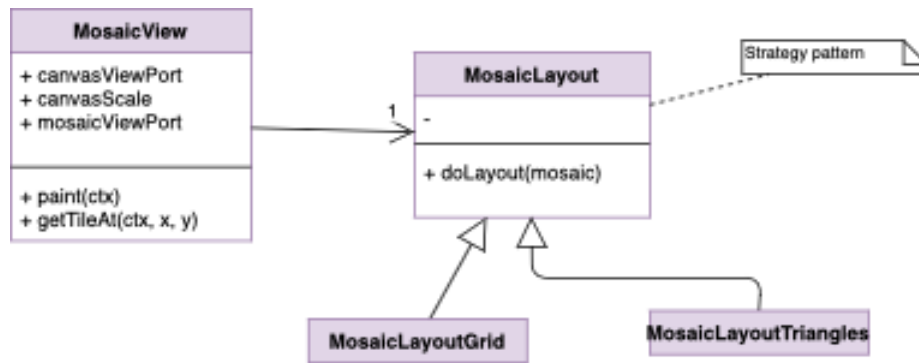


Figure 2.5: Class diagram representing the MosaicLayout and the MosaicView classes

The MosaicLayout class, seen in figure 2.5 has the responsibility of creating the layout of the grid of tiles. There are two different kinds of layout, these differences can be seen when picking the tile shapes. When picking the rounded square tile shape or the circle tile shape, the layout doesn't change, they occupy the same space and are aligned side by side. However, when picking the triangle tile shape, the layout changes. With this shape we have to perform some operations to make sure the layout is correct, when working with this type of layout we have to alternate between triangles and inverted triangles so it's different from the other tile shapes layout. One important aspect that is important and common to all the layouts is the setting of each tile in a certain position, the assembling of the mosaic, this is managed by the "doLayout" method. This class follows the Strategy behavioral pattern mentioned Design Patterns [Gamma et al., 2009]. This class is called from the "MosaicView" class. Which is where we set the viewport of the canvas, set the canvas scale and set the mosaic viewport. Meaning here is where we deal mostly with dimensions of the grid and focus points of the grid as well. By choosing a different viewport (available on the left panel in the interface seen in figure 1.1) we'll be calling this class to take over and redimension our canvas accordingly.

The Mosaic class, seen in figure 2.6, deals with creating the mosaic which is a list of tiles and setting the tiles where they are supposed to be. Therefore we have the tile class in the same category as the mosaic. The mosaic category is also in charge of defining the tiles shapes. Since we have the tile class in this category, we have to define certain values to each tile object we have. These values are the height, width, and position of the tile in a cartesian manner (using x and y coordinates). Furthermore, here we also control the color of the tile and the shape. In the Tile class we have functions that allow us to paint the tiles and a function that determines if a tile exists in a certain point. The latter function can be used when using the capability of dragging the mouse while clicking down to fill in the tiles in the path we are drawing. This is done by checking a point where the mouse passed by and determine if it was a particular tile, if so, we paint it, if not, we move onto the next tile in the list. The tiles only have one dimension once the height and width are defined. Each of the tile objects only has one position which is most of the time managed by the layout category.

The last class left is the Observer class, seen in figure 2.7. This class follows the



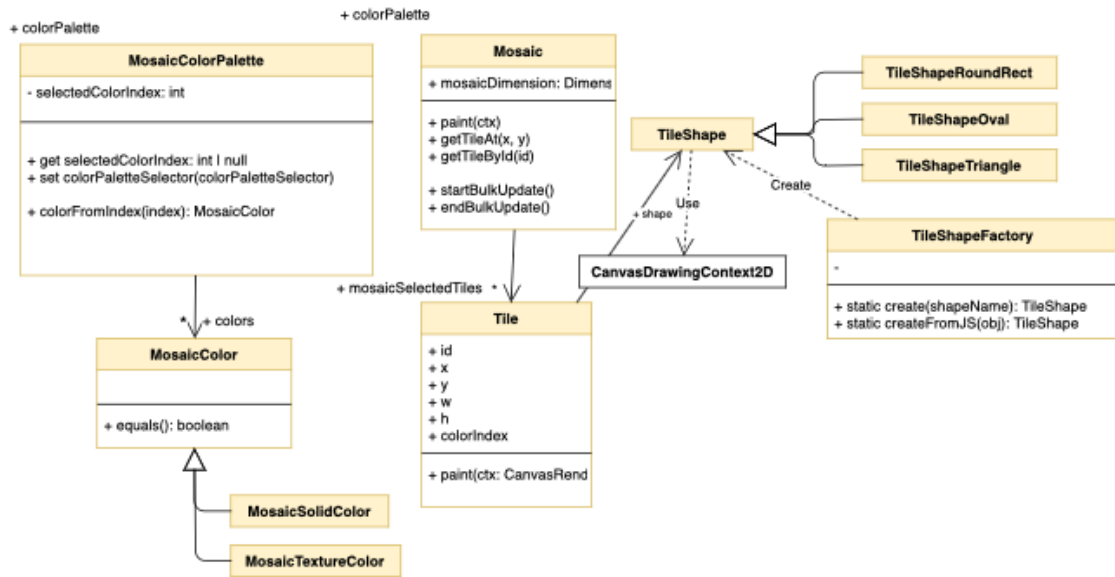


Figure 2.6: Class diagram representing the Mosaic and Tile classes

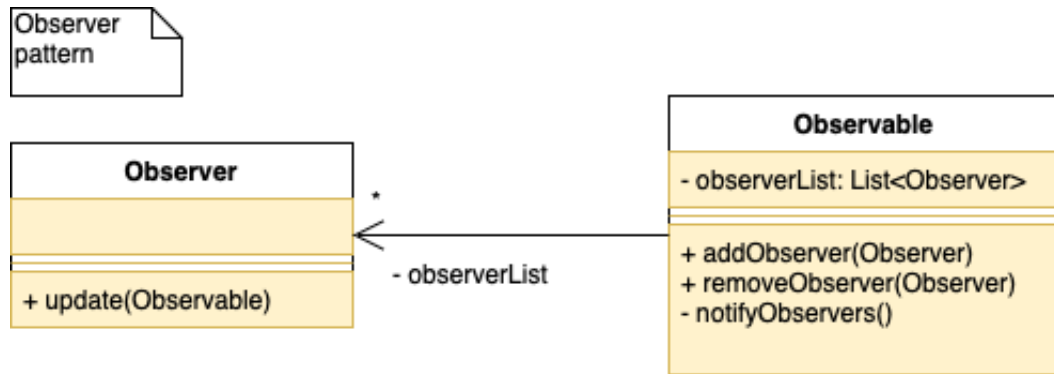


Figure 2.7: Class diagram representing the Observer class

Observer behavioral pattern [Gamma et al., 2009]. This means that changing the state of a class will notify others that observe it. As seen in figure 2.8, by changing the Tile we notify the Mosaic that will then change its internal state accordingly and notify MosaicView which will then notify the MosaicLayoutGrid.

When changing a tile, all the subsequent classes will suffer state changes. In this class we establish the observers and control the needed updates whenever a state changes.

### 2.3 Code base

The code base of the project follows the structure presented above by grouping each file that interferes with a certain class together in a folder with the name of the class as seen in figure 2.9. In most cases, the name of the class corresponds with the name of the class in the class diagrams from figure 2.1 to figure 2.7. This allows for a better comprehension of the flow of information on the platform and

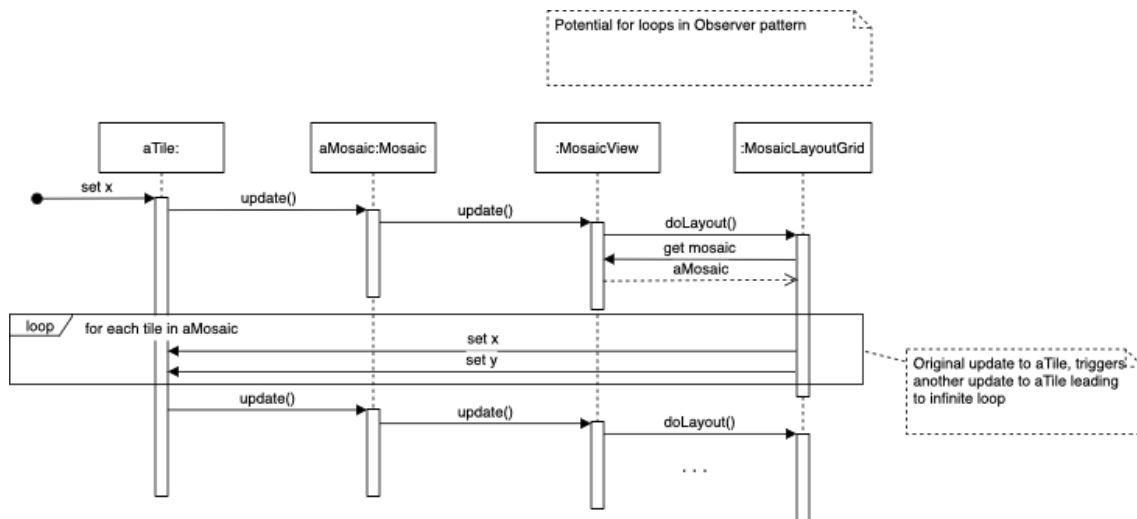


Figure 2.8: Diagram representing the Observer behavior

how different commands are executed.

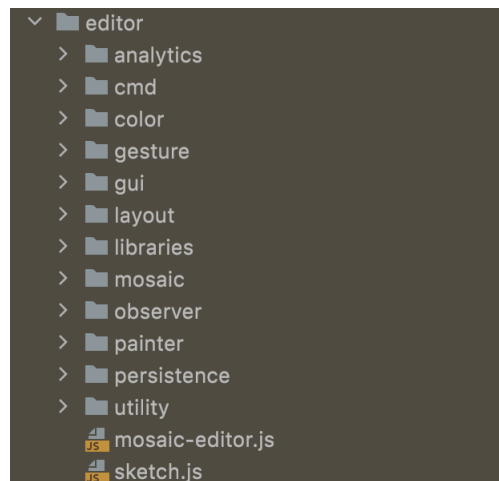


Figure 2.9: Structure of the classes on the existing project

Besides the JavaScript classes that exist in the project, one other aspect is crucial to the functioning of this platform and that is the configuration files. There are several JavaScript Object Notation (JSON) files that contain information needed to set up different components of the interface. The type of configuration files can be divided into two:

- General configuration of the application
- Configuration of the color palettes

In the general configuration of the application, the definition of buttons and their associated commands takes place. As well as the setting of default values for certain components, such as viewport, grid size and tile shape. Here we also call the configuration files that handle the configuration of the color palettes. The structure of these files can be seen below.

```

1 {
2   "homeButton": { "value": { "cmdName": "CmdGUIHome",
3     "target": "index.html" } },
4   "undoButton": { "value": { "cmdName": "CmdGUIUndo" } },
5   "redoButton": { "value": { "cmdName": "CmdGUIRedo" } }
6   .
7   .
8   .
9   "palettes": [ { "name": "Marmores",
10     "url": "resources/marmorePaletteConfig.json" },
11     { "name": "Cinca 33 cores",
12       "url": "resources/cinca33PaletteConfig.json" },
13     { "name": "Cinca 24 cores",
14       "url": "resources/cinca24PaletteConfig.json" },
15     .
16     .
17     .

```

In the color palettes configuration, there are again two types that can be distinguished, those are the definition of solid colors and the definition of textures. When loading textures, the objects referenced by the configuration file are image resources. When dealing with solid colors, the object is a hexadecimal value that defines a color. Below is presented the structure of the textures JSON file.

```

1 {
2   "name": "cinca24PaletteConfig.json",
3   "textures": ["resources/cinca/0201.png",
4     "resources/cinca/0129.png",
5     "resources/cinca/0140.png",
6     "resources/cinca/0131.png",
7     "resources/cinca/0114.png",
8     "resources/cinca/0112.png",
9     "resources/cinca/0215.png",
10    .
11    .
12    .
13  }

```



# Chapter 3

## State of the Art

Since the objective of this project is to improve a currently existing platform used for mosaicing, the intention of this section is to explore the competition. This means we are going to explore other tools that allow the user to draw mosaics. By analyzing these tools we will get a better understanding about what features other mosaic editors have and which of those would make sense implementing on the MosaicoLab's editor. After exploring mosaicing tools we will be having a look at design tools, tools that allow the user to draw. This will be done so that we can review more features that are not directly related to mosaicing, but might be very useful for the creative process nevertheless. After exploring these tools, we are going to start exploring platforms that might not be related to mosaicing but have features we intend to implement on our project, giving us a better understanding of how they should work.

### 3.1 Mosaic Editors

In this section, an overview of several online tools that share the purpose of this platform will be presented. The main purpose of these tools is to allow users to draw mosaics. The objective of this study will be to analyze and discover possible features to add to our mosaic editor from MosaicoLab.

#### 3.1.1 Macropolis

The Macropolis<sup>1</sup> tool works in a similar way to MosaicoLabs' tool, when we first get to the platform, we are presented with a grid of white squares and a color palette from where we can choose the colors we want to use. When accessing this editor, the user is presented with a grid of squares representing the mosaic tiles. On the right side, there are colors to choose from as well as patterns to fill in. When using this editor, the user is able to fill in the tiles by pressing each of them individually. When choosing the patterns to fill in, a layout is generated where

---

<sup>1</sup><http://macropolis.org/banderillas/famosaic/easy.htm>

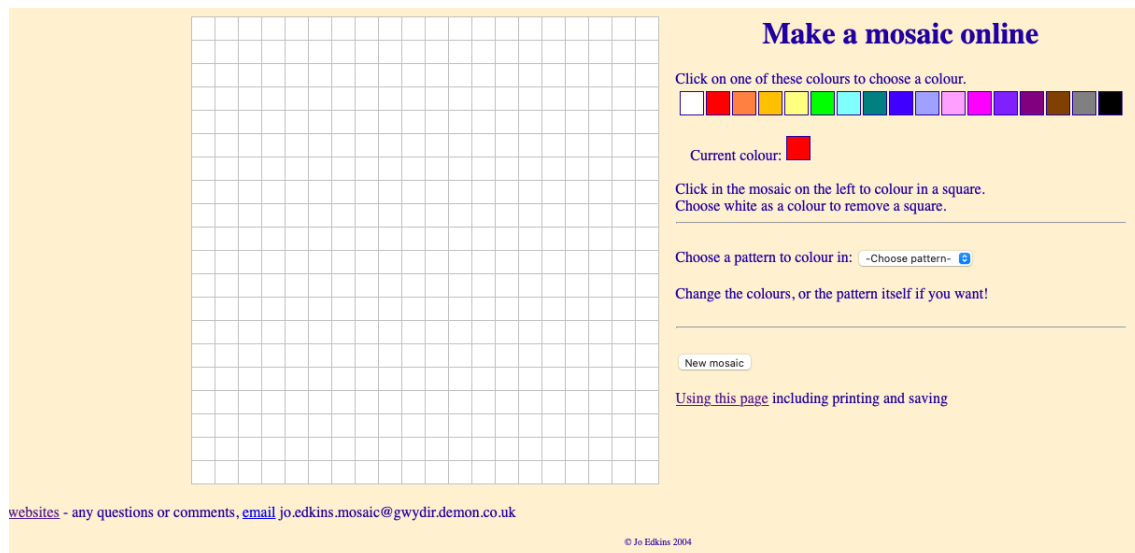


Figure 3.1: Interface of the Macropolis Mosaic Editor tool

several tiles are colored in grey, as seen in figure 3.2. By having some tiles filled in with grey, a design is built, from here, the user is free to pick any color they want to use on their mosaic and fill in the tiles according to the guidelines provided.

The way these patterns work is by painting specific tiles in a grey shade to exemplify the structure of a mosaic that would follow that pattern, this can be seen in figure 3.2.

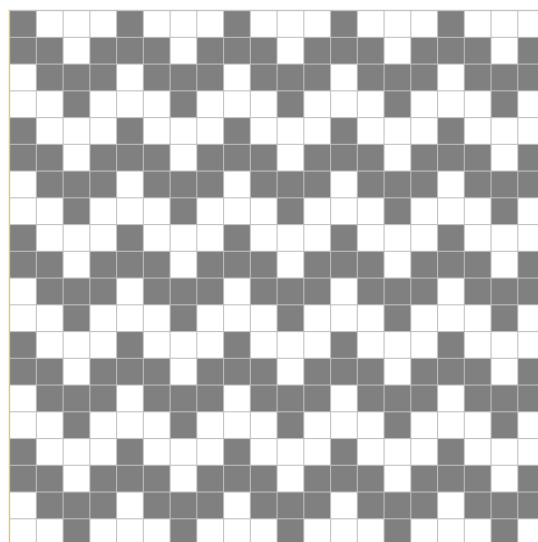


Figure 3.2: Macropolis editor when picking the chevron pattern

There are several options of patterns to choose, some of them are:

- Chevron, which presents a layout where several tiles are grey in order to build this design. This design is composed of several triangles, alternating between inverted and regular triangles
- Diagonals, this design fills tiles to build a layout where diagonals are filled in

- Squares, here the layout is composed of several rows of squares of the same dimension

When comparing this tool with MosaicoLab's editor, there are some clear differences worth noting. Firstly, in this editor, the user is not able to re-dimension the mosaic, the grid size is fixed. Furthermore the user can not make use of the dragging functionality. To fill in several tiles, the user must select each and every tile they wish to paint. This suggests that the tool was built using HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) and therefore does not have a JavaScript method to handle mouse dragging events.

There is a button on the editor that allows the user to save and print the mosaic they have drawn, however it does not work. The colors available are limited to the solid colors presented on the interface in figure 3.1. The patterns are a feature that does not exist on the MosaicoLab's editor and could be considered as a way to guide users to fill certain patterns that might be complex to do free hand.

### 3.1.2 Learning Playground

In this platform<sup>2</sup> the user is presented with a lot of options from the start, similarly to the other mosaic editors, a grid of squares is presented, as seen in figure 3.3.

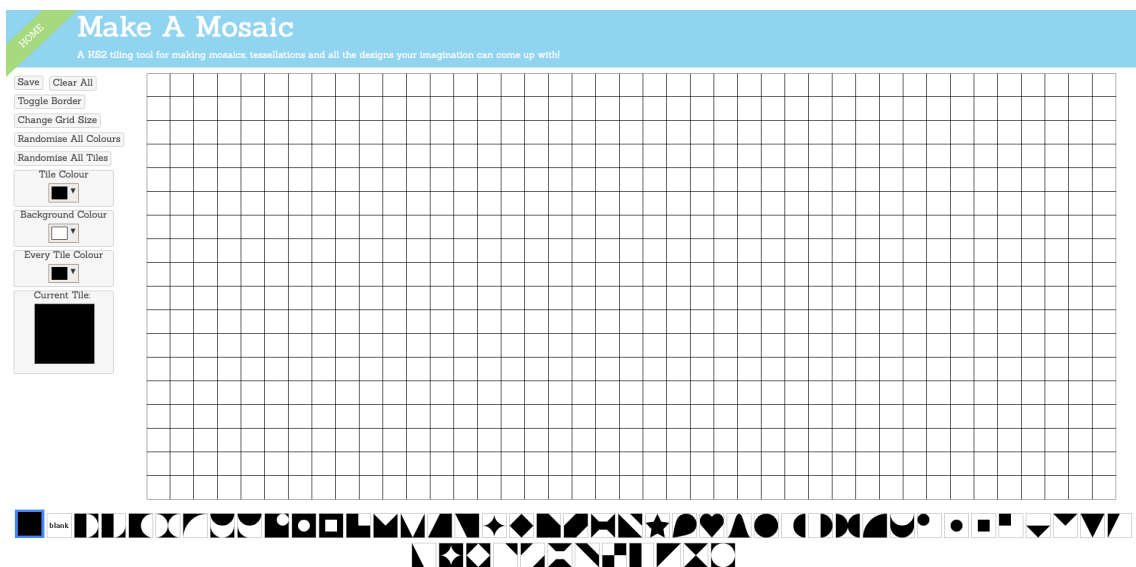


Figure 3.3: Interface of the Learning Playground Mosaic Editor tool

The grid of tiles is re-sizable. When choosing the "Change Grid Size" option, the tiles become larger therefore occupying the canvas and not allowing the rest of the squares to appear. This operation is not done the same as in the MosaicoLab's editor however. Here, what is being called a re-size of the grid is actually a change in viewport. The focus of the canvas is directed into the top left corner and makes

<sup>2</sup><https://www.learningplayground.co.uk/mosaic/>

those tiles bigger, just like a user could do by "zooming in". This means that if a user had already painted in this editor and toggles the change of grid size, the painted tiles remain painted but not visible due to the focus change to the top left corner.

This editor also has the option to toggle border, this option makes the lines that limit the tiles disappear. There are also several features involving the tiles available. Firstly, as is presented in figure 3.3, there are several tile filling options, these layouts are filled with a tile color and a background color. When filling out the mosaic with the tiles the user intends on using, they can then choose to randomize all colors. These allows for a variety of colors to appear on the mosaic. These operation changes not only the tile color but the background as well. There is also the option of filling the mosaic with random tiles. The colors can also be chosen by tone and hue, allowing for a great diversity of colors on the mosaic. When saving the project, a Portable Network Graphics (PNG) file is saved onto the computer being used, saving therefore, an image of the mosaic.

When comparing this mosaic with MosaicoLab's, there is a big difference in what the user is able to do, mostly due to the possibility of randomly filling the mosaic with different tiles, or randomizing all colors as well as the different tile designs available the user is able to choose. However, MosaicoLab's editor is focused on workshops and allowing the users to have a realistic view of what they can accomplish physically. Therefore, some of the features present in this editor wouldn't be necessary not prove to add great value for the MosaicoLab's editor. However, there are certain ideas that rise from analysing this tool. One could be, for example, allowing the user to pick a color present on their mosaic and replace all the tiles that are filled with that color by another color they wish to have. Even though this idea isn't directly related to the editor in cause, the randomizing colors inspired this possible feature which could allow our mosaic editor to become more interesting and dynamic. The idea of toggling the border could also prove to be very useful as a way to let the user see their mosaic without the digital black lines.

### 3.1.3 TileMaker

The TileMaker editor<sup>3</sup> is very different from the ones presented thus far, this is due to the fact that in this case, we are talking about an Islamic mosaic editor and not a roman mosaic editor. As seen in figure 3.4, this editor presents itself in a similar way as the ones presented in subsection 3.1.1 and subsection 3.1.2. There is a grid of squares which represents the mosaic and then several options for editing on the left. However, this tool focuses on symmetry and patterns.

Initially this tool has a centered dot on the canvas, from here, the user can drag the mouse starting from that dot, drawing a circle with a radius the size of the line the user defined by dragging the mouse. Several dots appear throughout the drawing phase of this editor that allow to draw circles or lines. Both the circles and lines drawn after the first circle need to connect two dots. Any changes

---

<sup>3</sup><https://tilemaker.qfi.org/create/>



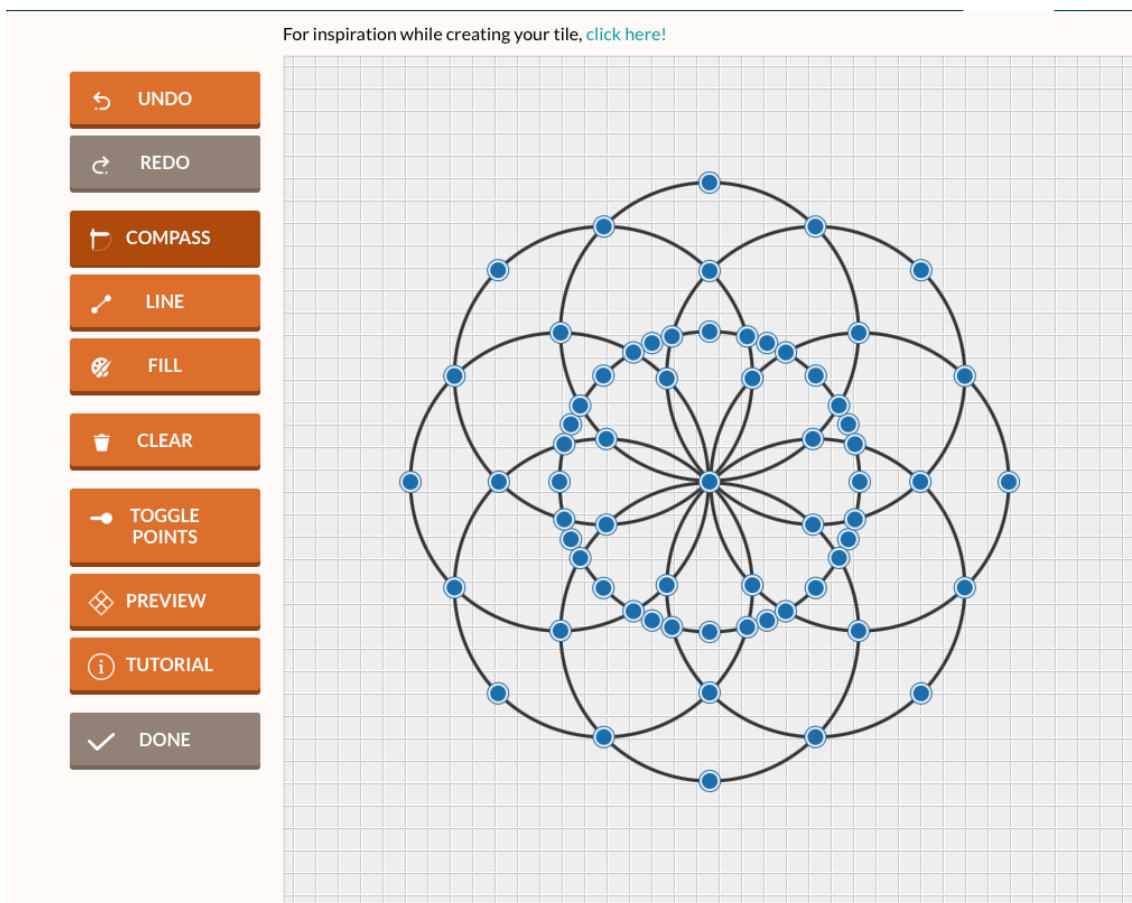


Figure 3.4: Interface of the TileMaker editor tool

done on one dot, for example, drawing a new circle or drawing a new line, will automatically be applied to all the other dots of that circle. This allows for the user to design very interesting layouts with only lines and circles that maintain symmetry.

Once the user finds that the drawing phase is completed, they can then proceed to the filling phase. This step consists on filling out the spaces created by the intersection of circles and lines. Whenever the user applies a color to a certain space, that same color is applied to all the spaces created the same way as the space filled out.

Another feature is available for the user and that is to preview the drawn mosaic. In this step, the user can toggle the "tesseleate" option which repeats the mosaic several times as if what the user drew was actually a tile.

This platform is very interesting in the way they approach designing a mosaic. Only using two design tools (compass and line), the user can build several different designs perfectly symmetrical by itself. This editor also has a tutorial on how to use it. Here is where the tools are explained, such as:

- "Compass - Start designing your tile by clicking the center point, moving you cursor to the desired size, and clicking again to create a circle. You can draw multiple circles around the original circle by clicking from one point

to another."

- "Line - You can create lines by using the line tool to connect one point to another along any circle."
- "Color - You can choose from the color palette or select a custom color by expanding the "show options" area and using the color picker."
- "Preview - You can view the tile individually or tiled by turning the "tileate" button on. Click "Back" and then "Done" to post your tile to the gallery."
- "Post to gallery - After you review your completed tile and are ready to post it to the Gallery, click the "Done" button. Your design will now be publicly available for view. You can also share your design with friends, colleagues and family through social media via Facebook, Twitter or Pinterest. You can also download your tile directly to your computer or share it with others through a direct URL link."

This editor presents several possibilities into mosaic design, however, the scope and objective of what is planned to be designed in this tool in comparison to what is expected from MosaicoLab's tool are not the same since this editor deals with designing Islamic mosaics and MosaicoLab's editor is focused on Roman mosaics. Therefore, some of these features are not suited for the MosaicoLab's editor. However, features such as a tutorial could become very useful as new features are added during this project in order to let the user know how the tool works. Furthermore, the "tileate" option was very interesting and could be implemented as a way of letting users see their creation repeated several times, as if creating a pattern in a bigger image. From a functionality standpoint, the fill feature would be a good addition to MosaicoLab's editor, by giving users the option of filling in areas between colored tiles. Even though the main tools used in this editor might not be suited for the scope of this project, there is always room to learn from the interface and what the user has access to.

### **3.1.4 Possible Features**

From the different tools analyzed in this section, several features that could be used for our own project can be noticed. Those features are:

- Macropolis 3.1.1
  - Provide pre-built patterns for the user to fill in as they please
- Learning Playground 3.1.2
  - Replace all color instances with another color
  - Toggling the border of tiles
- TileMaker 3.1.3
  - Tutorial (how to use the platform)
  - Tessellate
  - Fill-in feature

These features resulted from an analysis of what the other mosaicing tools offered and some interpretations of what can be used that would make sense in the scope of this project and the use case for this mosaic editor.

## **3.2 Drawing Software Applications**

The main purpose of this analysis is better understanding the types of drawing tools functionalities commonly offered by drawing applications and to reflect about whether some of these tools could be offered by the mosaic editor.

In order to accomplish a full analysis, three main platforms were taken into account, these are:

- Microsoft Paint
- Photoshop
- Gimp

From these platforms, several functionalities were extracted. These functionalities affect not only the editor and drawing capabilities but also external functionalities that exist on these platforms, such as sharing. After a complete review of these platforms, the functionalities achieved were divided between the following categories:

- Selection tools

- Painting tools
- Drawing tools
- File operations
- Image manipulating tools

### 3.2.1 Selection tools

From selection tools, the goal is to be able to create association between elements of the project. These can be:

- Area selection - using the cursor to select an area of the canvas that can then be copied, deleted, pasted somewhere else, generally using a rectangle area
- Special area selection - Instead of selecting areas through rectangles, the user can draw a figure and select the area inside that figure.
- Color selection - Selecting all instances of a color on the canvas

#### Area Selection

When it comes to selection tools, most drawing platforms supply the basic area selector tool, seen in figure 3.5. The default tool is generally applied by resizing a rectangle to fit the area the user wishes to select inside said rectangle. By doing so, the user can then apply any changes needed to the project. These changes, might be deleting the selected area, painting it with a single color or, copying the selected area to paste somewhere else.

This type of selection would be very helpful on the context of this project, since it would allow users to select entire regions of the mosaic. Furthermore, since the mosaics are composed of tiles, a rectangular shape would be the best shape to select correctly all the elements necessary.

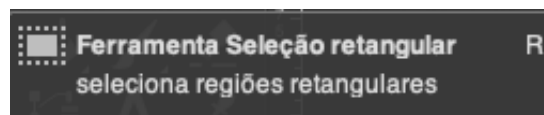


Figure 3.5: Rectangular selection

#### Special Area Selection

One other kind of selection tool is the Special Area Selection, commonly known as "Lasso" seen in figure 3.6. With this tool, the user can draw the area free handed and all elements present inside that area will be selected. After establishing the selected area, the functionalities are the same as the regular selecting tool. This

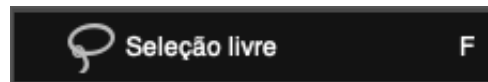


Figure 3.6: Free hand area selection

feature is generally present in Software that is used in a more professional level, such as Photoshop and Gimp.

Even though this tool makes a lot of sense and comes in handy for drawing software's, in the context of this project, it might not be the best. As mentioned in section 3.2.1, the rectangular selection makes more sense since it would be applied over a grid.

### Color selection

Besides these two area selection tools, there is one selection tool that differs from the previously explained in section 3.2.1 and section 3.2.1. In this case, the user won't define the area that they mean to select, they will however, choose the color they wish to select. This allows the user to select all instances of a color on a project. Sometimes this functionality has another feature which allows the user to select the immediately surrounding instances of the color or, selecting all instances of the color in the project. This means that if a section of a project is filled with a color, separated from the rest of the project by other colors (possibly including the same color) and the user selects that section, all other instances won't be selected if they choose not to. If they wish to, they can select all instances regardless of position on the canvas. This feature is presented on Gimp as seen in figure 3.7

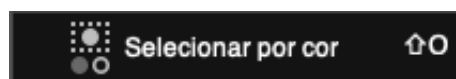


Figure 3.7: Color selection tool

This kind of selection would also make sense to implement on the project, since the user would be able to select all the tiles containing the same color at once without having to go through them one at a time.

### 3.2.2 Painting tools

This kind of tools have the goal of helping the user filling in their projects with color or shapes. Several tools can be used to accomplish this feat, namely:

- Fill in feature - To fill in spaces that are delimited by other colors
- Shape drawing - By selecting the shape and then clicking and dragging the mouse, the user can insert a certain shape into their design with ease
- Smudge - Mimics what would happen when rubbing a finger on paint

## Fill in

One of the most commonly known tools is the fill in tool, generally represented by a tipping bucket as seen in figure 3.8. This tool helps the user paint a certain area of the same color. The area to be filled is delimited by other colors or by the borders of the canvas, therefore not allowing for the color to leak to the other side of the already existing colors. Using this functionality the user can save time and not paint by hand the whole space, which would likely lead to small mistakes.



Figure 3.8: Fill-in tool icons for Paint, Gimp and Photoshop

## Shapes

One other tool that fits into this category is the Shapes tool, this allows the user to insert shapes into their project. For example, if the user wishes to draw a rectangle they can do so easily by inserting that shape. The way this functionality usually works is by presenting the shapes available to the user, the user can then select the shape they intend to use. After having chosen a shape, the user places the mouse pointer on the canvas, clicks down, and then drags the mouse until they are happy with the dimension of the shape. Some of the shapes available are:

- Rectangle
- Oval
- Animal shapes

## Smudge

In the category of painting tools, it's also worth mentioning the smudge functionality, this helps user create a natural blur effect on their projects. By clicking down and dragging the mouse, the user can create an effect as if smudging paint with their finger. This can be particularly useful when using digital tools such as the ones mentioned before to create realistic projects. This feature is the smudge tool and it is presented as seen in figure 3.9



Figure 3.9: Smudge tool icon

### 3.2.3 Drawing tools

These tools are mostly related with helping the user reach an end goal with their drawing. To do so, the user might use tools that help inserting lines in order to create a certain element on the canvas or, use tools that allow them to better place their elements on the canvas.

- Line drawing - After selecting this option, the user should click and then drag the mouse to draw a line
- Guidelines - Lines that help the user align certain elements of their project
- Draw path - Allows users to draw a line and modulate it with curve.
- Layers - Allows users to fix a drawing and then draw on top of it (different layer) without causing changes to the previous drawing
- Color picker - Besides picking colors from the palette, the user can choose colors from elements in the canvas. Since the user is able to import images from their computer, they can then use the color picker to choose colors from those images

#### Line drawing

Line drawing is a functionality that allows users to draw a straight line easily. If the user did not have this functionality, they would need to draw lines free handed which would most likely not be a straight line. With this functionality, it's also worth mentioning the draw path, this helps user draw curves in a seemingly natural way. By drawing a line and then manipulating the angle at which it bends and curves, the user is able to draw smooth curves.

#### Guidelines

Guidelines are an extremely helpful feature most drawing tools offer their users. It helps to keep elements of the project align by placing an easily removable line (which is not part of the project) onto the canvas. The user has the possibility of placing this lines wherever they prefer. This feature not only helps to keep elements align, but when using both vertical and horizontal lines, the user is able to achieve a balanced layout. This feature usually comes hand in hand with the possibility of inserting a grid over the canvas, allowing the user to more precisely align small elements and placing them where they are supposed to be.

## Draw path

These functionality let's users easily draw smooth curves. To use it, the user must first draw a line, from that line the user is free to continue adding lines. By adding lines to the existing line, it will form a path since the lines are connected. The user can then drag the mouse to smooth out the angle of connection between two lines, creating a curve. This helps the user creating curves with different angles. This functionality is usually presented as seen in figure 3.10.



Figure 3.10: Draw path functionality icon

## Layers

Layers allow the user to add another canvas on top of the original canvas. This means that if the user is happy with the drawing they have made but they still wish to add more elements in the background or foreground, they can add another layer. By drawing on the newly created layer, the original layer remains the same, however, the image the user sees will change with the applied changes on the new layer. After creating a layer, the user is free to navigate between all the layers of the project. This functionality's icon is commonly represented as seen in figure 3.11.



Figure 3.11: Layer functionality icon

## Color Picker

Color picker is a tool that helps user keep consistent color usage throughout a project. When clicking the color picker, the user then needs to select a color from the canvas that they wish to re-use, that color will then get selected and applied to whatever painting tool the user is making use of next. This is usually represented by a dropper, as seen in figure 3.12.

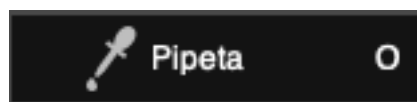


Figure 3.12: Color Picker functionality icon

### 3.2.4 File operations

These functionalities relate to capabilities the user can make use of with their final project. These might be saving it on a particular format or, sharing it online.



Export and publishing capabilities are another functionality these platforms offer, when exporting, the user is able to choose the format they intend to save the image. With these several formats, the user is able to export their projects into whatever format is needed, for example Tagged Image File Format (TIFF) is commonly used to print images.

Besides being able to export in several different formats, it is also important to let the user share their projects directly into the web. By directing the user to websites they might want to publish the picture, the user won't have to export the files and consume physical memory in order to store the image and then share it.

In the context of these project, certain file formats can be made use of, such as TIFF which would allow users to print their mosaic in a better resolution. Furthermore, the ability to publish the designed mosaic directly into the web is a feature that could be very useful, not only for the user to share with their friends the project they designed but also to give the platform more visibility.

### **3.2.5 Image manipulation**

In this section, several tools that allow the user to modulate their project without actually adding elements to the canvas will be presented. These tools are:

- Rotate - To rotate a selected element of the drawing or the whole drawing according to a predefined axis
- Cropping - Allows user to crop their images according to their needs
- Transform - Allows the user to set points on the image and then drag the mouse to transform the image into a new perspective

The rotate functionality is usually used to align vertically or horizontally a drawn element of the project or even the whole image. If the user imports an image that is slightly tilted towards the left or right, they can easily fix that by rotating the image. This functionality works by dragging the corner of the canvas element to the left or right, depending on the direction the user intends to rotate.

The cropping functionality allows the user to cut the margins of the project they don't intend on keeping. These can be resized to the user needs horizontally and/or vertically.

The transform is different from the other two as it allows for a 3D manipulation of the picture. By establishing points on the image, the user can then drag each point as to give it some relief.

These features are very common in drawing platforms and the last one in more professional software. However, they aren't properly fit for the project at hand. The rotation can be done once the mosaic is exported as an image and the cropping would just mean deleting rows/columns of tiles. The transform also doesn't

make sense as mosaics are usually 2D and in the scope of the project, they are intended to be kept that way.

### 3.2.6 Possible Features

From the different drawing tools analyzed in this section, some of them can be taken into account for our project. Those tools are:

- Selection tools 3.2.1
  - Area Selection 3.2.1
  - Color Selection 3.2.1
- Painting tools 3.2.2
  - Fill in 3.2.2
  - Shape Drawing 3.2.2
- Drawing Tools 3.2.3
  - Guidelines 3.2.3
  - Color Picker 3.2.3

These tools result from the analysis performed as well as some pondering on what would make sense to add to the editor. Since it is a drawing platform but the product should be realistic and aligned with what is expected to be possible doing at the Workshops held by MosaicoLab.

## 3.3 Re-mixing platforms

In this section, several platforms that allow the user to start projects from previously built projects will be analyzed. Even though these platforms don't directly relate to mosaic editors themselves, the feature of allowing users to start projects from previously built projects is desired for the MosaicoLab's editor.

### 3.3.1 Scratch

As mentioned in the beginning of this section, this tool<sup>4</sup> is about re-mixing projects. More concretely, re-mixing computer programs created by other users. The name of this platform ("Scratch"), was given as a reference to Disk Jockeys (DJ) that scratch the disks. This refers to the capabilities of starting projects on top of previously built projects and not having to create a new, empty project every time the user plans on using the platform.

---

<sup>4</sup><https://scratch.mit.edu>

To use this tool, the user must create an account giving their e-mail address, creating a username and a password as well.

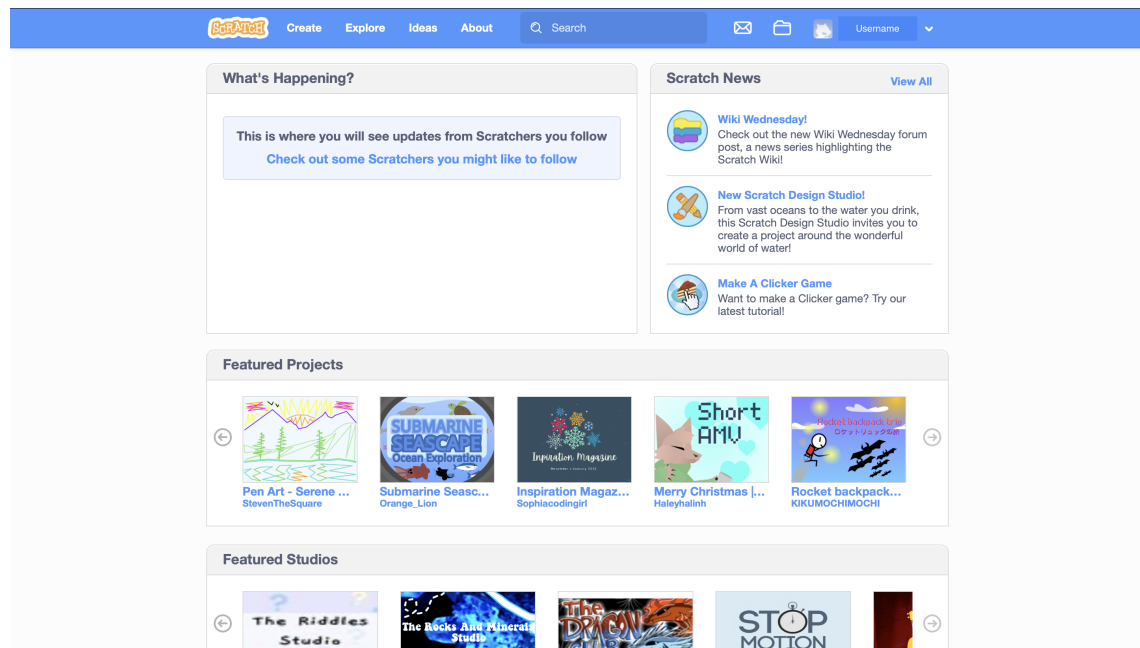


Figure 3.13: Interface of Scratch platform

As seen in the figure 3.13, the platform has a section that contains several projects. These projects were built by other users and any user can click on them to see some details about the project and then remix. Remixing being changing that project to better fit what the user intends to do, not changing the original project file but loading its contents into a new file.

Bellow the featured projects, the feature studios are presented, this studios are collections of projects that follow a certain theme.

When saving the project, it becomes a part of "My stuff" on the user profile, only accessible by the user. When sharing, the user can choose a studio to share it to or simply share it onto the main server.

In order to join a studio or be able to create a studio, the user must prove themselves as an active member of Scratch's community, by creating projects and commenting on projects already built. Private collections are possible by creating a studio and not allowing other users to add projects to that studio or comment, however, users can always follow studios.

When creating a studio, the creator gets the role of manager of the studio and is able to change the settings of the studio. The user can also attribute roles such as manager and curator to other users. Curators do not have all of the permissions that managers have like changing the settings of the studio, but they can still add and remove projects that they add themselves [scr, 2022].

Excluding Studios, there is no moderation of the contents published, this means any user can build a project and share it onto the server making it available for everyone to watch, regardless of sensitive content.

When pressing the create button, the user is directed to the page seen in figure 3.14. Here the user uses blocks of actions to create their history or animation. This tool works with the Blockly [blo, 2020] library. This allows for the user to create stories that are composed of several blocks. Each block represents an action on the interface, when stacking several blocks the user is able to create complex actions and stories.

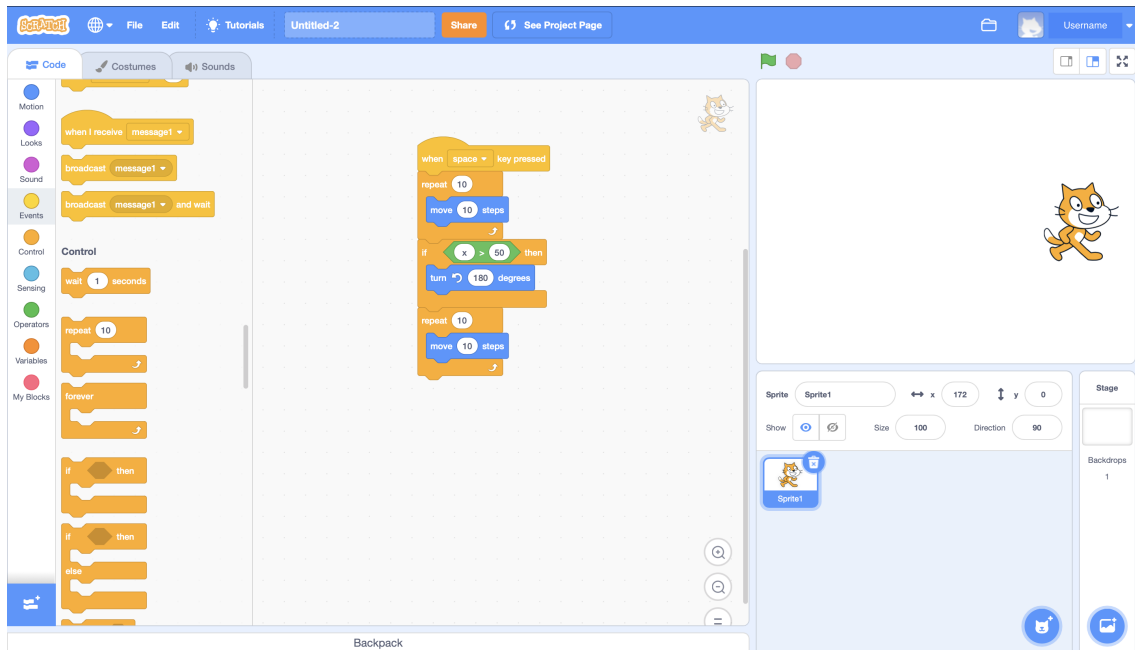


Figure 3.14: Scratch’s creating interface

The capability of allowing users to start their project from previously built projects would be very interesting to apply into the mosaic editor. By building a platform that allows users to remix mosaics created by other users, some very interesting mosaic patterns and designs may arise. However, certain problems exist when we let users take control over the content that’s available on the page, for example the lack of a moderator for the contents being published directly onto the server. This is a problem that needs to be addressed when planning out the implementation of such a feature. A possible solution would be to implement the sharing of mosaics in a similar fashion as the studios in Scratch. This way, a curator would take charge of choosing which mosaics are published onto the server, allowing for content control.

### 3.3.2 Snap!

This platform<sup>5</sup> works in a very similar way to the one mentioned in subsection 3.3.1. The main goal remains as re-mixing computer programs developed by other users, or building your own computer program that can later be re-mixed. This computer programs consist in blocks of code that represent a certain action on the interface, by stacking several blocks of code, the user can build a computer program that represents an interesting story.

<sup>5</sup><https://snap.berkeley.edu>

The user must sign in using email, establishing a username and a password in order to publish projects. If a user only wishes to try the platform and create a project, they are still able to without signing up.

Just like Scratch 3.3.1, the main page presents several projects featured on the website, as seen in figure 3.15. This projects were built by other users and are accessible to any user that joins the platform.



Figure 3.15: Snap's main page

Furthermore, they organize the programs already existing by several categories, such as Animations, Music, 3D and the Topic of The Month which changes every month and features every work related to the current monthly topic.

This is a very interesting approach as a programming language learning tool, since it allows users to try other projects and see how they were built in order to learn and possibly in the future include those new capabilities on their own projects.

This tool allows for users to create collections. When creating a collection, it first is private, as in it isn't visible by other users. However, whenever the user wants, they are able to publish that collection.

When a user finishes their project, they are able to save it onto the server or onto

the computer. When saving onto the computer, a Extensible Markup Language (XML) file is downloaded. This file can be imported into Snap!'s editor to continue working on it later. When saving on the server, the user can always go back to the project on his profile and keep working on it from there.

Once again, there is a lack of control of the files being uploaded into the server. The users are free to upload their projects without anyone checking if there's some sort of sensitive content or copy right infractions.

It would be very interesting adopting the private collections feature of this platform and applying it to MosaicoLab's platform for users to save their mosaics while they are not ready to publish them. However the publication of a collection would be difficult, since the creator of a collection was the user, assigning curators would be difficult since the possibility of deleting projects from their collection seems like an appropriation of intellectual property. This could be solved by not allowing to publish collections directly onto the server but only the mosaics in those collections. This way, the curators would be free to choose from the given mosaics the ones better fit to be displayed in public collections.

### 3.3.3 Glitch


This platform differs from the ones presented in subsection 3.3.1 and subsection 3.3.2. This one is intended towards actually writing code instead of using blocks that represent code for the final product. Similarly, the final product is now a website or visual element designed by the user instead of a story composed of sprites and actions.

When signing up, the user has the option of signing in using different methods such as:

- Facebook
- GitHub
- Google
- Email


After signing up and logging in, the user is presented with the dashboard, where the projects developed by them appear. In this same screen, the user can also see the projects they archived to try later.


When moving onto the Discover tab, seen in figure 3.16, several built in projects appear. These projects have the intention of teaching new users to design their screens. Starting with a Basic website all the way to using Databases. As seen in figure 3.17, there are also user built projects. After the user has spent some time on the platform, they are able to start projects from other user projects instead of only the ones offered by default by Glitch's platform.




# There's always something new to make on Glitch

Our Hello apps are built with best practices in mind and sample code you can remix to make your own. Or grab a minimal template and build from a blank slate.

 **Basic Website**  
A simple website starter  
[Hello Website](#)  
[blank version →](#)

 **Hello Node!**  
Full stack, ready to go  
[Hello Node](#)  
[blank version →](#)

 **Glitch in Bio**  
Your own free links page  
[Remix your own](#)  
[learn more →](#)


 **Blog with Eleventy**  
As easy as blogging gets  
[Hello Eleventy](#)  
[minimal 11ty \(by 11ty\) →](#)




Figure 3.16: Glitch's Discover page


## Even more to try

Some of the best from around our community. Find something new to try today!

### Most Viral on Glitch 2021




These apps were *the moment* for many – shared widely across social media, subreddits and blogs!

-  **uk-food**  
specialty british food store
-  **thisolesyadoesnotexist**  
This Olesya Doesn't Exist — GAN generated photos of Olesya
-  **evergiven-everywhere**  
Get this big ship stuck wherever you want

 by Community [View all 7](#)

### From "The Gallery" 2021

A selection of some of the many apps shared by the community in "The Gallery" of the Glitch support forum over the past year!

-  **hex-triplet-words**  
Hex triplets are six-digit codes made up of the characters "0"...
-  **mostcommonjsprojects**
-  **music-visualizer**  
makes music look cool 🎵


 by Community [View all](#)

Figure 3.17: Featured Projects on Glitch

After selecting the project, the user gets a preview of what the end result is supposed to look like and can then, choose to re-mix that project. While the user re-mixes, they are able to change any part of the code they want so that the project better fits what the user is trying to accomplish.

On finishing, the user can download the project, which will download a Zip Format (ZIP) file. The project can also be shared, through a link produced by the platform. When sharing the user gets to pick which users they want invite to edit the code, they can also send a link that allows to edit the code, or a link to see the live site. The link that allows to edit the code only works when sharing to signed in users that have developed enough projects on the platform to get the permission to edit other users projects.

When creating a project and choosing the option to share, the user can choose to keep the project private, however, this functionality depends on payment to the platform. Upon finishing the creation of a project, it is automatically saved onto the server and accessible to anyone on the platform.

### 3.3.4 Possible Features

From the re-mixing platforms, several features and functionalities can be taken into account when designing our platform. Some of which are:

- Scratch 3.3.1
  - Starting projects from a previously built one
  - Curators for public projects
- Snap! 3.3.2
  - Private collections
  - Publish private projects
- Glitch 3.3.3
  - Different options to sign-up or login (Facebook, Google, Github, Email)

When talking about curators for public projects, we can imagine the scenario where a user publishes a mosaic. That mosaic may or may not be containing explicit content, either on the mosaic itself, the title, or the description. Therefore, a curator would be needed to control what mosaics should be publicly displayed or not.

Private collections means the user might want to build their own collection, either consisting of only their mosaics, or consisting of their mosaics and public ones. By creating a private collection, the user is able to combine all the mosaics they want as they please, without needing to concern about whether other people will like it, since it's their own!



By publishing private projects we mean that if a user draws a mosaic and saves it as private(not being allowed to appear on public collections). They will be able to change the mosaic visibility by making it public.



# Chapter 4

## Methodology and Work Plan

In this chapter a presentation of the methodology adopted for the project will be given as well as the work plan in the first semester and for the second semester.

### 4.1 Methodology

For this project, three methodologies were taken into account [Dinnie, 2018], Scrum, Kanban and Waterfall. At the end, a reflection on the three methodologies will occur which will end by justifying the chosen one.

#### 4.1.1 Scrum

The Scrum methodology [Cervone, 2011] was taken into account as it allows meetings with the supervisor to share what was developed from the previous week as well as defining the goal for the upcoming week.

This method is composed of three major components:

- Roles, the role each individual has on the project
- Process
  - Kickoff, creating the backlog (a list of project requirements)
  - Sprint planning, defining the formal outcome for a particular sprint
  - Sprint, concretely developing or building artifacts of the project
  - Daily Scrum, discussion between the team members about what was achieved since the last Scrum
  - Sprint review meeting, where the time frame for the sprint has ended and the team meets to understand what was accomplished this sprint (if the goals were met).

- Artifacts, documents or developed products that allow for a better understanding of what has been accomplished.

The roles in this project would be fairly simple since there are only two people working on it, the supervisor and the student. The supervisor would take on the role of the Product Owner which would analyse the artifacts to get a better understanding of what was achieved. The student on the other hand would take on the role of Project Manager as well as the developer. Since the work needs to be organized, established and developed by the student. Daily scrum meetings would occur, where a reflection about what was achieved that day would be done. At the end of the week a sprint review meeting would take place with the Product Owner present. These meetings serve as guidance and allow for the Product Owner to share their thoughts on what has been done and if what has been done is in line with what was proposed.

### 4.1.2 Waterfall

The Waterfall methodology is a sequential development process that flows like a waterfall through all phases of a project (analysis, design, development, and testing, for example), with each phase completely wrapping up before the next phase begins. [David, 2014]

This means that it follows a very structured development process. This type of methodology is particularly better than most when the following points are proven:

- Don't have ambiguous requirements
- Offer a clear picture of how things will proceed from the outset
- Have clients who seem unlikely to change the scope of the project once it is underway

The usual life cycle of a project that follows this methodology is as presented in figure 4.1.

Several steps can be seen in figure 4.1, the requirements phase is the initial step where the elicitation of requirements takes place. This might happen with meetings with the client to know which functionalities they want, as well as brainstorming ideas and then presenting them to the client. Afterwards, comes the design phase, where software developers design a technical solution to the problems set out by the product requirements. This solutions might be data models, layouts or scenarios. In this step is also where the scope of the project is established.

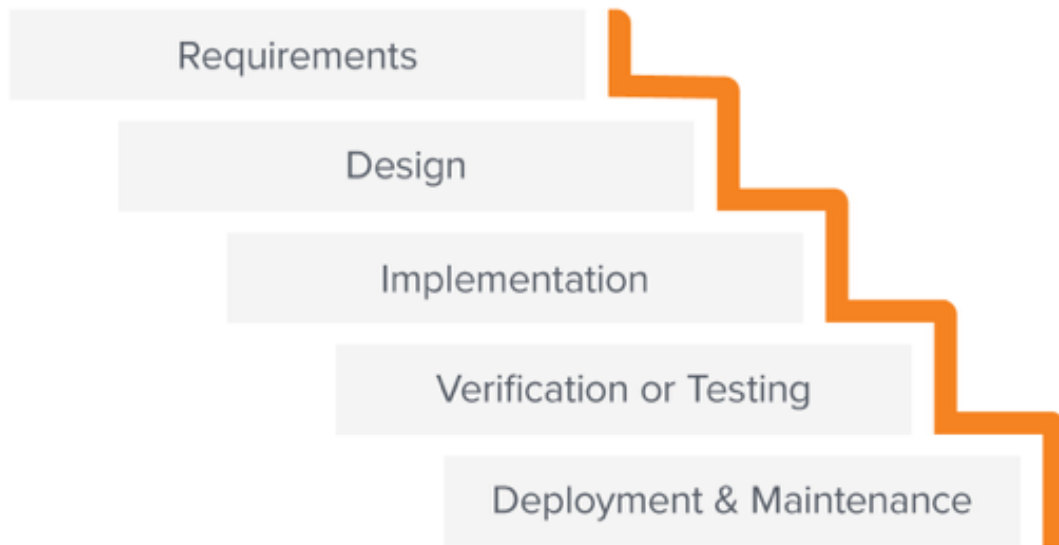


Figure 4.1: Waterfall life cycle

Following this step, the implementation phase begins, this can be a relatively short phase depending on the complexity of the requirements. Most of the searching and analysing other tools steps have been completed, therefore, all that's left is to actually implement the functionalities. If new requirements come up during this step, often, the team will go back to the design phase where they will study other tools or technologies that might be used. Testing takes place during implementation as to make sure the functionality implemented works, but also directly after all the implementation process is finished. The entire platform will get evaluated and testers will try to find errors on the platform. If they actually find any errors, the team will go back into the implementation phase to correct them. When all the errors are corrected, the product is finished and it can be deployed and maintained as well as presented to the client.

This methodology has some disadvantages such as:

- Projects can take longer to deliver with this chronological approach than with an iterative one, such as the Agile method
- Clients often don't fully know what they want at the front end, opening the door to requests for changes and new features later in the process when they're harder to accommodate
- Clients are not involved in the design and implementation stages
- Deadline creep—when one phase in the process is delayed, all the other phases are delayed

Common roles in Waterfall projects are:

- Developer - Person that is responsible of implementing the requirements

- Tester - Employee that is in charge of making sure the requirements are completed in a correct way, by using the platform created by the developers
- Business Analyst - Employee whose role is to make the product visible and popular in digital market
- Project Manager - Person in charge of subdividing the project into smaller categories and distributing tasks between the other team members

[Sergeev, 2016]

### 4.1.3 Kanban

In this methodology [Ahmad et al., 2013], the goal is to be able to visualize the workflow, which can be achieved with Kanban tables. These tables are divided into at least three columns:

- To Do, which contains all the tasks that are still waiting to be started
- Doing, which contains all tasks that are currently being done
- Done, which contains all tasks that have been completed

Using this table allows for a better understanding of the work that has to be done, the work that's being done and the work that is already finished. By having a visual tool such as a Kanban table, the workflow becomes easier and more effective, since the work will be done by objectives.

This method offers values such as:

- Transparency – sharing information openly using clear and straightforward language improves the flow of business value.
- Balance – different aspects, viewpoints, and capabilities must be balanced in order to achieve effectiveness.
- Collaboration – Kanban was created to improve the way people work together.
- Customer Focus – Kanban systems aim to optimize the flow of value to customers that are external from the system but may be internal or external to the organization in which the system exists.
- Flow – Work is a continuous or episodic flow of value.
- Leadership – Leadership (the ability to inspire others to act via example, words, and reflection) is needed at all levels in order to realize continuous improvement and deliver value.
- Understanding – Individual and organizational self-knowledge of the starting point is necessary to move forward and improve.

- Agreement – Everyone involved with a system are committed to improvement and agree to jointly move toward goals while respecting and accommodating differences of opinion and approach.
- Respect – Value, understand, and show consideration for people.

This aspects make this method the more reliable when working with smaller teams since working by objectives can be more realistic than working by sprints.

#### **4.1.4 Reflection**

When taking into consideration the several methodologies presented, some points can be made about each one. The Scrum methodology takes into consideration several roles which would be complex to manage since the number of people working on this project is very little (2). Furthermore, the concept of working according to sprints and deadlines, implementing small features at a time doesn't seem like a reliable method in this project, taking into consideration that there are several aspects to it.

The Kanban methodology could be used not as much as a methodology but mostly as a tool to keep track of the improvements and work that is done, being done and to be done. Therefore, the overall life cycle of the project would not be clear as this methodology takes into account the tasks that need to be done, and not so much the overall structure and workflow that needs to happen to complete the tasks.

The Waterfall methodology is a good combination of the previously mentioned methodologies as it allows for a better life cycle structure and definition and, also allows for a work flow focused on completing tasks and concrete goals instead of focusing on sprints and short time frames. This way the developer can take on the role of tester as well and make sure everything is turning out properly as the project advances.

#### **4.1.5 Applying this methodology**

This methodology was applied by early on in the project, correctly define the requirements, this step took some time as several iterations were needed to come up with the final results. As new requirements were added, new studies were conducted on platforms from where it was possible to understand how the functionalities should work. A database model was also drawn out as soon as the requirements and the scope of the project were well defined and drawn out as to have a good understanding of what was supposed to be done and how.

The overall life cycle of this project can be seen in figure 4.2.

Even though this life cycle may appear rigid, it does allow for changes as in the end of every iteration, the results are analyzed and if they do not conform, the developer will take a step back to further improve on what needs to be improved.

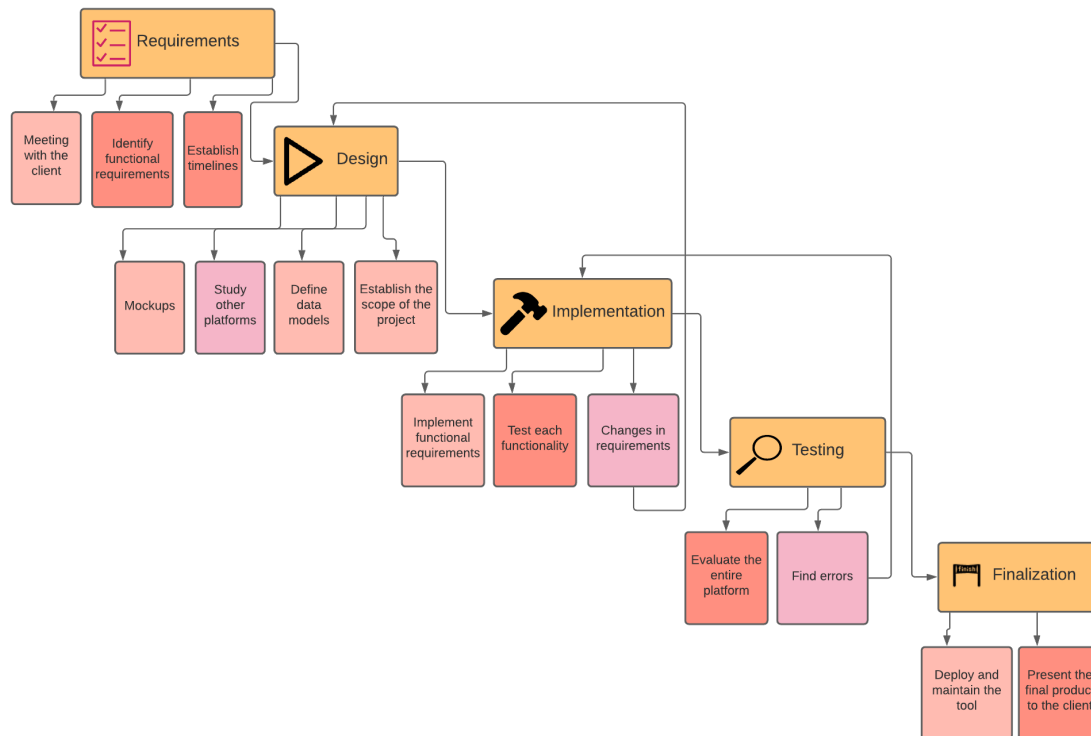


Figure 4.2: Life cycle of the project using the waterfall methodology

Taking into consideration that there is a time limit to be accomplished, it can be said that if a given iteration starts taking more time that was planned, it can be dropped as to allow for the other iterations to be taken care off. Once the iterations that should be taken care of in a certain time frame are completed, then, the developer can go back to the iteration where they were stuck.

## 4.2 Work Plan

In this section an explanation of the work flow followed so far will be given. There will also be an overview of the processes involving developing this project.

### 4.2.1 First Semester

While designing a work plan to develop this project, having a methodology in mind helped with depicting the objectives. Firstly several steps needed to be taken in order to accomplish the final product. Therefore, these steps were established and modified, ending on a final list consisting of:

- Writing the document
  - Establish structure of the document
  - Write scratch text for each section of the document



- Iterations of writing the document
- Studying the current platform
  - Analysing the behavior of the tool
  - Analysing the existing code base
- Studying other tools
  - Mosaic Tools
  - Re-mixing platforms
  - Drawing Software
- Meetings
  - With the representative of MosaicoLab and the author of the Mosaic Editor
- Establishing functionalities
  - Requirements from other tools
  - Requirements from meetings
- Mockups and scratch architecture
  - Establishing the design of the platform
  - Drawing the database
- Implementation
  - First feature implementation

This steps represent the backbone of the project, they are the points that are needed to cover in order to accomplish the project correctly. As seen in figure 4.3, the goals appear on the left, on the right, the time-span for each task is visible.

As seen in figure 4.3, writing the thesis document was a task that started in the beginning of the semester and kept being done until the very end. While writing this document, several other tasks were being completed in parallel.

At first, the objective was to define the "skeleton" of this document beginning with the chapters. After that was done, the document could begin to be filled with the necessary information. From the first meeting, a draft text was written on every chapter as to have a better comprehension of what was expected to be written. Through several iterations, the drafts became more refined and overall correct until the final version of each chapter was achieved.

In order to keep up with the time frame established. Several goals were kept in check as a way to keep track of what still needs to be done. Through weekly meetings with the project's supervisor, several sections began to be written in the final form, which is the way they are written in this document, which allowed for checkpoints to be achieved and being closer to finishing this document.

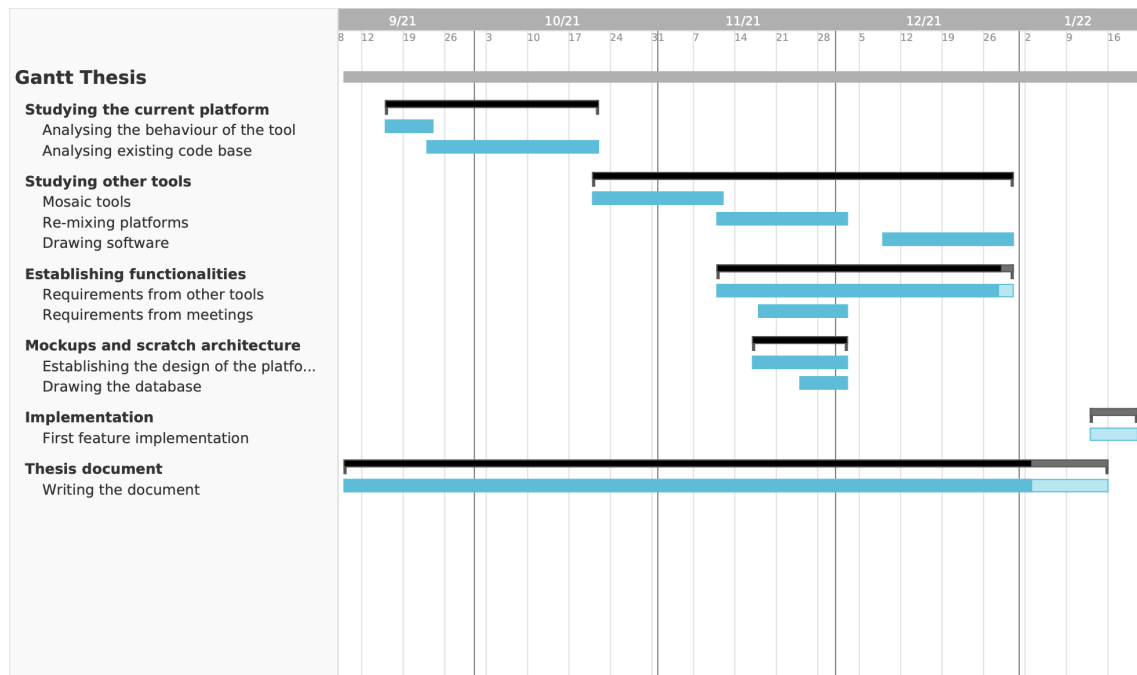


Figure 4.3: Gantt diagram of this project

Starting with the Introduction (section 1.5, section 1.2, section 1.4 and section 1.3). Several iterations of writing and rewriting the document occurred in order to accomplish the final version of each section. The objective of developing this project needed to be well defined as to correctly understand the goal and what can be considered the success criteria.

Following the writing of the Introduction chapter, a study of the current platform took place. This task can be divided into two different steps. The first one being correctly comprehend the current platform's functionalities, how it works and how it can be used. Secondly, a deeper knowledge of the platform had to be acquired as a way to prepare for implementing the functionalities onto the existing tool. This step took much longer than the previous, since it was needed to analyse the architecture of the code base, understand the existing classes and methods and how they interact with each other. Similarly, writing the Background section 2 took sometime. Several iterations occurred as sometimes too much detail was given which could become confusing to readers.

Afterwards, the chapter that needed to be closed was the State of the Art 3. The State of the Art chapter involved research which was done from the first draft and then continuously done throughout the development of this document. Some of the presented tools were discussed previously with the supervisor and other tools were found and suggested to the supervisor. This tool took several iterations to be correctly completed. Without focusing on irrelevant aspects of these platforms, but keeping true to a good analysis of the platforms, the text was written and re-written as to accomplish the ideal structure and contents.

On November 13th, a meeting with the representative of MosaicoLab took place. This meeting had the objective of presenting the features found to be interesting to apply to the editor, as well as receiving feedback from the representative as

to what their vision was of the editor and the platform. This allowed for several requirements to be drawn out as well as getting a clearer picture of what the project would end up looking like.

Using the State of the Art section 3 and the requirements listed by the Mosaico-Lab’s representative, some low level wireframes were done in collaboration with the supervisor. The database was also drawn out and the requirements list was much more composed and structured.

From here on out, the work done had the objective of completing the remaining chapters as well as implementing the functionalities.

## 4.2.2 Second Semester

In the second semester it is expected to fix whatever problems may arise on the document, as well as implement the project. In this stage, a more detailed work plan is needed.

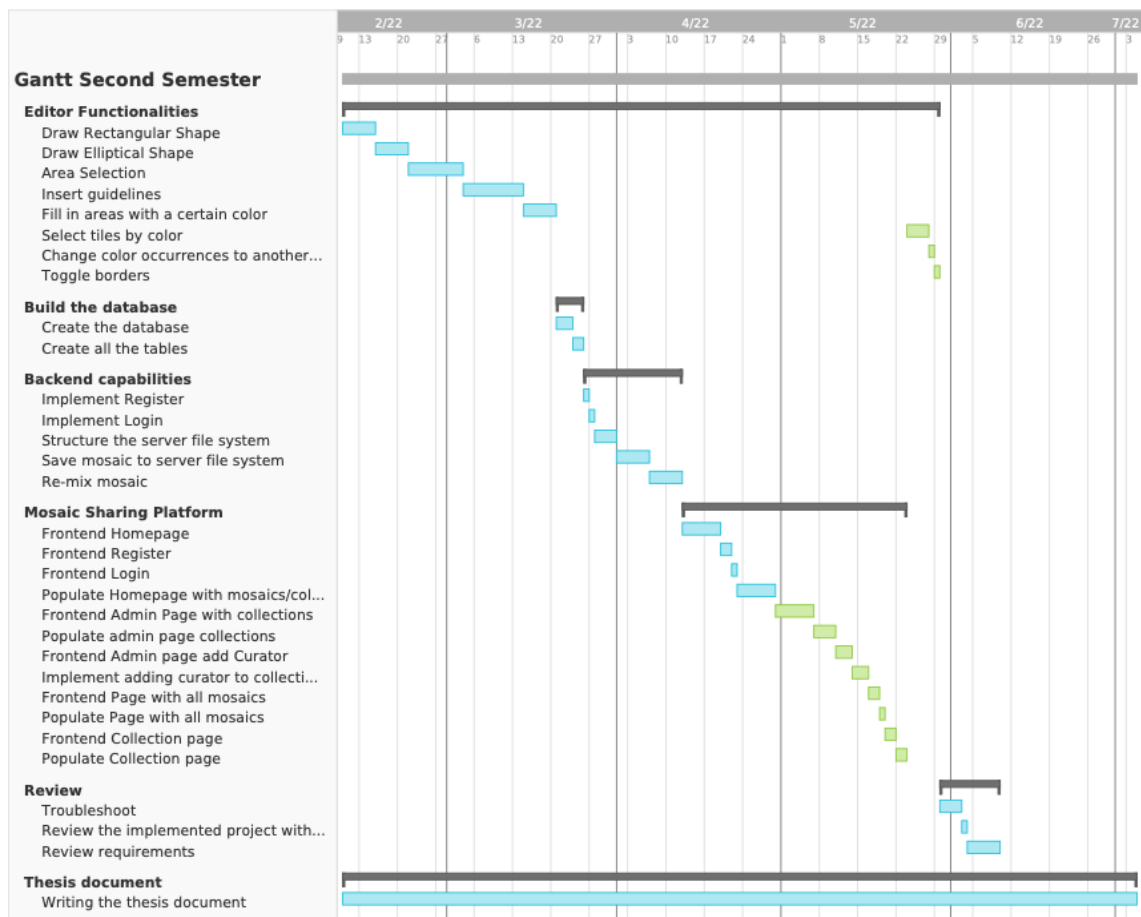


Figure 4.4: Work Plan for the 2nd Semester

As seen in figure 4.4, the work is divided in the several stages of the project(the mosaic editor tool, the backend capabilities and the mosaic sharing platform). In the picture, several rectangles representing the estimated time spent on each task are seen. These rectangles have two colors representing different things:

- Blue - It's a Must implement
- Green - It's a Should or Could implement

The presented diagram is somewhat optimistic as it points out that the project will be complete before the due date with almost a month to spare. It is important to notice that this are estimates and not definite times. More than likely, there will be a learning curve when adding features to the currently existing mosaic editor, meaning that the first features will probably take more time than expected. The same can be said about implementing the frontend pages of the sharing platform and the communication between the backend and the frontend.

The different colors to represent the different priority requirements were used since it may not be feasible to expect all the requirements to be implemented before the deadline. Using different colors, we can at least see which requirements will take priority and must be completed. This does not exclude from the fact that if the planning is accurate, all the requirements will be completed regardless of their priority, but at least allows for some room when discussing which must be implemented and which could or should.

After the Must features are implemented, the chosen features to implement next were the mosaic sharing platform lower priority features as those seemed to be more important for the project. After those functionalities are completed, the next step would be to implement other features of the editor itself.

There will also be a review phase where the overall platform will be evaluated and a meeting with the client will take place to present what has been done and see if it is aligned with what was expected.

One activity that will continuously happen throughout the second semester is the writing of the thesis document. The work flow will be stated as well as the choices taken throughout the implementation of the project. There will also be an analysis on the behaviour of the platform and the objectives achieved.

The time attributed to each task presented in the Gantt diagram was estimated by the developer. Taking into account the requirements, the technologies and how experienced they may or may not be with the languages to be used, rough estimates were given for each task.

Tasks related to presenting information and writing information to the database were seen as more attainable in a short period of time than tasks related to drawing tools. Therefore, the tasks related to the Mosaic Sharing Platform should take less time individually to implement correctly. While the mosaic editor tasks are related with a more logical and problem solving aspect of programming that will take more time to design a solution and implement it correctly.

### 4.2.3 Achieved Planning

After the intermediate defense, the mosaic editor was a platform that was studied enough to start adding some functionalities, however, it seemed more im-

portant to start creating the Mosaic Sharing Platform which is the central point of this project. Therefore, instead of continuing working on the mosaic editor as planned, the second semester started with the creation of the new platform.

As we can see in the figure 4.5, the requirements related to the Mosaic Sharing Platform switched with the requirements of the editor in terms of chronological order. Some other changes occurred in the planning such as the implementation of the client-side functionalities taking place at the same time as the server-side functionality (e.g Register and Login). Furthermore, a testing phase occurred, both for functional tests and usability testing. Requirements with priority "Should" related to the Mosaic Sharing Platform were completed before implementing the "Must" priority requirements related to the Mosaic Editor. This is due to the fact that leaving the platform in a state where only the highest priority requirements were implemented would lead to a counter-intuitive user experience and a low quality project. Therefore, dedicating more time to the new platform from the beginning would make more sense instead of going back and fourth between the Mosaic Sharing Platform and the Mosaic Editor.

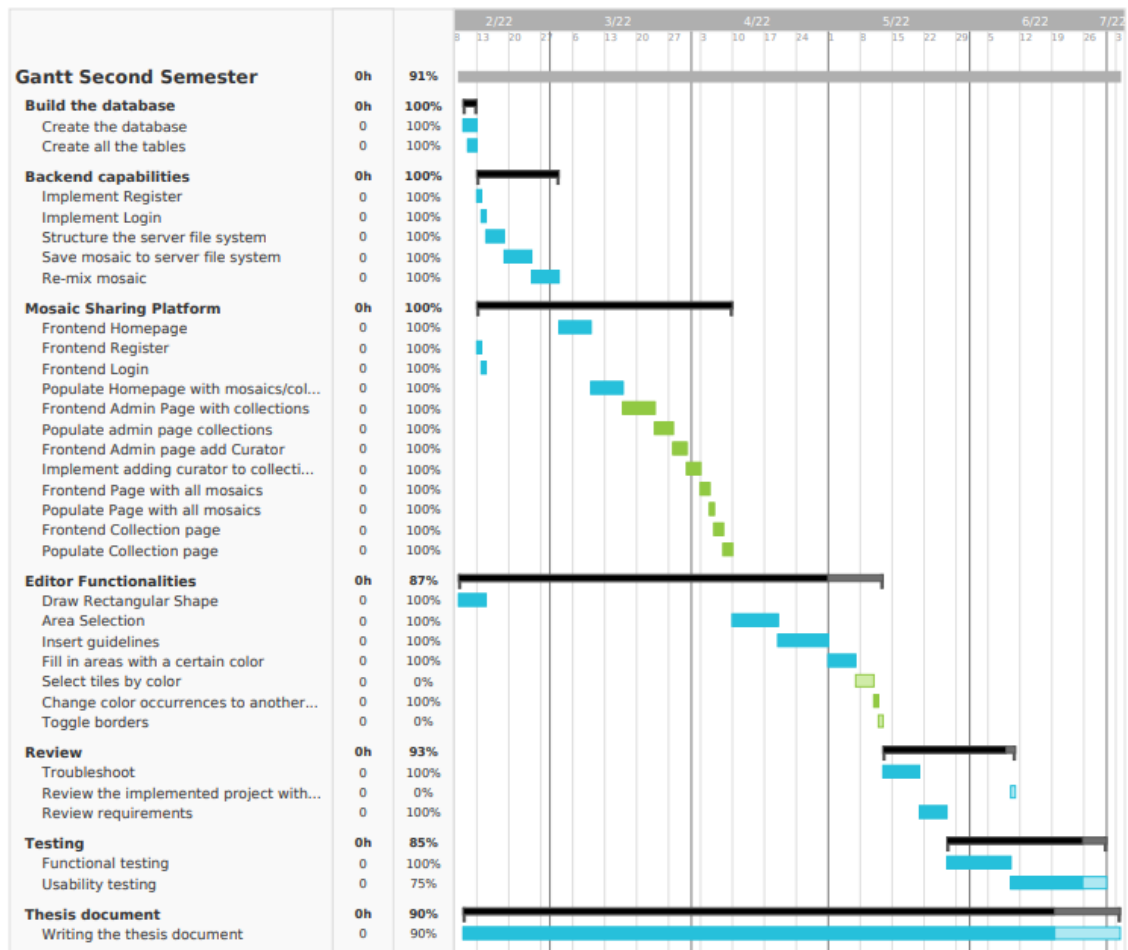


Figure 4.5: New planning for the second semester

## 4.2.4 Risk Analysis

When it comes to developing a new platform, several risks exist that might delay or prevent the implementation of functionalities. Therefore, a risk analysis to comprehend those risks and establish a mitigation protocol need to take place.

Some of the risks associated with this project are related to the functionalities to be added to the mosaic editor tool, and other risks are associated with the platform. As a visual method, a risk matrix was developed as to provide a better understanding of the probability and impact associated with each risk, that matrix can be seen in figure 4.6.

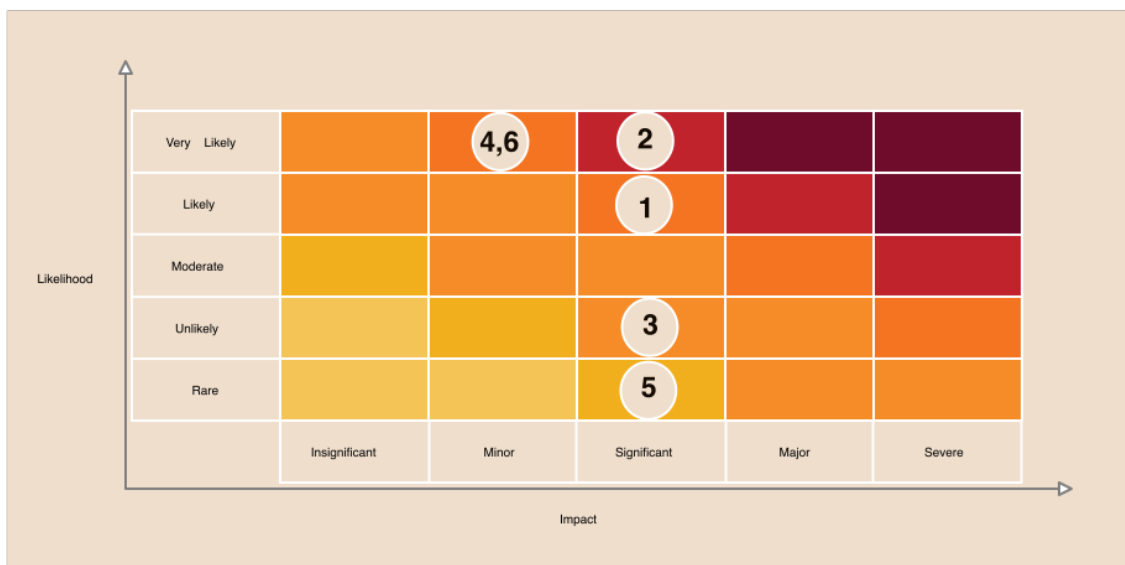


Figure 4.6: Risk Matrix of the risks associated with the project

To comprehend the matrix, the risks presented with numbers are defined as:

- 1 - Conflicting schedules with University classes that may delay the implementation process
- 2 - Working on an already existing project and code base
- 3 - Dependency problems when deploying the tool
- 4 - Delay associated with developing a new platform using pure JavaScript with no framework
- 5 - Lack of communication leading to functionalities being implemented differently to what was imagined
- 6 - Delay associated with learning how to implement certain functionalities

### Risk 1 - Conflicting schedules

Since this project takes place in the context of Master degree dissertation, there would always be the risk of overlapping with classes of the university. This risk

is very likely to occur since classes will take place in the second semester. However, this can be dealt with a proper project work plan designed and a structured guideline as to how to implement this project.

### **Risk 2 - Delay associated with working on an existing project**

This risk relates to the fact that the mosaic editor is an already existing tool to draw mosaics. Working on a project that has been developed by another developer and having to add features and change code can be very intimidating. It is very important to have a clear idea of where the code needs to go, where to insert new code and how to make everything work together. The code base for the existing mosaic editor is very extensive and therefore, learning every aspect of this project will take considerable time. Even though this project has been studied during the first semester, getting hands on and writing code is a different experience which may delay the implementation of the project. This risk is very likely to occur since the project is an already existing project and not a brand new project where the developer will be working from scratch and therefore, the aforementioned problems are relevant. The impact is also significant since if it takes too much time to get used to the project and the code base and understanding where the code needed is supposed to be written, it could lead to the project not being finished on schedule.

### **Risk 3 - Dependencies**

This relates to the servers capacity. However, in this case, it is easily stopped by not using frameworks and tools that might cause dependencies. Using the JavaScript, CSS and HTML languages, the work can be accomplished with no need to create dependencies within the project. Therefore, the probability is lower.

### **Risk 4 - Delay associated with not using frameworks**

As explained in Risk 3, in order to prevent dependencies, the work will require the use of the tools without frameworks. This can lead to a delay since frameworks are created as a tool to spare time on the developer side. This means that using frameworks, some functions and aspects of the project wouldn't need much attention from the developer in the first place. When not using frameworks, the developer must be conscious of all possible functions that aren't present from the frameworks as well as errors that may appear along the way. This a minor risk since it would only take more attention and perseverance, however it is very likely to happen.

### **Risk 5 - Lack of communication**

This risk relates to personal interpretations of how each feature of the platform and tool are supposed to work. It is directly influenced by poor communication between the developer and the supervisor as well as the MosaicoLab's representative. The probability on this risk is very low (Rare) since the work will constantly be shared between the developer and the supervisor as to always be certain of what is being implemented and if it is being implemented correctly. Therefore, this risk might never occur while developing. The only chances of this risk taking place are before starting the implementation.

### **Risk 6 - Delay associated with learning curve**

Similarly to Risk 4, this risk is associated with the learning curve of the JavaScript language when it comes to specific functionalities. This risk exists since some of the proposed features might be very complex to implement, however, with research and help from the supervisor when needed, they can be easily surpassed. Therefore the risk is low but the likelihood is high.

## **4.2.5 Materialized Risks**

During the development of this project, some of the previously mentioned risks manifested. Namely, the second risk presented in section 4.2.4. Working in an already existing code base proved to be challenging when starting the implementation process in this project. The first tool added to this project took 2 extra days to be completed. However, the process for the next tools was similar, therefore, the implementation time for the other tools balanced out the extra time spent on the first.

The other risk that materialized was the sixth risk 4.2.4. This risk, relates to the learning curve with the JavaScript language. Even though the developer had already used this language, they weren't proficient. In the implementation of the feature that allows the user to save the mosaics with a request sent to the Mosaic Sharing Platform, much more time was spent than what was planned, between learning how to use the *fetch* Application Programming Interface (API) and establishing all the right parameters to using JavaScript, to read the information placed on the forms provided, as well as finding out where to place the code that would allow this new feature to make sense.

Between these two risks, less than 1 week was added to the development time. The impact of the risks was somewhat reduced since the first risk allowed to save time on the next tools to be implemented and the second risk occurred early into the implementation, where with help from the supervisor as well as having studied how to use the functions needed allowed for the time spent developing this feature to be reduced.



#### **4.2.6 Threshold of Success**

The goal of this project is very clearly defined as well as the requirements. With this in mind, the threshold of success should be to implement all the high priority (Must) functional requirements for the Mosaic Sharing Platform.

For the mosaic editor, even though it is expected to add new features, it is more important to establish a new architectural model that allows the creation of these new features to be made effortlessly, while implementing at least three new features that make use of this new architectural model.



# Chapter 5

## Elicit Requirements

In this chapter an overview of the requirements will be given. Through meetings with the representative of MosaicoLab's as well as the original MosaicEditor tool author, several requirements surfaced. Some of these requirements were already established in the proposition for the internship, however, some other requirements were added to the already existing list. In order to better understand these requirements, an overall picture of the end product will be given.

### 5.1 Mosaic Sharing Platform

From the several meetings with the original MosaicEditor tool author and MosaicoLab's representative. The overall idea of what was planned to implement became clearer.

Firstly, this will be a platform where users can see other users mosaics and work on top of those mosaics. The public mosaics would be organized by collections. A collection is a group of mosaics that follow a common theme. They can also be mosaics created under the same circumstances. For example, if a workshop would occur in a school, the mosaics drawn, would be organized in a collection that would only contain mosaics from that event.

To make sure that the public mosaics don't contain any sensitive content, it would be useful to have a curator. Someone whose role would be to perform an evaluation on the contents of the mosaic to make sure it can become public. In the particular case of the school workshop, these role could be attributed to the teacher for example.

In other contexts such as regular users that are just creating mosaics for fun and publishing them. Another person would have to take on the role of curating those mosaics to make sure they can be publicly shown. In order to help organizing these aspects, the consensus came to be that a curator should be attributed to a collection. These way, there is always someone in charge of evaluating the mosaics and their contents.

Collections can have a description and a name specific to that particular collection. Only the administrator of the website would be able to create collections and therefore, only they are able to attribute the curator role to users. When the user clicks on a collection, they will be redirected to a page where the information about that collection is presented, such as the name, the description and all the mosaics inside that collection.

The user could also be able to create their own private collections, where they arrange their mosaics according to whatever parameter they wish. These private collections can be shared by the user that created them. However they are not public, as in, they will not appear in the home page. The user can choose mosaics from their private collection to share but not the entire collection.

## 5.2 Requirements

In these meetings we divided the work that had to be done into several categories:

- The overall platform
- The MosaicEditor tool
- The backend portion of the project

The functional requirements established for the whole project are as presented in table 5.1. The table consists in 4 columns, the first one being theme which is the area where these features will be inserted. Then we have the requirement itself which contains a brief explanation of the requirement, subsequently the role is presented where the entity that is able to perform the activity described in the requirement is defined. Lastly there is a Priority column.

The priority column has different tags relating to different degrees of priority:

- (M) - Must - Requirement that must be present by the end of the project
- (S) - Should - Requirement that should be present by the end of the project
- (C) - Could - Requirement that could be implemented
- (W) - Won't - Requirement that will not be developed

Theme	Requirement	Role	Priority
Platform	Login and Register Interface	All	(M)
	Login with Google	All	(C)
	Login with Facebook	All	(C)
	Login	All	(M)
	Register	All	(M)
	Save Mosaics to Server File System	All	(M)
	Attribute title to mosaic	All	(M)
	Share Mosaics	All	(M)
	Admin Page to create collections	Admin	(S)
	Create collection	Admin	(S)
	Attribute description to collection	Admin	(S)
	Admin Page to add curators to collections	Admin	(S)
	Attribute roles	Admin	(S)
	Interface with all public mosaics	Admin/Curator	(S)
	Collection Interface	All	(S)
	Add mosaic to collection	Curator/Admin	(S)
	Private collection page	All	(C)
	Create private collection	All	(C)
	Interface with user's private collections	All	(C)
	Homepage with collections	All	(S)
Suggest mosaics to add to a collection	Curator/Admin	(C)	
Open previously built mosaic	All	(M)	
Editor	Draw Rectangular Shape	All	(M)
	Draw Elliptic Shape	All	(M)
	Select tiles by color	All	(S)
	Change color occurrences to another color	All	(S)
	Rectangular Area selection	All	(M)
	Free hand Area selection	All	(W)
	Fill in areas with a certain color	All	(M)
	Insert horizontal guideline	All	(M)
	Insert vertical guideline	All	(M)
	Toggle Borders	All	(C)
	Generate mosaic from image	All	(W)
	Tesselate (from TileMaker 3.1.3)	All	(C)
	Notate on tiles/group of tiles	All	(C)

Table 5.1: MoSCoW table for priority of different requirements

From the presented requirements, it is possible to make a distinction between the ones that already existed when the project came to be and the ones found and proposed during the project. Therefore, the requirements that existed before the beginning of the project were:

- Registering and logging in
- Saving mosaics on the server
- Sharing mosaics

- Automatic mosaic generation from pictures with the possibility of applying different techniques
- Cropping
- Exporting capability (to Scalable Vector Graphics (SVG), PNG)
- Different painting tools (e.g., line, shape drawing, coloring tools)
- Use of guide lines to aid in the drawing
- Different tile shapes

After the project began, several tools were studied and meetings with the client occurred which allowed for new requirements to appear on the project. These requirements are:

- Login with Google
- Login with Facebook
- Attribute title to mosaic
- Admin Page to create collections
- Create collection
- Attribute description to collection
- Admin page to add curators to collections
- Attribute roles
- Interface with all public mosaics
- Collection interface
- Add mosaic to collection
- Private collection page
- Create private collection
- Interface with user's private collections
- Homepage with collections
- Suggest mosaics to add to a collection
- Allow user to change the language of the platform
- Rectangular area selection
- Free hand area selection

- Select tiles by color
- Change color occurrences to another color
- Tessellate
- Notate on tiles/group of tiles

Given the amount of requirements found during the first semester and the complexity of each one, through meetings with the clients, the priority of each of these requirements was attributed and some of the original requirements were abandoned. This allowed, however, to the possibility of building a more engaging and interactive platform than what was planned initially.

### **5.3 Backend**

In the backend portion of this project, several aspects need to be established. These are, the database, the relationships between tables and the server file system to store the project files.

The database will have to contain entities that regulate the information being presented. Therefore, several roles need to be created, namely:

- Admin
- Curator
- Regular user

There will also be present a mosaic entity that will contain details such as the title, the creator and a reference to the file stored in the server file system. Furthermore, collections will also be an entity on the database that will contain the mosaics that are part of that collection, a description of the collection and the Curator associated with it.

In this portion of the project, the handling of logging in and registering will take place, as well as:

- Saving mosaics to Server File System
- Attribute title to mosaic
- Create a collection
- Attribute roles
- Attribute description to collection
- Create private collection

- Open previously built mosaic
- Suggest mosaics to add to a collection
- Add mosaic to collection

These requirements directly deal with getting information from the database or the server file system as well as creating new object instances on the database.

## 5.4 The Mosaic Editor tool

For the Mosaic Editor tool we are looking for new features to add, the ones that have been established are:

- Draw Rectangular shape
- Draw Eliptic shape
- Select tiles by color
- Change color occurrences to another color
- Rectangular area selection
- Fill in areas with a certain color
- Insert horizontal guideline
- Insert vertical guideline
- Toggle borders
- tileate (from TileMaker 3.1.3)
- Notate on tiles/group of tiles

These features were discovered through the analyses performed on the State of the Art chapter 3. Where several tools were studied and features were noted to be discussed with MosaicoLab's representative as well as the MosaicEditor author. Some other features were already established on the objective of this project.

With the Selection tools, the user can change the color of several tiles, independent of location and original color into one that they pick. Using the Fill in areas with a certain color, the user can choose an area that has the same color and select a different color to change them. This means that the area filled will be the same size of the original area, only a different color. On the topic of changing the color of tiles, the user is also able to change color occurrences to another color. This means that the user can pick a color present in their mosaic and change all instances of that color into a new one.



The addition of these tools will allow the users to save time when painting their mosaics by using tools with which they are familiar with. Even though the creative process will still be the same, by helping the user work faster, they will have more time to think about what to do next on that mosaic which could lead to very interesting projects appearing. The editor in itself can help the user come up with ideas, by saving the user's time when performing tasks that can be performed quicker than if the new tools weren't there.

## 5.5 The Overall Platform

From the requirements pointed out in the Backend section 5.3, some issues arose regarding how these capabilities will be seen by the user. From meetings with the representative of MosaicoLab and the author for the MosaicEditor, the agreed platform would have a main page where the collections would be presented to the user. There would also be present a way of redirecting the user to the Mosaic Editor tool in a form of a button. Furthermore, since several roles exist for the user, different interfaces would be needed.

The tool will contain several common interfaces that can be accessed by all entities regardless of roles, such as:

- Login and Register Interface
- Homepage with collections
- Interface with user's private collections
- Collection Interface

Associated with these interfaces, several requirements will be available for all kind of users on the platform, namely:

- Login
- Register
- Share Mosaics
- Create private collection
- Open previously built mosaic

The platform will also come into play when it comes to the creative process of the users. Since a lot of mosaics will be public, the users can get ideas from already existing mosaics and changed them in a way that pleases and satisfies them creatively.

### 5.5.1 Admin

The users with Admin role will have the capability of assigning the curator role as well as the ability to create collections. Therefore, a separate page will be needed, in this page, the admin gets the option of creating a new collection, naming that collection, assigning a description and possibly loading some mosaics into the collection. When those processes are finished, the admin will then have the task of assigning the role of curator to that particular collection. Hence, a list of all the users will be presented, from where the admin can choose who gets that role.

Apart from this capabilities, the admin will have the same permissions as the curator.

Associated with these capabilities are the following interfaces and functionalities:

- Interfaces
  - Admin Page to create collections
  - Admin Page to add curators to collections
  - Interface with all public mosaics
  - Interface specific to a collection
- Functionalities
  - Create collection
  - Attribute description to collection
  - Attribute roles
  - Add mosaic to collection

This functionalities and interfaces are specific to the admin user except the interface with all public mosaics as well as the functionality to add mosaic to a collection which are common between the admin and the curator. The admin is the user with highest permissions since they are able to directly change elements on the database.

### 5.5.2 Curator

The Curators will have the responsibility of filtering which mosaics are accepted into the collections they are assigned to. These users will need a separate page where all the mosaics available on the server will be presented. From this screen, the curators will then be able to choose the ones they wish to add onto their collection(s). This role is very important, as it is the only way of moderating content that is presented on the main page for all users. The curators have the responsibility of not accepting mosaics that might contain sensitive content.

When it comes to interfaces associated with this role, only one exists that isn't shared with the regular user, that is the Interface with all public mosaics. From

this interface, the curator will be able to choose mosaics to add to a collection they are curating. Therefore, they also have the functionality of adding a mosaic to a collection.

This role also has some access to the database since the curators are able to change elements in the collection's table.

### **5.5.3 User**

The user, will have the responsibility of drawing mosaics, and populating the server files with mosaics from where the curators can choose and exhibit on the collections. The user will be able to create private collections, which will contain mosaics that are not supposed to be public. These collections can be shared via Uniform Resource Locator (URL) with other users or other people that are not even registered on the platform. If the user becomes comfortable with the idea of publishing the mosaics designed that only exist on their private collections, they will be able to choose them to publish them. Making those mosaics available for the Curators to add to their assigned collections.

## **5.6 Quality attributes**

Besides the functional requirements previously exposed, it is important to define certain aspects of the platform. Since the platform to be implemented is supposed to be used by very different types of people, perhaps by the first time during the workshops carried by MosaicoLab. It is important that the platform is easy to use. This refers to the quality attribute of usability. Furthermore, it is important that in the future, developers are able to maintain the platform as well as add new functionalities when needed. This aspect relates to the maintainability quality attribute.

### **5.6.1 Usability**

Usability is defined as "a quality attribute that assesses how easy user interfaces are to use. The word "usability" also refers to methods for improving ease-of-use during the design process." [Nielsen, 2012]. This means that the platform should not only be easy to use but also quick to pick up from the first time using it. As mentioned before, the audience for this platform will be very wide, not only the people who will use it during workshops but also other users that may find out about the platform. As so, being extremely demanding to personally provide assistance to all the users that come across this platform, it is important that they are able to pick up on how to use the platform as quickly as possible. Therefore, the scenario for this attribute relates to how quickly a new user can perform the tasks needed to accomplish a certain goal. With this goal in mind, the scenario depicted in table 5.2 was developed.

Source	User
Stimulus	Wants to create a new collection and add a mosaic to that collection
Artifact	User interface
Environment	Runtime
Response	User creates a collection and adds a mosaic to that collection
Response metric	Operation time $\leq 1$ minute

Table 5.2: Usability scenario

The table depicts the scenario that a user should not spend more than 1 minute to perform the actions of creating a personal collection and adding a mosaic to that same collection. This measure of time accounts for the user to reach the collection creation page, create a collection and choose a mosaic to add to that collection. This does not account for the time a user can spend thinking of a name and description for the collection.

### 5.6.2 Maintainability

Maintainability is defined as "the degree to which software is understood, repaired, or enhanced" [Novoseltseva, 2020]. Therefore, it should be easy to make changes to software, not only for bug fixing but also to add new features. Since this project relates to a drawing platform as well as a sharing platform, several new features can be implemented to make the drawing experience better, easier as well as the sharing experience wider in terms of how to attract different audiences. This can be made possible by making sure the code is easy to understand, the project is properly structured and the documentation is available. A maintainable software should follow some guidelines:

- Can I find the code that is related to a specific problem or change?
- Can I understand the code?
- Can I explain the rationale behind it to someone else?
- Is it easy to change the code?
- Is it easy for me to determine what I need to change as a consequence?
- Are the number and magnitude of such knock-on changes small?
- Can I quickly verify a change (preferably in isolation)?
- Can I make a change with only a low risk of breaking existing features?
- If I do break something, is it quick and easy to detect and diagnose the problem?

As such, for this project, the goal regarding this quality attribute will be to create a software where a new developer can easily answer the questions presented. Since

this quality attribute can be somewhat subjective in terms of how maintainable is the piece of software being developed, it becomes hard to test, since different developers with different experience levels could run into different problems or understand the code at different paces. Therefore, testing this quality attribute will not be possible, however, by the end of the project, an explanation on how this attribute was accomplished will be given.



# Chapter 6

## Architecture

In this chapter the architecture of the work to be developed will be presented, providing a better understanding of the structure of this project. Furthermore, the technology choices and possible architecture decisions will be explained in greater detail.

As mentioned in previous chapters, this work can be divided into two parts, the Mosaic Sharing Platform and the mosaic editor, we will take a look into both of them separately and then how they communicate with each other and how the integration takes place in this project.

### 6.1 General Architecture

As to understand the usage of this platform, the diagram presented in figure 6.1 was designed. Users will typically begin on the Mosaic Sharing Platform to browse mosaics (either public mosaics or their personal collection). Through the Mosaic Sharing platform the users can access the Mosaic Editor to edit or remix an existing mosaic or start a new one. On the Mosaic Editor, users can save their mosaic through the Mosaic Sharing Platform.

One aspect that needs clarification from figure 6.1 is the fact that mosaic information doesn't include the files that represent the mosaic. These files are an image file in PNG format that is used in the platform to show a preview of the mosaic as well as a JSON file that represents the project and is used when a user remixes an existing mosaic. The mosaic information that is included in the database will be further analysed in section 6.2.2.

The behaviour of each platform will be explained in greater detail in the following sections.

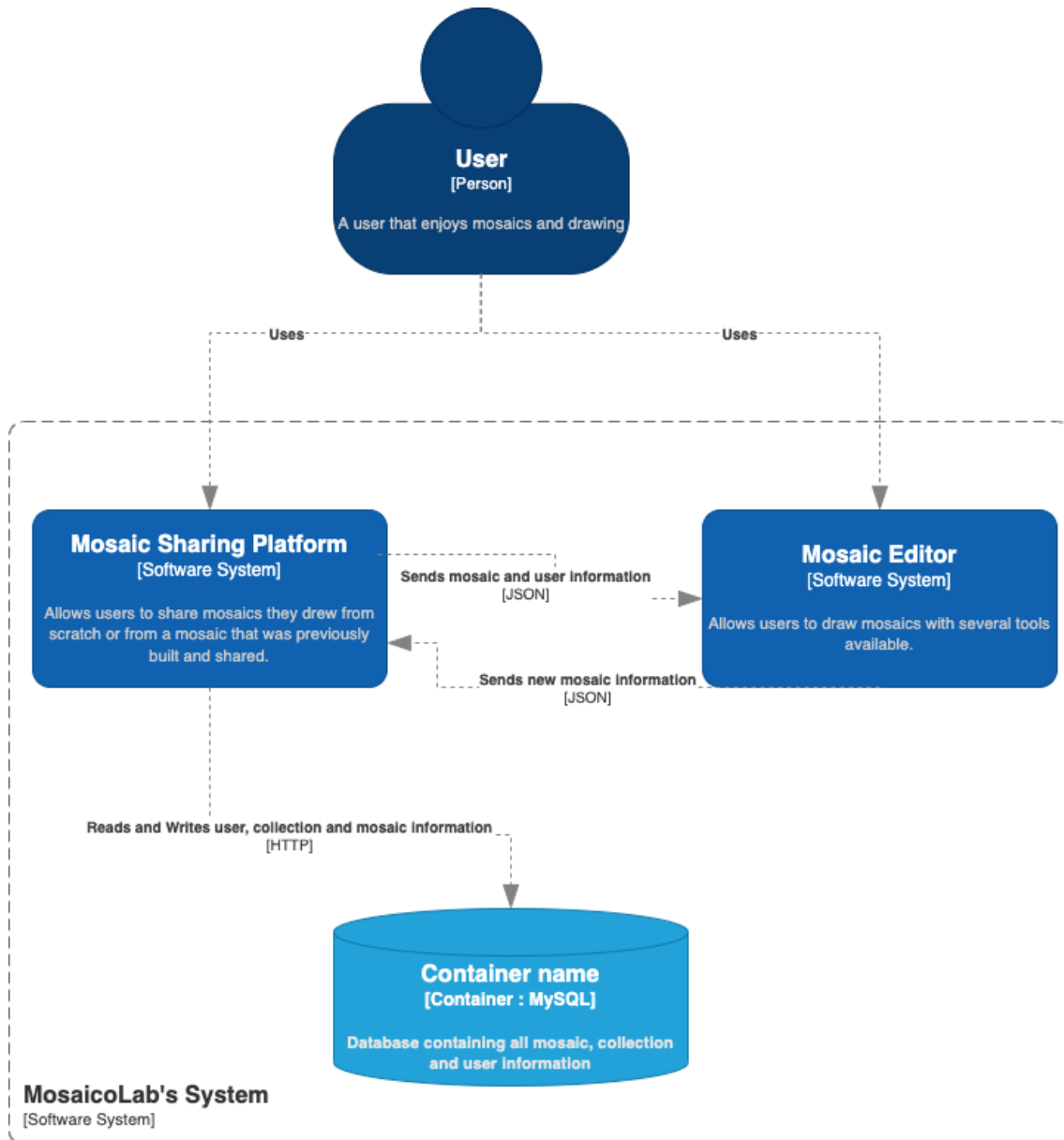


Figure 6.1: Context Diagram representing the basic functionality of the platforms

## 6.2 Mosaic Sharing Platform

Regarding the architecture of the Mosaic Sharing Platform. several decisions were made that need explaining. In this section, an analysis on the technologies used and project structure will be performed.

### 6.2.1 Technologies

For the Mosaic Sharing Platform, the chosen language was PHP as explained in section 1.3. Using PHP: Hypertext Preprocessor (PHP) the integration with the frontend is straightforward, the HTML used for each web page is present in the



PHP files.

For the frontend portion of the project, not only was HTML used to create all the elements needed in each webpage, but CSS was also used to style each of those elements.

In terms of non static content, such as images and JSON files that result from the creation of Mosaics from the users of the platform. Those files are saved on the server, allowing for quick access.

In order to decide where to store the files (between the server file system or the database) it is important to mention the pros and cons about saving the files in the file system instead of the database.

The pros for saving the files in the server instead of the database are, as mentioned in [Singh, 2019]:

- Read and Write operations to a database is slower than a file system
- The database backups become increasingly bigger with the number of files, making restoration and replication slower
- Access to files requires going through the application and database layers, increasing the application memory
- Allows usage of external services for serving the files (e.g. Amazon S3)
- File servers are much cheaper compared to database
- Avoids database overhead with Create, read, update and delete (CRUD) operations
- Implementing the access to server files and showing those files is easier than reading the file information on the database, converting to file and then using the file.

From the list presented above, it is comprehensible that managing files directly from the file system is quicker, does not require as much hardware resources and is overall easier to implement. However, there are still downsides to using the server to store files instead of the database, such as the ones mentioned in [Singh, 2019]:

- Databases provide Atomicity, Consistency, Isolation and Durability (ACID) compliance for each row
- Databases provide data integrity between the file and its metadata
- Database Security is available by default
- Backups automatically include files, no extra management
- File deletion/ updating is always in sync with row operations, no extra maintenance needed

With the list of cons presented, it is clear that the main problems associated with using the server file system instead of the database are the lack of security, the lack of ACID compliance as well as the regular backups. However, the backups would become increasingly larger with the number of files, which would take us back to the pro presented previously regarding the hardware resources needed to maintain those backups.

Taking into account the several pros and cons and taking into consideration that the project must be budget friendly, storing the files in the server was the method considered for this project. This doesn't discard the fact that this project could achieve similar results by storing the files in the database, however, in the long run it might run into maintainability problems due to the size of the database.

### 6.2.2 Database Architecture

As previously mentioned data storage is important and therefore the database needs to be defined. In the context of this project and with help of the information gathered, the database model can be seen as presented in figure 6.2. This model was the result of several iterations of discussion and implementation. Through 3 iterations, the final database model was achieved. This model has two other entities that were not referred in section 5.3 which are the "Suggestions" table and the "Annotations" table. The "Suggestions" table will allow for a direct correlation between mosaics and the collection they are most suited to be a part of and the "Annotations" table will maintain a relationship between a mosaic and its creator annotations on certain tiles or group of tiles.

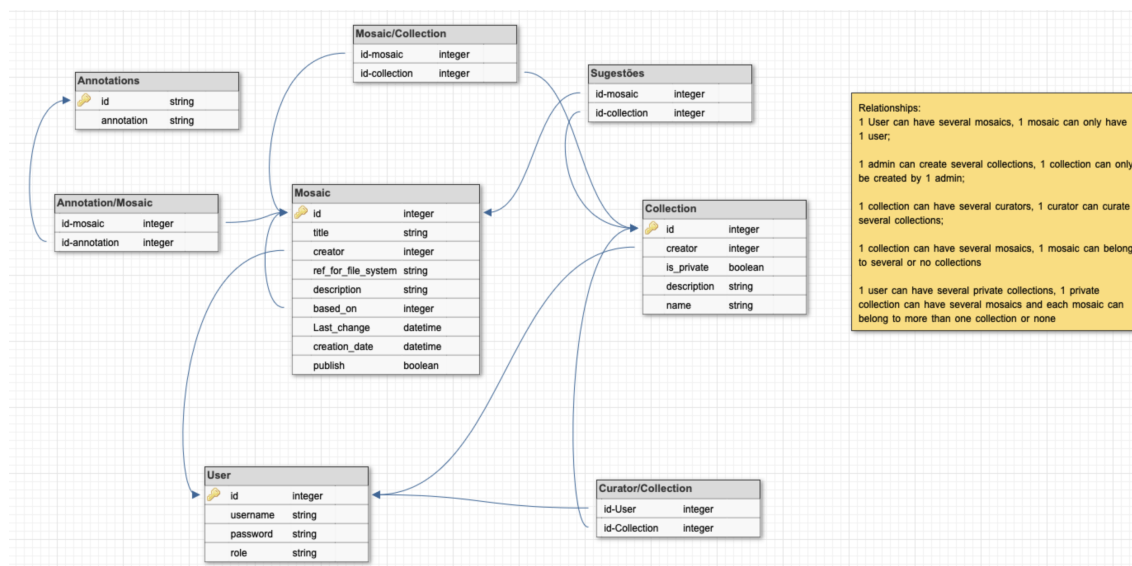


Figure 6.2: Database model for the new platform

In this model, several tables can be distinguished, there are 4 main tables that represent specific entities. The remaining tables are used for association purposes only containing foreign keys to build a relationship between two other tables. The main tables are:

- Mosaic
- Collection
- User
- Annotation

The Mosaic table contains information such as the title of the mosaic, the creator of the mosaic, the description, the reference for the file system, which mosaic it was based on (in case it was based on any mosaic), the date of the last change, the date of when the mosaic was firstly created and whether the mosaic is public or not. To access the files in the server file system, the field with the reference for file system is used.

The Collection table has fields similar to the Mosaic table, namely the creator, whether the collection is public or not, the title and the description.

Since there is a relationship between these two tables (Mosaic and Collection) one other table that relates the two of them is required, therefore, in order to keep track of which mosaics exist in a given collection, the table "Mosaic/Collection" was created with only 2 fields, the id of the collection and the id of the mosaic.

One other table has the same fields as the "Mosaic/Collection" table, that is the "Suggestions" table. However, the use of this table is different, in this table the mosaics are not present in the collection they are associated with, they are however, suggested to be placed in that collection. For instance, if a mosaic is based on another mosaic that exists in a given collection, perhaps the newly created mosaic may fit into that collection as well, therefore it can be added to the suggestions table.

The User table contains only 3 fields, these are the username, the password and the role. The user table relates directly with the Mosaic table, since the mosaics have a field dedicated to the creator, a direct relationship between these tables exist. The user also relates directly with the Collection table for the same reason, as a creator of said collection. However, the relationship between user and collection is not just of creator, it is also of curator. Since a user can curate collections, a table was created that maintains the connection between a user and the collections the user curates. This way, it is possible to verify which users have permission to add mosaics to a given a collection.

The last main table that needs to be discussed is the Annotations table. This table is used to store annotations of a given mosaic. As a possible requirement for this project, it was proposed to allow users to annotate mosaics. To better organize the annotations, a new table was created where the annotation itself would be stored. To be able to relate the annotation to the mosaic that was annotated, one other table was created. This table is the "Annotation/Mosaic" which contains the id of the annotation written and the id of the mosaic where the annotation was written.

## 6.2.3 User Operations

The sharing platform was idealized as a website where users would be able to instantly get to see projects developed with the mosaic editor, regardless of being registered. Therefore, some structural work needed to be done to better understand how the platform would behave depending on the state of logged in or not logged in of a user. To better structure the operations available to each kind of user, the diagram depicted in figure 6.3 was designed.

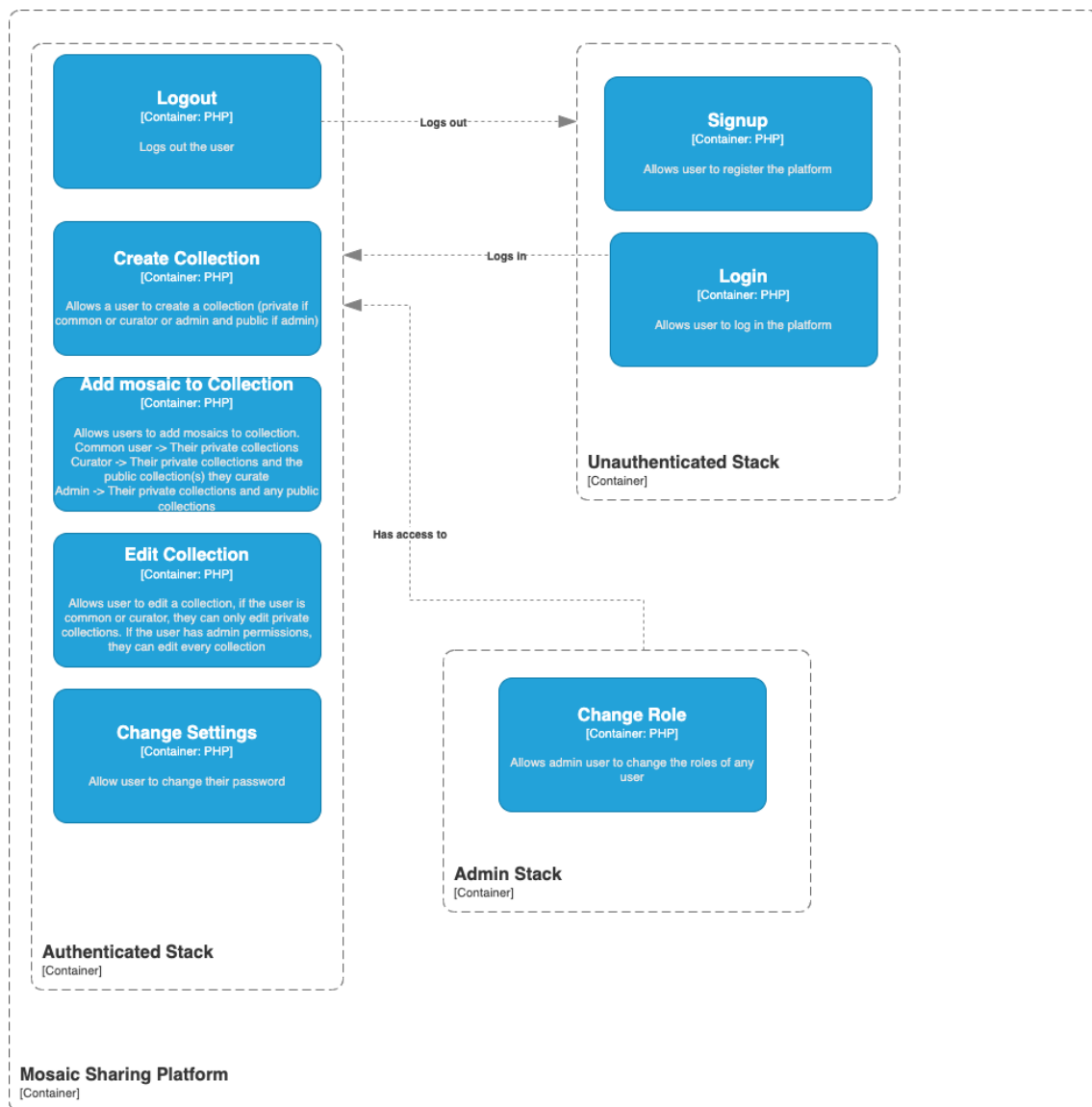


Figure 6.3: Container Diagram with the different stacks for user operations

As the image shows, we can distinguish three stacks of operations available to each type of user:

- Unauthenticated stack
  - Signup operation
  - Login operation

- Authenticated stack
  - Logout operation
  - Create collection operation
  - Add a mosaic to a collection operation
  - Edit a collection operation
  - Change user settings operation
- Admin stack
  - Change user role operation

The unauthenticated stack represents all the operations a user that hasn't logged in yet can perform. These operations don't allow the user to add information into the database, besides their own when they sign up. After logging in, the user can then make use of all the operations in the Authenticated stack. An admin user can make use of all the operations in the authenticated stack plus changing the role of other users.

All users that have an e-mail associated with their account, can become curators for public collections. If they don't have an e-mail associated, they aren't eligible to become curators. This decision came in response to the possibility of sensitive content being displayed on the platform. If the user in charge of curating mosaics doesn't have an email associated with them, they won't have much of a responsibility since that account isn't connected to them.

The operations that allow users to create a collection, edit a collection and add mosaics to a collection need to be further explained. As previously mentioned, a user needs to have an email associated to become a curator for a public collection. However, every user is able to create private collections. Furthermore, the user that creates a private collection is inherently the curator for that collection. This means that all users have the ability to create and edit collections as well as add mosaics to their collections. But, these collections are private and not public and can only be seen by them.

The user is always able to go to the mosaic editor, however, they need to be logged in to be able to save their mosaics in the database.

## 6.3 Mosaic Editor

For the mosaic editor, work had to be done related to implementing the new features, both painting tools and utilities such as the guidelines. Functionalities that require new painting tools are:

- Draw Rectangular Shape tool
- Draw Elliptic Shape tool



### 6.3.1 Communication with the Mosaic Sharing Platform

When referring to the communication with the Mosaic Sharing Platform we are talking about the process of obtaining the user and mosaic data which are present in the Mosaic Sharing Platform by the mosaic editor.

As seen in the activity diagram depicted in figure 6.5, the user would start their mosaics in the Sharing Platform, after which they would go to editor. When in the editor, the user information is fetched from the Mosaic Sharing Platform as it is important to know if the user is logged in or not, since a mosaic needs to have a creator to be saved into the database. If the user is logged in, they can proceed to draw their mosaic, if they are not logged in, they will have the choice to log in, by being greeted with a URL that redirects them to the Mosaic Sharing Platform. The user can, however not log in and draw the mosaic either way, however, as mentioned, a mosaic requires a user to create it. This means that the user will not be able to save the mosaic they drew if they don't log in. If the user is logged in however, the save button is available and they are free to save their mosaic. When saving the mosaic, the information and files representing the mosaic will be sent to the Mosaic Sharing Platform that will save both the information to the database and the files to the server file system.

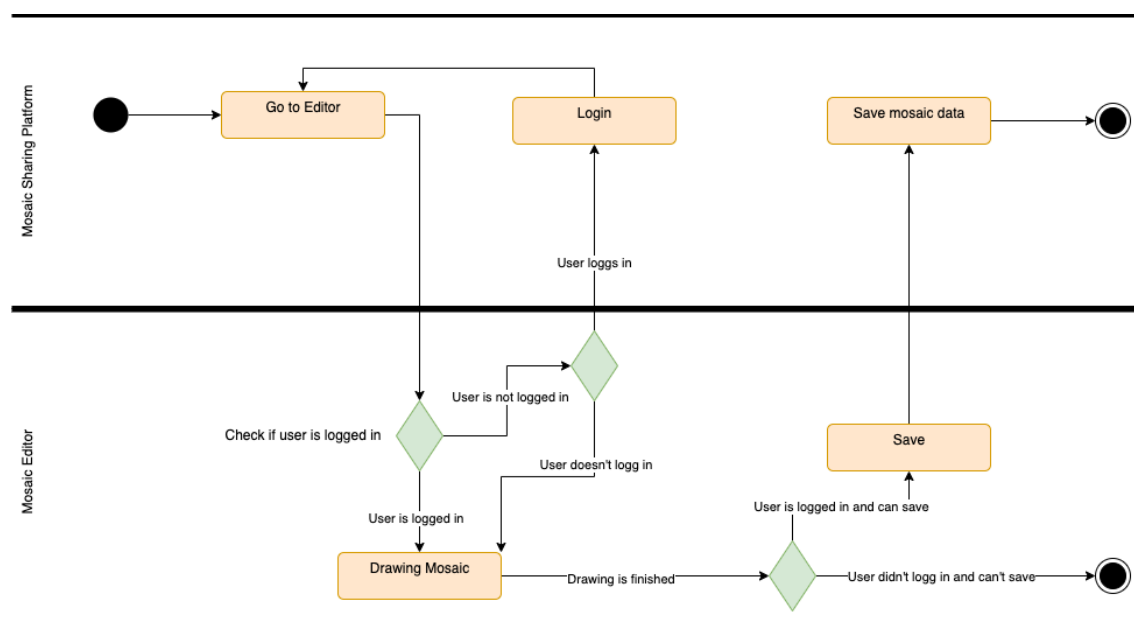


Figure 6.5: Activity Diagram representing the flow of information and the user tasks when using the platform to draw mosaics.

In order to distinguish between a new mosaic and saving over an already existing one, the need to have separate buttons for these operations arose. One button would allow the user to save, which would replace the previous mosaic entry on the database. The other button would create a new entry on the database similarly to the "Save as" button present in most application that deal with file editing (Paint, Word, Google Docs). If the user drawing a mosaic is the creator of the mosaic (meaning they started drawing from a mosaic that already exists, and they are the creator of said mosaic) the "Save" button will be available. If the user

is not the creator of the mosaic they are drawing over or if the mosaic is being drawn from scratch, then, the only button available will be the "Save as" button.



# Chapter 7

## Implementation

In this chapter we will be able to take a closer look into how the project was implemented. Both in the sense of the steps taken to achieve the final product as well as some technical aspects related to the project. Furthermore, we will see how the Mosaic Sharing Platform works and the added functionalities to the Mosaic Editor.

### 7.1 Development Environment

When explaining the implementation process of a given project, one very important aspect that needs to be referred is the development environment.

For this project a cross-platform Integrated Development Environment (IDE) was used, in this case, *PHPStorm*. This IDE was already being used by the creator of the Mosaic Editor and since it supported all the PHP capabilities needed to implement the Mosaic Sharing Platform it was used for this portion of the project as well.

The database was built using a MySQL client named *MySQLWorkbench*, this allowed for a better understanding of how each table was connected to one another as well as being able to see the information present in the database at the current time.

When running a PHP script, a local sever is needed for testing purposes. Therefore the *MAMP* tool was used.

### 7.2 Mosaic Sharing Platform

This portion of the project entailed the creation of an entirely new platform, therefore, the work done was very extensive. To better explain the process of implementing this portion of the project, the project structure will be presented and explained, the communication with the database will also presented as well as

how the files are managed through the server and how the platform behaves.

### 7.2.1 Project Structure

It's very important that a platform that may suffer some changes in the future, be easy to apply those changes to. Therefore, the project structure needs to be understood.

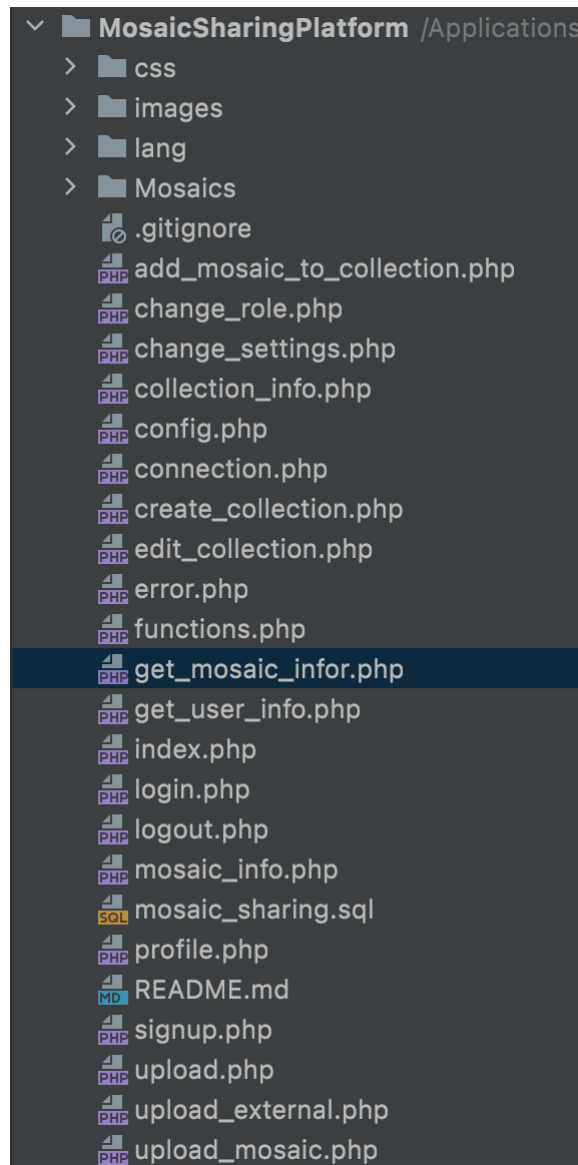


Figure 7.1: Project Structure of the Mosaic Sharing Platform

As presented in figure 7.1, four main folders can be seen followed by the several web pages that exist on the same level for the new platform. As to allow future developers to use their own environment for testing and developing purposes, a configuration file was created named "config.php". This file contains the database information as well as the links used to access the mosaic files in the server file system and the links of the Mosaic Sharing Platform as well as the link for the

mosaic editor. When adding new functionalities, the developer can change the links to their localhost. Using the configuration file, the developer is also able to establish the location for the folder where the mosaics will be saved.

### CSS folder

Starting by the first folder, this is where the CSS files exist. Currently, only one file exist in this folder. The style sheet used was adapted from an existing project, however, several new element styles were added for more specific behaviours.

This folder was created with the objective that if new developers want to work on this platform and add their own style, they will be able to do so by adding a new file to this folder.

### Images folder

This folder stores image files that are used in this project. For example the image used as the favicon of the webpage.

In case in the future, new developers need new images for certain HTML elements, this folder will already be created as a way for the images to be stored separately from the other files.

### Languages folder

In this folder two files exist. As is presented in figure 7.2, one of the files contains the English text and the other file contains the Portuguese text.

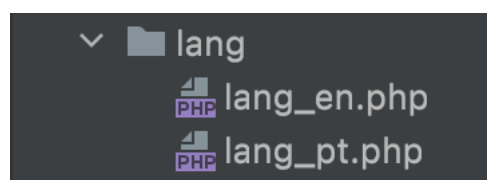


Figure 7.2: Language files available for the platform

These files were created with the intent of allowing a broader audience to experience the platform. These files contain the definition of several variables to a certain text. The same variables have the same text with different translations, as is visible in figure 7.3.

<pre>define("_DESCRIPTION", "Description:"); define("_PUBLICS", "Public mosaics"); define("_CHANGE_ROLES_EXP", "Change role of other users"); define("_USERNAME", "Username"); define("_ROLE", "Role"); define("_SAVE", "Save"); define("_NEW_PASS", "New Password:"); define("_CONFIRM_PASS", "Confirm New Password:"); define("_EDIT_COL", "Edit collection:"); define("_CREATE_COL_TITLE", "Create new Collection"); define("_COL_NAME", "Collection Name:"); define("_COL_DESCRIPTION", "Collection Description:");</pre>	<pre>define("_DESCRIPTION", "Descrição:"); define("_PUBLICS", "Mosaicos Públicos"); define("_CHANGE_ROLES_EXP", "Alterar papéis de outros utilizadores"); define("_USERNAME", "Nome de utilizador"); define("_ROLE", "Papel"); define("_SAVE", "Guardar"); define("_NEW_PASS", "Nova password:"); define("_CONFIRM_PASS", "Confirme a nova password:"); define("_EDIT_COL", "Editar coleção"); define("_CREATE_COL_TITLE", "Criar nova coleção"); define("_COL_NAME", "Nome de coleção:"); define("_COL_DESCRIPTION", "Descrição da coleção:");</pre>
---	---

Figure 7.3: Comparison between an excerpt of the English language file and the Portuguese language file

If a new developer wishes to add a new language, they would only need to create a new language file and replace the text with its translation to the desired language.

### Mosaics folder

As mentioned previously, the mosaics are saved in the server. When testing and using the *localhost*, the mosaics were saved on the developers' computer inside this folder. With the code written, this folder is always created on the server in case the folder doesn't yet exist. This folder contains several folders with unique *ids* which are the same as the "ref\_for\_file\_system" values on the database, inside each of those folders there is one PNG file named "image" and one JSON file named "project"(figure 7.4).

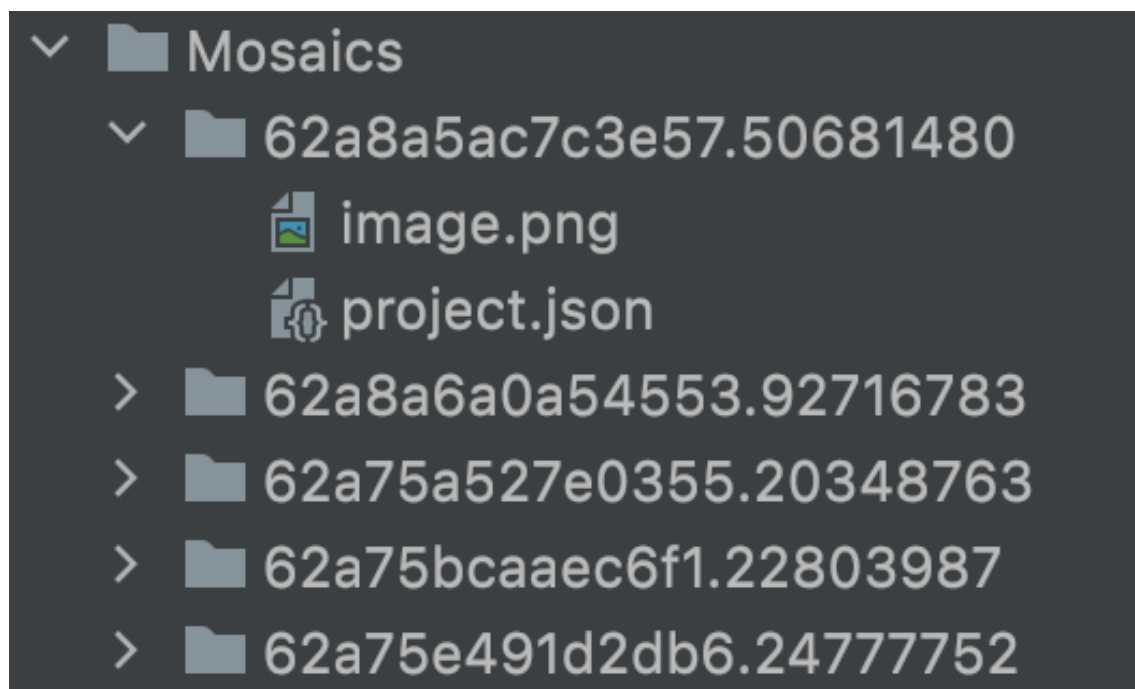


Figure 7.4: Folder that contains all the Mosaics and their project and image files

## 7.2.2 Communication with the database

Since this project has a lot of communication with the database for reading and writing data. It is important to structure those occurrences. The way this was performed was by creating a file named "functions.php" which would contain all the functions related to reading data from the database. This file is included in all the web pages, so that all information that is needed from the database can be accessed through any web page.

One of the most important functions present in the "functions.php" file is the "check\_login" function (listing 7.1). This fetches the user session data to verify the user id and then checks if that id is present in the database. This function then returns the user data, which is very important in all of the platform to know which functionalities they are able to access or not.

```
1
2
3 function check_login($con){
4
5     if(isset($_SESSION['user_id'])){
6         $id = $_SESSION['user_id'];
7         $query = "select * from users where id = '$id' limit 1";
8         $result = mysqli_query($con, $query);
9         if($result && mysqli_num_rows($result)>0){
10            $user_data = mysqli_fetch_assoc($result);
11
12        }
13    }else{
14        $user_data = false;
15    }
16    return $user_data;
17    die;
18
19 }
```

Listing 7.1: check\_login function used to get the user information

The rest of the functions are mostly related to fetching some sort of information needed from the database. As we can see in listing 7.2, the structure of this functions is as follows, the needed fields for the Structured Query Language (SQL) query are sent by parameter to the function (in the listing, the variables are the connection to the database and the user id). Then we create a query that will fetch the needed information. In this case, the result will contain several objects, therefore, we create an array and populate it with the results. Finally, the function returns the result.

```

1
2
3
4 function get_users_private_mosaics($con, $id) {
5
6     $query = "SELECT * FROM mosaic WHERE public=1 OR
7     (public=0 AND creator='$id');";
8     $result = mysqli_query($con, $query);
9     $mosaics = array();
10    while($row = mysqli_fetch_assoc($result)) {
11
12        $mosaics[] = $row;
13    }
14
15    return $mosaics;
16 }

```

Listing 7.2: Function used to get the mosaics that can be placed in a given users private collection

All the other functions present in this file follow the same pattern as this, therefore once a new developer understands how one of the functions work, all of the other functions are easy to understand.

However, this file only contains read operations. In order to have something to read, something must be placed in the database to begin with. All the files that will produce a new entry or update a table in the database, contain the script to do so. For example, the web page dedicated to add mosaics to a collection. In this web page, the user can pick mosaics to add to that collection and mosaics that are already present in the collection to remove. This entails altering the database information.

```

1
2 <?php
3
4 if(isset($_POST['submit'])) {
5
6     $query2 = "DELETE FROM 'mosaic-collection'
7     WHERE id_collection='". $col. "'";
8     mysqli_query($con, $query2);
9
10    if(!empty($_POST['formMos'])) {
11
12        foreach ($_POST['formMos'] as $id_mosaic) {
13            if(in_array($id_mosaic, $suggestions)) {
14                $query = "DELETE * FROM suggestions
15                WHERE id_collection='". $col. "'
16                AND id_mosaic ='". $id_mosaic. "'";";
17            }
18
19            $query = "INSERT INTO 'mosaic-collection'
20            (id_mosaic, id_collection)
21            VALUES ('$id_mosaic', '$col')";";
22            mysqli_query($con, $query);
23        }
24    }
25    echo "<meta http-equiv='refresh' content='0'>";

```

```

26 }
27 ?>

```

Listing 7.3: Script used to communicate with the database in the web page dedicated to adding or removing mosaics from a collection

As we can see in listing 7.3, the server receives a "POST" request (in this case, it has been submitted a "form" with all the mosaics to be added to the collection). Therefore, we will delete all mosaics currently present in the collection and then add all the mosaics that were "checked" to be present in the collection.

This is just an example of a behaviour that occurs in several web pages, when it is needed to insert or delete information, that operation takes place directly on the PHP file that is in charge of rendering the web page.

### 7.2.3 Managing files from the server

As previously mentioned, the server file system contains all the mosaic information, organized as presented in figure 7.4. In several instances, mosaic file information is needed, therefore the storage is very important. In this subsection the uploading of the files to the server will be presented as well as how they are used.

#### Uploading Files

To allow for the edition and addition of files to the server, a file named "upload\_external.php" exists. This file takes care of receiving the mosaic information and adding the new mosaic object to the database and the mosaic files to the server. To do so, some steps are needed.

Firstly, we need to have the user data as to know the creator of the mosaic, the JSON and PNG files are fetched, as well as all the other parameters (mosaic title, reference for the file system, whether it is public or not, mosaic description, name of the files).

From this first step, some information can already be used to deduce if the mosaic is being added or edited. To do so, we verify the existence of the reference parameter. If the parameter was sent then the mosaic exists and it's being updated. If the parameter does not exist then it is a new upload and a new entry to the database must be performed. This first step can be seen in listing 7.4.

```

1
2
3 $user_data = check_login($con);
4
5 $fileJson = $_FILES['file-json'];
6
7 $filePng = $_FILES['file-png'];
8
9 $mosaic_name = $_POST['mosaic_name'];
10
11 if(!empty($_POST['ref'])) {

```

```

12     $mosaic_ref = 'Mosaics/'. $_POST['ref'].'';
13 }else {
14     $mosaic_ref = null;
15 }
16
17 if(isset($_POST['public']) && $_POST['public'] == "Yes") {
18     $is_private=1;
19 } else {
20     $is_private = 0;
21 }
22
23 $mosaic_description = $_POST['mosaic_description'];
24
25 $fileNameJson = $_FILES['file-json']['name'];
26 $fileTmpNameJson = $_FILES['file-json']['tmp_name'];
27 $fileSizeJson = $_FILES['file-json']['size'];
28 $fileErrorJson = $_FILES['file-json']['error'];
29 $fileTypeJson = $_FILES['file-json']['type'];

```

Listing 7.4: First step needed to upload a new mosaic or an edited mosaic

After having all this information we must verify that the files have the correct extensions (JSON and PNG) and check if the files have no errors. This is done with two simple verification's presented in listing 7.5.

```

1
2
3 $fileErrorJson = $_FILES['file-json']['error'];
4 $fileExtJson = explode('.', $fileNameJson);
5 $fileActualExtJson = strtolower(end($fileExtJson));
6 $fileExtPng = explode('.', $fileNamePng);
7 $fileActualExtPng = strtolower(end($fileExtPng));
8 $allowed = array('json', 'png');
9 $fileErrorPng = $_FILES['file-png']['error'];
10
11 if(in_array($fileActualExtPng, $allowed) and in_array(
12     $fileActualExtJson, $allowed)) {
13     if($fileErrorPng === 0 and $fileErrorJson === 0) {
14         .
15         .
16         .

```

Listing 7.5: Verifying that the files have the correct extensions and that there are no errors

PHP allows for verification code to detect error on files, therefore if the code is "0", no errors were found.

The next step is to check if the mosaic reference was passed or not. If no mosaic reference was passed, then a new directory needs to be built, to do so we use the "uniqid" method to generate a unique id for the mosaic reference and create a new directory under the Mosaics folder. We then double check that the directory is in fact unique by searching for a directory with that id. When no directory is found we insert the files to the new directory. This process can be seen in listing 7.6.

```

1
2 $directoryName = uniqid('', true);
3 $newDirectoryName = 'Mosaics/'. $directoryName.'';

```



```

4
5 if(!is_dir($newDirectoryName)) {
6
7     mkdir($newDirectoryName,0777,true);
8     if($fileActualExtJson == 'json') {
9         $fileNameNewJson = "project.".$fileActualExtJson;
10        $fileNameNewPng = "image.".$fileActualExtPng;
11
12    } else {
13        $fileNameNewJson = "image.".$fileActualExtJson;
14        $fileNameNewPng = "project.".$fileActualExtPng;
15    }
16
17    $fileDestinationPng = $newDirectoryName.$fileNameNewPng;
18    $fileDestinationJson = $newDirectoryName.$fileNameNewJson;
19    move_uploaded_file($fileTmpNameJson, $fileDestinationJson);
20    move_uploaded_file($fileTmpNamePng, $fileDestinationPng);

```

Listing 7.6: Process of creating a new directory for the files received

If the mosaic reference is passed, then the process is slightly different. First, we must check the directory exists, then we delete the previously existing files and then we add the new files to the directory that already existed. The process of deleting the files in the directory is presented in listing 7.7. The rest of the process is presented in listing 7.6 after the "mkdir" method is used.

```

1
2 if($mosaic_ref != null && is_dir($mosaic_ref)) {
3
4     $dir = new DirectoryIterator(dirname($mosaic_ref));
5     // Deleting all the files in the list
6     foreach ($dir as $fileinfo) {
7         if (!$fileinfo->isDot()) {
8             // Delete the given file
9             unlink($fileinfo->getPathname());
10        }
11    }

```

Listing 7.7: Deleting the contents of the directory to add the new files

One last thing that needs to be taken into account regarding this file that uploads the mosaics to the server is that the operation of inserting the data to the database occurs as well. When the mosaic reference isn't passed (mosaic does not exist yet), a "Insert" query is used. When the reference is passed, a "Update" query is used. Since we also get the name, description, if it is public or not we can add those values to the query to fill the most information possible. Furthermore, if it is a new mosaic, we use the "creation\_data" field, if it is an old mosaic we use the "last\_change" field to establish the date.

## Using the files

The mosaic files are used in several instances and therefore it is important to know how they are presented and used. A good example of fetching the image files is the index page since in the homepage of the platform, several mosaics are

presented in the public collections and therefore, the image file is needed. To do so, the mosaic information is read from the database. From that information, the reference for the file system is extracted. Using that reference it is now possible to insert the image by using the HTML "img" tag which allows us to choose the source. This can be seen in listing 7.8.

```

1
2 <?php $mosaic = get_mosaic_by_id($con,$mosaic_id['id_mosaic']);
3 $mosaic_name = $mosaic['title'];
4 $ref = $mosaic['ref_for_file_system'];?>
5 <div class='w3-display-topleft w3-black w3-padding'><?php echo
   $mosaic_name?></div>
6 <a href='mosaic_info.php?mos=<?php echo $mosaic['id']?>'><img src='
  <?=MOSAIC_FOLDER?><?php echo $ref?>/image.png' alt='Mosaic'
  style='width:100%'></a>

```

Listing 7.8: Using the image file to display the mosaic

The JSON files however, are never directly exposed in the Mosaic Sharing Platform. They are, however, used to start mosaics starting from that file by the mosaic editor. This can be done by placing the directory of the mosaic to be edited or remixed in the URL that links to the mosaic editor. This can be seen in listing 7.9. Here we have the mosaic information once again. On the "href" field we call the mosaic editor URL followed by the project file where we call the Mosaic Sharing Platform Mosaic folder, followed by the reference of the mosaic and the name of the file (project.json).

```

1
2 href='<?=MOSAIC_EDITOR?>?project=<?=SHARING_PLATFORM?><?=
  MOSAIC_FOLDER?><?php echo $ref?>/project.json'

```

Listing 7.9: Using the mosaic directory to get the JSON file for the mosaic editor to use

## 7.2.4 Platform behaviour

So far, the several backend communications have been explained as well as how the mosaic data is used from the database and the server. However, there are several aspects that still deserve some attention regarding the platform behaviour.

Namely, how the user session is created. This happens in the login page, the information read is verified by the data in the database and if it checks out the session data stores the user id. The process can be seen in listing 7.10. Firstly we search the database for the username inserted by the user. If we get a result (represent in the figure by "\$result") that contains any information (the second if statement). So we just have to check that the password inserted matches the password present in the "\$result" object which contains the users which has the username inserted data. If the password matches we can now establish the session id to match the user id.

```

1 $query = "select * from users where username = '$user_name' limit 1
  ";
2 $result = mysqli_query($con, $query);

```

```

3
4 if($result){
5     if($result && mysqli_num_rows($result)>0){
6         $user_data = mysqli_fetch_assoc($result);
7
8         if($user_data['password'] === $password){
9             $_SESSION['user_id'] = $user_data['id'];
10            header("Location: index.php");
11            die;
12        }
13    }
14 }

```

Listing 7.10: Login verification and storing the user id in the session

Having the session id allows for any web page to call the method presented in listing 7.1 to get the user data.

Having the user data, the platform adapts according to the user role and state of logged in or not. To exemplify this, we can use any page. All the pages check the user information to draw the navbar on the top portion of the web page. The process starts by verifying if the user data exists. If it does not, then we know the user is not logged in and therefore we present the options to login and signup. If the user is logged in we check if the user is an admin user, if they are, then we can add a tab to the navbar for quick access to the web page that allows to change roles. Any user logged in that is admin or not has access to the remaining web pages, the profile and the logging out button. This verification steps can be seen in listing 7.11.

```

1
2 <div class="w3-top">
3     <div class="w3-bar w3-black w3-wide w3-padding w3-card">
4         <a class="w3-bar-item w3-button" href="index.php" ><b>
Mosaic Sharing Platform</b></a>
5         <a class="w3-bar-item w3-button" href="<?MOSAIC_EDITOR?>"
target="_blank" rel="noopener noreferrer"><?=_EDITOR?></a>
6
7
8         <div class="w3-right w3-hide-small">
9             <?php if($user_data):?>
10                <?php if($user_data['role'] == 'admin'):?>
11                    <a href="change_role.php" class="w3-bar-
item w3-button"><?=_ROLES?></a>
12                <?php endif; ?>
13                    <a href='profile.php?id=<?php echo $user_data['id'
]?>' class='w3-bar-item w3-button'><?php echo $user_data['
username']?></a>
14                    <a href="logout.php" class="w3-bar-item w3-button"
><?=_LOGOUT?></a>
15                <?php else:??>
16                    <a href="login.php" class="w3-bar-item w3-
button">Login</a>
17                    <a href="signup.php" class="w3-bar-item w3-
button"><?=_SIGNUP?></a>
18                <?php endif;?>

```

Listing 7.11: Verifying if the user is logged in and what their role is

As such, if the user is an admin user the navbar will be presented as seen in figure 7.5, while the not logged in user will be presented with the navbar seen in figure 7.6 and a regular user will be presented with the navbar seen in figure 7.7.



Figure 7.5: Admin Navbar



Figure 7.6: Not logged in user navbar



Figure 7.7: Regular navbar

The process of verifying what role a user has is present in several other instances. If a user is an admin user they will have the possibility of creating a new public collection right in the home page. Furthermore, if a user is admin, they are also able to edit and add mosaics to any public collection. This possibilities are presented in the web page where the collection information is displayed. This can be seen in figure 7.8.

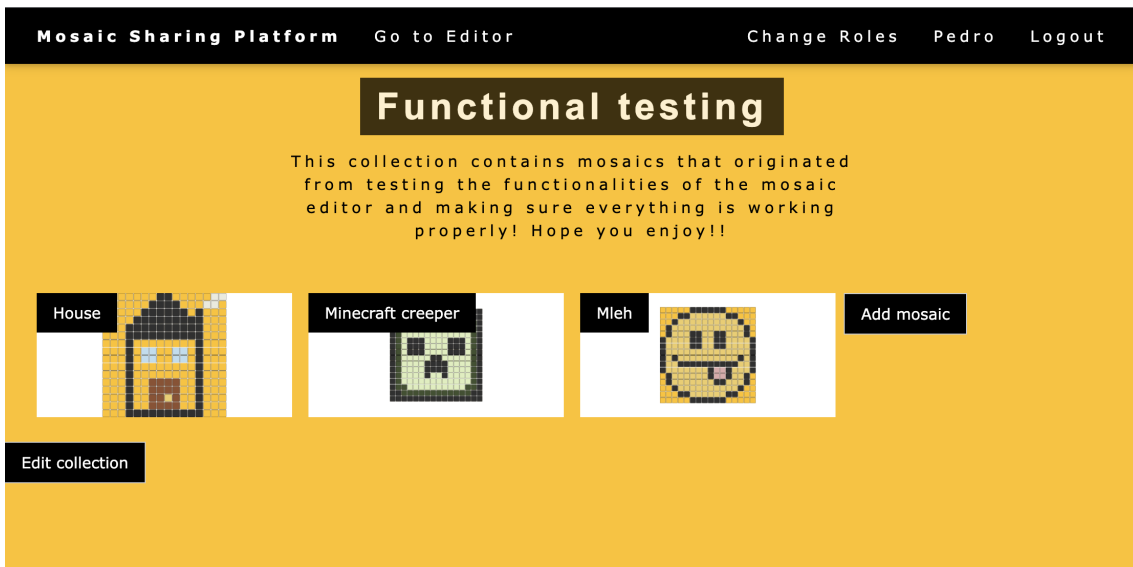


Figure 7.8: Admin collection info page, for regular users, the add mosaic button is locked as well as the edit collection. For curators, the add mosaic button is visible but not the edit collection.

Apart from buttons being available, to certain user roles, there are pages which are not supposed to be accessed by any user. This entails the addition of verification to some pages, these pages are:

- Change roles
- Collection info (private collections)
- Mosaic info (private mosaics)
- Add mosaic to collection (on any collection that the user does not curate)
- Edit collection (on any collection the user has not created or the user is not admin)

The process of checking who is visiting is fairly simple. For the first page in the list above, the only users that can access are admin users. Therefore, we check the user role before rendering the page. If the role is not admin, then we redirect the user to an error page as seen in listing 7.12.

```

1 if ($user_data['role'] != "admin") {
2     header("Location: error.php?err=notallowed");
3 }

```

Listing 7.12: If statement that checks if a given user has admin role to access the change role web page

For the second page, we need to check if the collection is private, if it is, then we verify if the users id corresponds to the collections creator. If the ids don't match, then the user is redirected to an error page once again. The process can be seen in listing 7.13.

```

1 if($collection['is_private'] == 1 && $user_data['id'] !=
    $collection['creator']) {
2     header("Location: error.php?err=notallowed");
3 }

```

Listing 7.13: If statement that checks if a given user is the creator of a private collection

For the third page the process is the same, but instead of checking for the collection creator, we check for the mosaic creator.

To add mosaics to a collection it is more complicated since several scenarios exist. Firstly, we make the verification performed in the previous pages, in case the collection is private. If the collection is public however, we must check if the users id matches any of the curators of this collection or if the user role is admin. If any of these conditions pass then the user can see the page, if not, then the user is once again redirected to the error page. The process of verification can be seen in listing 7.14.

```

1
2 $curators = get_curators_from_collection($con, $col);
3 if($collection['is_private'] == 0) {
4     if(!in_array($user_data['id'], $curators) && $user_data['id']
5         != $collection['creator'] && $user_data['role'] != "admin") {
6         header("Location: error.php?err=notallowed");
7     }

```

7 }

Listing 7.14: If statement that checks if a given user is a curator or the creator of the collection or if the user is an admin user

The edit collection page is much simpler since the only users that should be able to edit collections are the admin users for the public collections and the creators for the private collections. Therefore only two verifications need to happen, one of them is the same as the verifying if a user can see a private collection and the other is the same as the one presented in the first case, where we verify if a user is an admin user in case the collection is public.

The error web page to where the users are redirected when trying to access web pages they don't have the rights to see is a blank page with the error message "You don't have permissions to see this page." shown.

Other behaviours related to how the platform adapts can be seen with the language selection. Previously it was mentioned that two language files existed and how they were structured. But it has not been explained how they are used. A drop down menu is present in the right most corner of the navbar. This drop down contains the options of languages to select. These options are part of a form and whenever an option is selected, the form is submitted. The implementation of this form can be seen in listing 7.15.

```

1
2 <form method='get' action='' id='form_lang'>
3     <select name='lang' onchange='changeLang();'>
4         <option value='en' <?php if(isset($_SESSION['lang'])) &&
5             $_SESSION['lang']=='en'){echo "selected";} ?>>English <span>#
6             x1F1EC;&#x1F1E7;</span> </option>
7         <option value='pt' <?php if(isset($_SESSION['lang'])) &&
            $_SESSION['lang']=='pt'){echo "selected";} ?>>Portugues <span>#
            x1F1F5;&#x1F1F9;</span></option>
        </select>
    </form>

```

Listing 7.15: Form used to display the drop down menu with the different languages the platform has available

Having the language selected it is now needed to load the correct file. Since the language letters are submitted ("en" for English and "pt" for Portuguese), we can use that to fetch the files as shown in listing 7.16. We check if any language was picked, if so we set the language for the session to be that language. If the language present in the session is not the same as the one submitted, the page is reloaded. On loading the page we check if the session has a picked language. If it does, we include the file that relates to that language, if not, by default the English page is loaded.

```

1
2 if(isset($_GET['lang']) && !empty($_GET['lang'])) {
3     $_SESSION['lang'] = $_GET['lang'];
4
5     if(isset($_SESSION['lang']) && $_SESSION['lang'] != $_GET['lang']
6         ')] {

```

```
6     echo "<script type='text/javascript'>location.reload();</  
7     script>";  
8 }  
9 if(isset($_SESSION['lang'])) {  
10     include "lang/lang_" . $_SESSION['lang'] . ".php";  
11 } else {  
12     include "lang/lang_en.php";  
13 }
```

Listing 7.16: Logic used to know which language file to use

Two other files that have not been mentioned yet but need to be understood are PHP scripts used to return information when called by the mosaic editor. One of the files returns the user id and the other, receives a mosaics reference for the file system in order to search the database for that mosaic information and return it. These files are used so that the mosaic editor can see if the creator of the mosaic is the user that is currently logged in and provide additional options. This will be further explained in section 7.3.1.

## 7.3 Mosaic Editor

In this section, the work developed in the mosaic editor side of the project will be presented. This work can be divided into two categories, the communication with the Mosaic Sharing Platform and the added functionalities to the editor.

### 7.3.1 Communication with the Mosaic Sharing Platform

One aspect that is required to make the platform work in an intuitive manner is to be able to save the mosaics from the mosaic editor into the Mosaic Sharing Platform. This entails that the mosaic editor possesses some information, user information and the mosaic information as in, the mosaic that appears on the canvas when the user gets to the mosaic editor. To ensure the mosaic editor has this information, the *fetch* API, available in *JavaScript* is used.

As mentioned in the previous section, the Mosaic Sharing Platform has two scripts in charge of returning the information needed. But how is this communication done? Well, firstly, the *javascript fetch* function is used. This function allows to post information into other addresses. Using this function, we make reference to the file in the Mosaic Sharing Platform that is in charge of returning the user information. The Mosaic Sharing Platform therefore will receive a request and act accordingly, returning the user id of the current user. If the user is not logged in it will return *null*.

Even with the user information, there is still a need to know if the mosaic the user is working on belongs to them or not. This can be done using the *fetch* function once again. Since when getting to the mosaic editor the URL will either contain a "project" element or not, we are able to extract from the "project" element (in case

it exists) the reference to the server file system. The reference is also present in the database since it is important to keep a relationship between the mosaic objects and the correspondent mosaic files. Therefore sending the reference we are able to get from the URL through the *fetch* function, we receive a response from the Mosaic Sharing Platform containing all information regarding that mosaic, including the creator.

Now that the mosaic editor has all the information it needs, it is important to establish some behaviour flows. The behaviour is represented by the activity diagram present in figure 7.9.

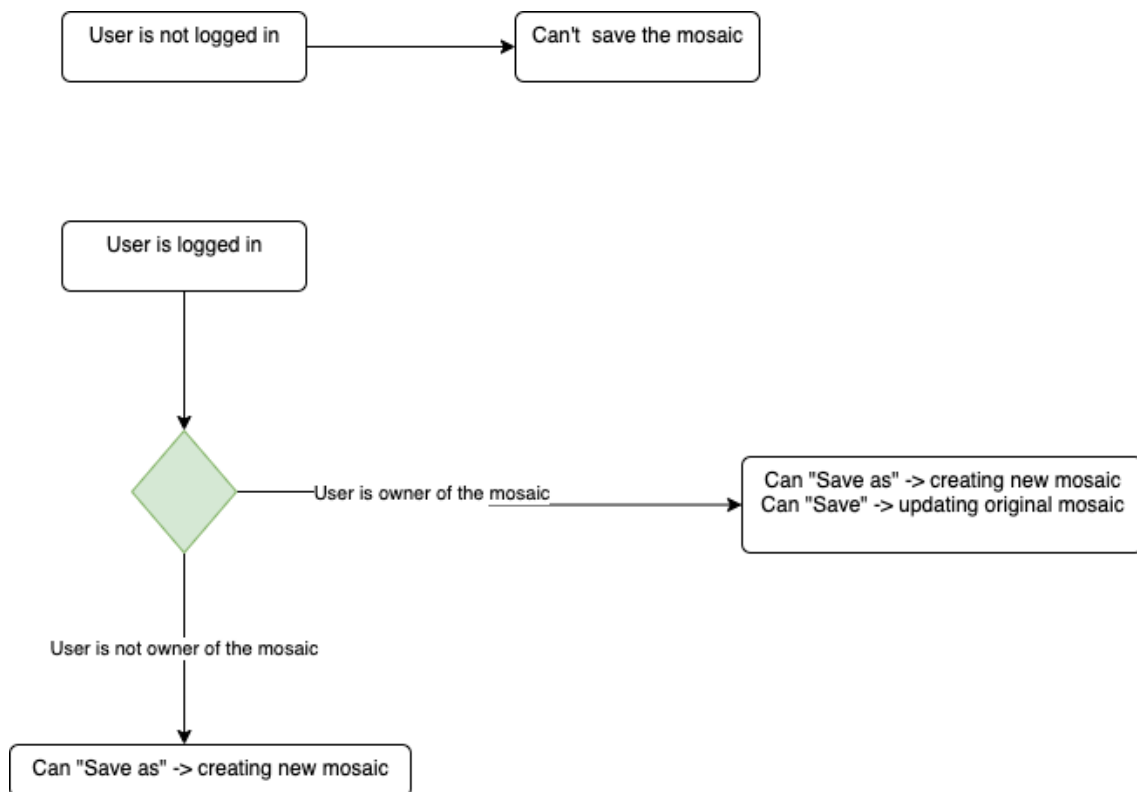


Figure 7.9: Logic behind how a user can save a mosaic depending on the mosaic and user information

As seen in the diagram, the user can perform different types of "save" operations depending on whether they are the owner of the mosaic or not. If the user is not logged in at all, they can't save the mosaic. This approach made the most sense since the creator of a mosaic might not have finished what they were aiming to accomplish, therefore, it is important that they are able to change their mosaics. However, they might also be finished with a mosaic but want to change some aspects of that same mosaic and create a new one instead of updating the original. A user that is not the owner of the mosaic shouldn't have the option to update the original mosaic, therefore, the only option is to create a new one based on the original.



### 7.3.2 Added Functionalities

For the new functionalities, a new architecture was proposed, mentioned in chapter 6. This would entail the implementation of a "Responsibility Chain" which would take care of handling the new drawing/painting functionalities.

To implement this new architectural design pattern, a new class was created named "Tool". This class would have three parameters:

- nextTool (Tool)
- active (Boolean)
- name (String)

Since this design pattern symbolizes a chain, the "nextTool" represents the next Tool in the chain. The active parameter will be used to know which tool to use. The name is the parameter that allows to differentiate the Tools from one another.

Furthermore, some methods need to be implemented in this class, namely:

- setNextTool(Tool)
- processEvent(evt)
- isActive(), returns a Boolean
- setActive(name)

The "setNextTool" can be used to add a tool to the Tool chain, generally to the "front" of the chain. The "processEvent" method is called when an input is received and needs to be processed. The "isActive" method serves as a way to understand if a tool is active or not and the "setActive" method receives a tool name, navigates the chain searching for that name. If a certain tool has a different name, then it's active parameter is set to false, otherwise, it is set to true. This means that the newly activated tool will be the one to be used and all the other tools will not be active, therefore not hindering the process handling.

Since the buttons representing the tools are set in the sketch class, it is mandatory that their correspondent tool is set from there too. However, the handling of events takes place in the mosaic-editor class. This entails that when rendering the buttons, the mosaic-editor class is called with the method "addTool" this method calls another class named "ToolFactory" which is in charge of verifying that the tool to be added actually exists and then returns the tool object, which is then added to the chain present in the mosaic-editor class.

When a button is pressed, the corresponding command takes care of calling the mosaic-editor method "setActiveTool" which takes the tool name and a boolean as parameters. This mosaic-editor method will then call the tool chain "setActive" method to activate the tool in question.

When an input arrives at the mosaic-editor, the tool chain is called with the method "processEvent" and the event as the parameter. From there on out, each tool will evaluate their own state of active from the parameter "active", if they are active, they will handle the event, if not, they will refer to the next tool in the tool chain and call their "processEvent" function. This is repeated until the end of the chain. This behaviour is represented by an activity diagram presented in figure 7.10

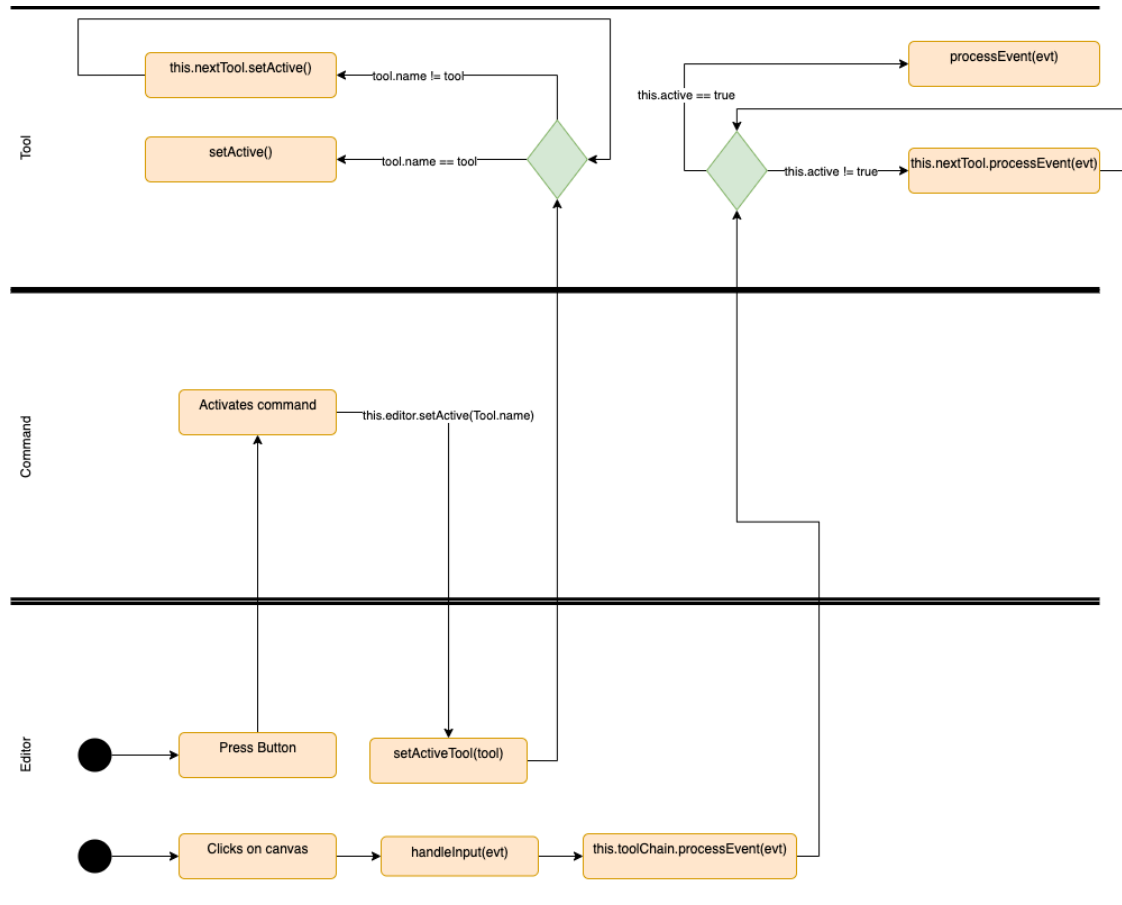


Figure 7.10: Activity Diagram for the Chain of Responsibility applied to the mosaic editor

### DrawShapeTool

The DrawShapeTool was implemented in the first semester and adapted into the current architecture this semester, its basic functionality relies on gathering the mouse position, when it first presses down on the canvas, during the movement and finally when it is let go. With the initial mouse position and the position while it is being pressed down and dragged along the canvas, a rectangle is drawn, representing the size of the rectangle to be drawn once the mouse is let go of. When the mouse is freed, using the initial point and the final point we can calculate all the tiles to be painted.

Basically, using two points (P1 and P2 for example), we must calculate the difference between their x and y values to obtain the tiles to be painted. For the upper

line of the square, we need the P1 y value, and the beginning of the line is delimited by the P1 x value and the P2 x value. For the lower line of the square the same thing applies, however, the y value of the line is the same as the P2 y value instead. The logic is the same for the vertical lines but instead of calculating the difference between the x values, we calculate the difference between the y values of both points.

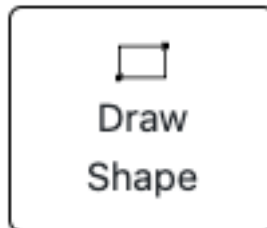


Figure 7.11: Button used to initiate the draw shape operation

### SwitchColorTool

For this tool, the objective is that the user can change all color occurrences of a certain color, to a selected one. Therefore, the process is very straightforward, the user clicks a tile, the tile color is saved so that we know which color to look for. Then, we search all the tiles that have that color, and change their color to the selected color. The user can select the color to change before pressing the button to switch color or after pressing the button but before pressing the tile with the color to be changed.

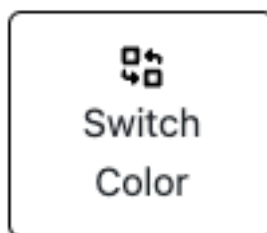


Figure 7.12: Button used to initiate the switch color operation

### AreaSelectionTool

The area selection tool allows the user to drag a rectangle around the tiles they wish to select, upon letting go of the mouse button, the tiles inside that rectangle become selected. Once the tiles are selected, the user can then perform any of the other tools actions only in the selected tiles instead of the whole mosaic. To achieve this, we made use of an already existing parameter in the mosaic-view class which adds tiles to an array and then makes them "Selected" by attributing some CSS style. The array is then used in the other tools instead of the whole mosaic array. The selection process works similarly to the DrawShapeTool when

the mouse has not been let go of. When the mouse is freed, we fetch all the tiles inside the boundaries defined by the two points based on their coordinates and add them to the array in the mosaic-view class.



Figure 7.13: Button used to initiate the area selection operation

### FillInTool

Something that needs to be taken into consideration before explaining the implementation of this tool is how the tiles are drawn. The tiles on the mosaic make use of a *quadtree* data structure. "Quadtrees are trees used to efficiently store data of points on a two-dimensional space. In this tree, each node has at most four children." [Kamath, 2018]. Using this data structure, we define the size of the tiles and their position on the mosaic. Therefore, several functions exist that make use of this data structure to find interceptions between tiles and the mouse position.

This tool took two iterations to accomplish. The basic idea is that a user can use this tool to fill in areas of a mosaic. An area is delimited either by the limits of a mosaic or by another color. For instance, if a user were to draw a square, therefore creating an enclosed area, when using the fill in tool inside the square it is supposed to stop at the edges of the square. In the first iteration, the solution was to use recursion. This solution was as follows. First, we would check the color of the tile clicked, then we would call a function that would take a tile, the original color of the tile and the index of the tile. Since we know the mosaic is an array of tiles organized in increasing order from top left to bottom right, we know that to reach the tile below a certain tile, we would have to search for the tile in the index of the previous tile added to the width of the mosaic (in number of tiles). To get the tile above the same process but instead of adding we would subtract, to go to the right we would increment by one and to go to the left we would decrement by one. Knowing this, the function called would check if a given index is between the boundaries of the mosaic (meaning that it is bigger than 0 and less then the length of the array). Then, it would check if the color of the tile is the same as the original color. If both these conditions were true, then we would paint the current tile, and call the function once again, for the tile above, below, to the left and to the right. This would ensure that all the tiles that should be painted were indeed painted. However, a problem arose, if a mosaic doesn't follow a layout, then this method won't work. Take the figure 7.14 for example.

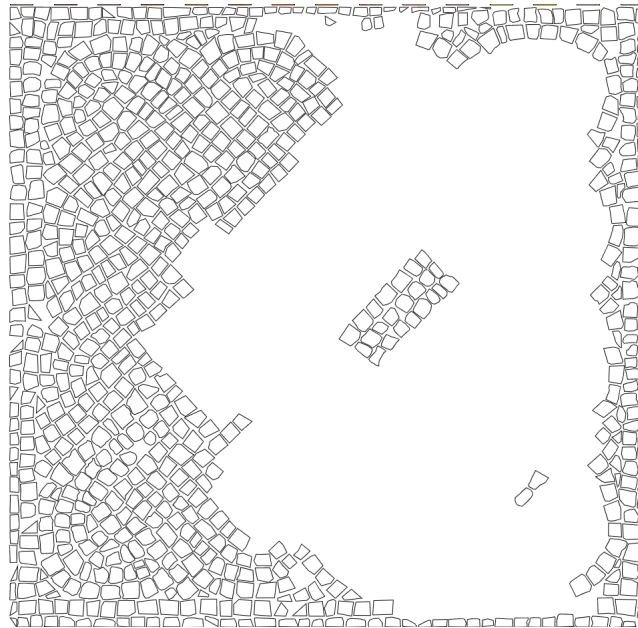


Figure 7.14: Mosaic without a predefined layout where the first technique used to paint the tiles doesn't work

In this mosaic, we can't depend on the indexes to paint the mosaic, therefore, a new solution is needed.

The new solution relies on creating a new function that uses the *quadtree* data structure. Using this data structure, we can now create a square that has a dimension of 1.5 times bigger than the tiles *quadtree* bounding box. Then we search for collisions between the new *quadtree* square and any other tiles bounding boxes. From the collisions, we can obtain the tiles that exist in the direct proximity of the current tile. Using this method, it is possible to get a list of tiles that are close to the current tile, from there we can reproduce the same method used previously, only instead of calling the function for the tiles below, above and directly to the sides, we call for all the tiles found using the new method (listing 7.17).

```

1
2 fillIn(tile, originalColor, mosaicView, newColor, cmdList,
   paintedTiles, sketch, selected, selectedTessels) {
3     if(tile.colorIndex != originalColor || paintedTiles.has(
   tile.id)){
4         return;
5     }else {
6         let cmd = new CmdTileChangeColor();
7         cmd.mosaicView = mosaicView;
8         cmd.tesselId = tile.id;
9         cmd.colorIndex = newColor;
10        paintedTiles.add(tile.id);
11        cmdList.addCommand(cmd);
12        let tessels = mosaicView.getTesselsAround(sketch.
   drawingContext, tile);
13        for (let i = 0; i < tessels.length; i++) {
14            if (selected) {
15                if (selectedTessels.includes(tessels[i],0)) {
16                    this.fillIn(tessels[i], originalColor,

```

```

mosaicView, newColor, cmdList, paintedTiles, sketch, selected,
selectedTessels);
17     }
18     } else {
19         this.fillIn(tessels[i], originalColor,
mosaicView, newColor, cmdList, paintedTiles, sketch, selected,
selectedTessels);
20     }
21 }
22 }
23 }

```

Listing 7.17: Recursive function created for the fill in tool

In listing 7.18 the function that allows to find the collisions between the bounding boxes is presented. Using a scale of 1.5 (can be changed directly in the code) we then search the *quadtree* data structure for any collisions that may occur with previously existing *quadtree* nodes (tiles), therefore finding returning all the tiles that are found. This then is used to color the tiles found using the implemented tool.

```

1 getTesselsAround(ctx, tessel) {
2     let offset = 1.5;
3     let tessels = [];
4     let colliding = this._quadtree.colliding({x: tessel.x - (
offset*tessel.w - tessel.w )/2, y: tessel.y - (offset*tessel.h -
tessel.h)/2, width: tessel.w * offset, height: tessel.h *
offset }).map(function(el){return el.tessel});
5     colliding.forEach(function (tile) {
6         if (tile !== null) {
7             tessels.push(tile);
8         }
9     });
10    return tessels;
11 }
12 }

```

Listing 7.18: Function created to find the tiles around a given tile

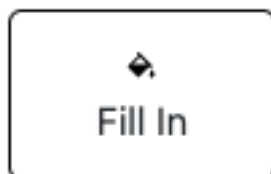


Figure 7.15: Button used to initiate the fill in operation

## Guidelines

This functionality doesn't count as a tool since it's behaviour is not directly related to the mosaic canvas. However, it also comes in handy when drawing a mosaic on a big grid or with a non grid layout as a way to have a better understanding of where the tiles to paint are related to each other and be able to paint a straight line.

To implement this feature two HTML elements were created. One represents the vertical guideline and another represents the horizontal guideline. These elements have functions that deal with the dragging actions, using the mouse coordinates, we can drag and drop the guidelines to the desired places. The existence of the guidelines does not impede the regular function of the drawing functionalities, therefore, the user can toggle the appearance of the guidelines on or off.

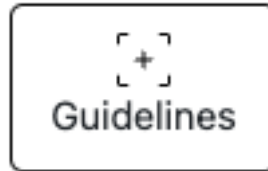


Figure 7.16: Button used to toggle the guidelines





# Chapter 8

## Evaluation

In this chapter, an overview of the overall platform and the requirements that were accomplished will take place. To do this, it is important to talk about the testing phase of the project. At the the end of this chapter, an overview of the referred functional requirements will be given. According to the quality attributes chosen for this project (Usability and maintainability) as well as all the functional requirements proposed to the platform several tests were conducted, namely usability tests and functional tests.

### 8.1 Functional Tests

Functional testing refers to the tests done in order to evaluate whether the functional requirements are correctly working. In this project, not only were the functionalities tested while they were being implemented, but at the end, the whole project was evaluated in terms of whether the functionalities worked as planned. To evaluate whether the functional requirements were correctly functioning, it was important to test all the capabilities of the platform.

With this objective, the authentication features were tested with correct inputs as well as incorrect inputs or partially incorrect (unfinished password, correct username wrong password). The platform behaved as supposed, meaning only correct authentication information was accepted and none other.

To create collections it was tested if a new entry in the database was created whenever a new collection was created. Similarly, it was tested if any mosaics upload to the platform registered not only in the database but also the server file system, which correctly occurs.

It was also tested if any user could access another users private collection. This was verified and all private elements of the platform have a verification of user id or user role, therefore not allowing any improper use of the platform.

To test the addition of mosaics from the mosaic editor, *Postman* was used as to create a form-data which would contain all the information that was desired to place in order to create a new entry in the database and the server file system

regarding a mosaic. Once all those processes were verified and indeed worked. The implementation took place and the tests occurred once again. According to the tests performed, any mosaic produced can now be sent to the database and the server file system immediately from the mosaic editor.

Regarding the Mosaic editor, the process was different, except for the operations of saving and checking user information and mosaic information, the tools had to be evaluated on if they did what they were supposed to. To test the draw shape functionality the squares drawn started from different points, meaning we could drag the mouse from the upper left corner of the rectangle until the lower right corner. But, it is also possible to draw the rectangles from any way the mouse is dragged, from the top to the bottom or bottom to the top, there is no difference, not even in the side where it starts (left or right) or where it ends.

The area selection worked in a similar fashion to the draw shape tool, at this point we are checking if once the mouse stops being dragged and the button is let go of, the tiles become selected, and that is visible when using the editor.

For the switch color tool the process was simple, the functionality in itself has a behaviour that can be understood easily. When clicking a tile, all tiles with the same color will have their color changed into the selected color. The testing of this feature was slightly more complicated when taking into account the possibility of having selected tiles. When there are selected tiles, if the user uses the switch color tool inside the selected tiles, only the selected tiles will be affected by the tool. On the other hand, if a user makes use of the switch color tool with selected tiles but does not press inside the selected tiles, the tool will work as normal.

For the fill in tool the testing was harder, using the new quadtree method, it is important to establish by how much is it supposed to increase the area of the tile bounding box, this required a try and error approach. Firstly we started by increasing the area by 2, doubling it. This led to the area being too great, and the tiles that intersected this new square being outside the range desired, therefore, it was needed to reduce how much we increased the area. With the value of 1.5, it is possible to achieve closer results to what would be expected with this tool. It is worth mentioning that using the factor of 2 to increase the area, the tool worked as expected on regular grids, however, in the grid shown in figure 7.14, it reached tiles outside a delimited area. With the factor of 1.5, it is much closer for the second case, while still working as pretended in the regular grids. To better understand, to use this tool with a non regular grid, it is important that the area to be filled in is very well delimited, sometimes painting a thicker boundary between the inside and the outside of the area to fill in is going to make a great change on whether only that area is painted or not.

## 8.2 Usability testing

Usability is related to how easily a new user can start using a platform. This is connected to how intuitive the platform is, how accessible the actions needed to perform are and how fast can a user learn to navigate the platform.

### 8.2.1 Procedure

To make sure the platform and the mosaic editor fulfilled the usability requirement, several tests were conducted with different people. This usability test was equal for everybody and the script used was created with the intent of the users making use of all the functionalities there are in the platform. The tasks included:

- Registering and Logging in
- Open a public mosaic of a specific public collection
- Remix that mosaic to create a given one (presented in figure 8.1)
- Save the newly created mosaic with a specific name
- Create a personal collection
- Add the newly created mosaic to the newly created collection
- Create a new mosaic from scratch

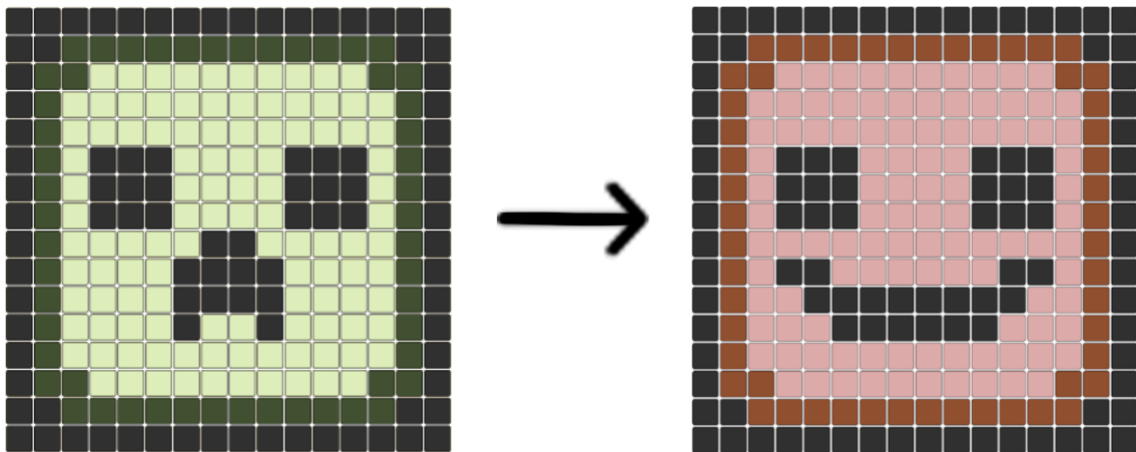


Figure 8.1: Mosaic on the left is the mosaic to be remixed into the mosaic on the right

The tasks planned for the user forces them to use every feature of the platform there is available for a common user. Furthermore, some questions needed to be answered during this tests, namely, what do the subjects interpret the objective of the platform to be, as soon as they enter it. What do they think they are seeing at any given moment of navigating the platform. What do they feel like the "Public" labeled button on the mosaic editor save menu mean?

When completing the several tasks asked for the usability testing, the participants were asked to fill out a form with metrics of usability of the platform.

Besides the first two questions that were placed for demographics purposes, all the other questions were presented with the possibility to answer with a scale of 1 (Strongly disagree) to 5 (Strongly agree). The responses gathered can be seen in the Results subsection 8.2.2. The questions asked were:

- Gender
- Age
- I think that I would like to use this platform frequently
- I found the platform unnecessarily complex
- I thought the platform was easy to use
- I think that I would need the support of a technical person to be able to use this platform
- I found the various functions in this platform were well integrated
- I thought there was too much inconsistency in this platform
- I would imagine that most people would learn to use this platform very quickly
- I found the platform very cumbersome to use
- I felt very confident using the platform
- I needed to learn a lot of things before I could get going with this platform

### 8.2.2 Results

According to the answers provided, every user had a good understanding of the platform's objective and what they were seeing since there is a short summary of the objective of the platform in the homepage. All of the participants understood immediately how to navigate through the platform and perform the tasks at hand.

The new mosaic every user drew from scratch, even though it didn't have a pre-defined end goal, had a general goal. The goal was to draw a house. From the several tests performed, all the houses were gathered in one collection presented in figure 8.2.

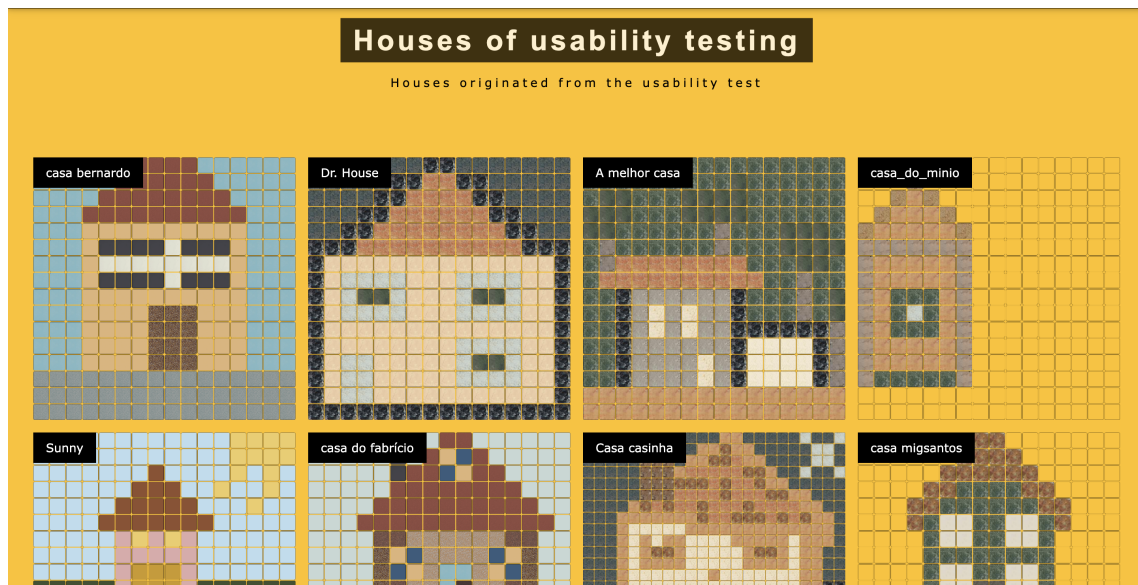


Figure 8.2: Collection containing all the house mosaics created from the usability testing

The realization of this usability testing also provided proof that all the implemented functionalities worked as supposed since all the tasks were completed without any difficulties. Furthermore, to add the mosaics to the public collection, the interviewer also had to make use of the admin functionalities, and since all the mosaics are displayed as they should, it can be inferred that all the capabilities on the admin side also work as supposed.

The answers provided in the form can be seen in figure 8.3 to figure 8.14.

Please specify your gender

10 responses

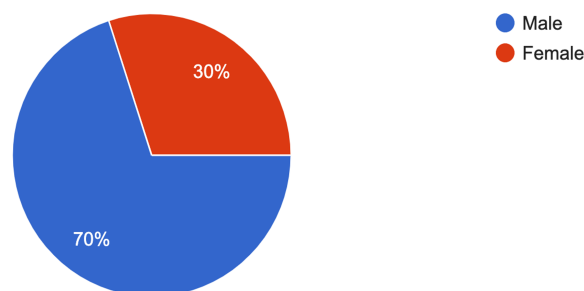


Figure 8.3: Gender of participants in the usability testing

From the 10 tests that were conducted, 3 of the participants identified as female, while the remaining 7 identified as male, as seen in figure 8.3.

Please specify your age

10 responses

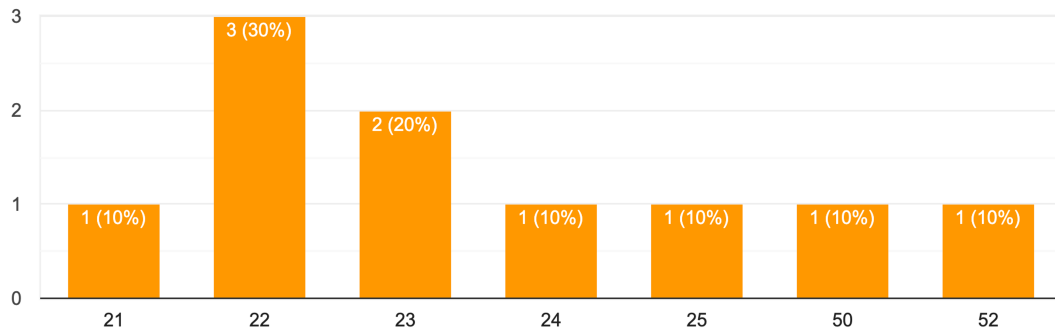


Figure 8.4: Ages of participants in the usability testing

The ages vary between 21 and 52 years old being that most of the participants are younger than 24 years old.

I think that I would like to use this platform frequently

10 responses

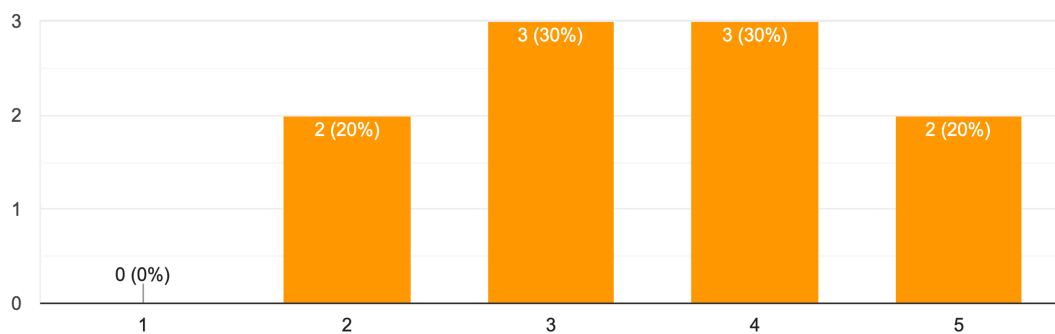


Figure 8.5: Answer to the first question of the form regarding the project

Since this platform is very specific in terms of its objective, it is not expected that all participants would see themselves using this platform frequently. However, some of the participants showed a great amount of interest in using this platform again in the future (figure 8.5).

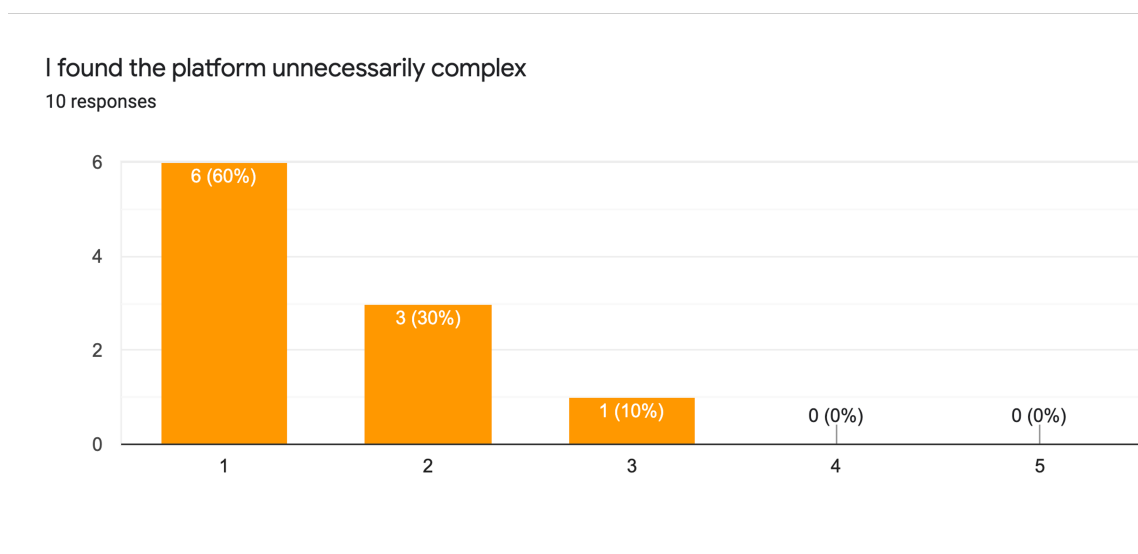


Figure 8.6: Answer to the second question of the form regarding the project

Some of the more important answers for this quality attribute relate to how easy the participants felt the platform was to use and how easily do they think a new user could start using the platform. Therefore, from all the answers provided, some provide some additional value for the usability purposes, namely, as seen in figure 8.6, no participant believed that the platform was unnecessarily complex. Most participants were strongly disagreeing with that statement, 3 were disagreeing and 1 participant was neutral.

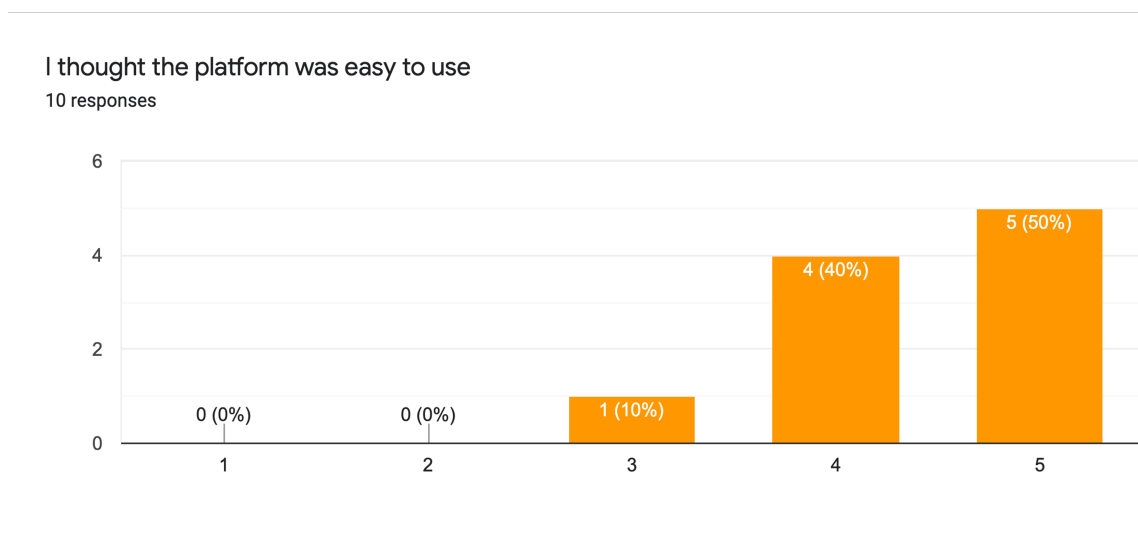


Figure 8.7: Answer to the third question of the form regarding the project

As seen in figure 8.7, all participants agreed that the platform was easy to use, excluding one participant that was neutral to the question.

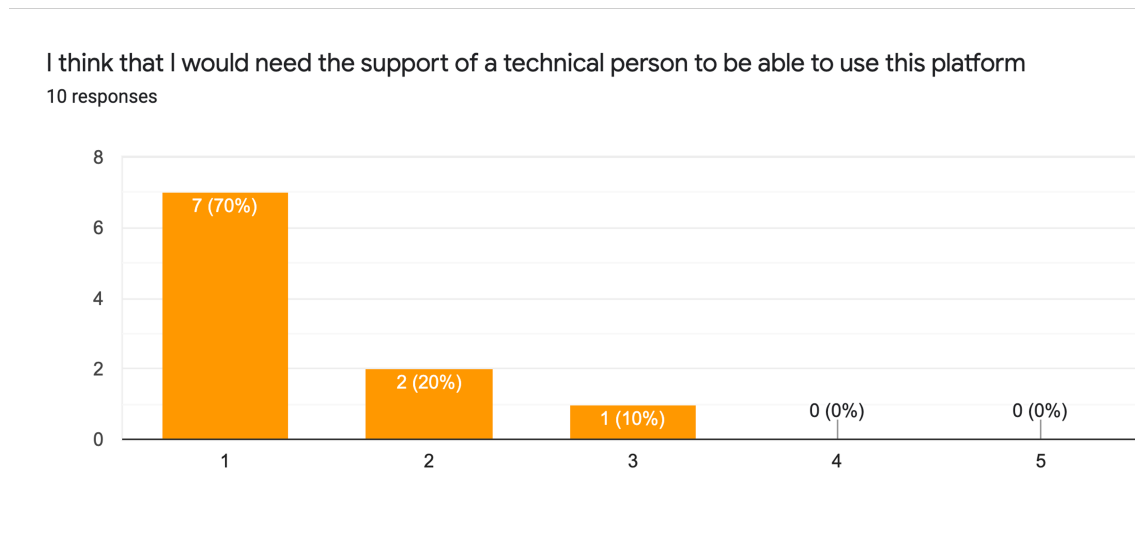


Figure 8.8: Answer to the fourth question of the form regarding the project

70% of the users completely disagree with the idea of needing support of a technical person to be able to use this platform, indicating that the functionalities were intuitive and they are able to do everything by themselves (figure 8.8). One of the participants remained neutral regarding needing support and 2 of the participants disagreed with needing support of a technical person to use the platform.

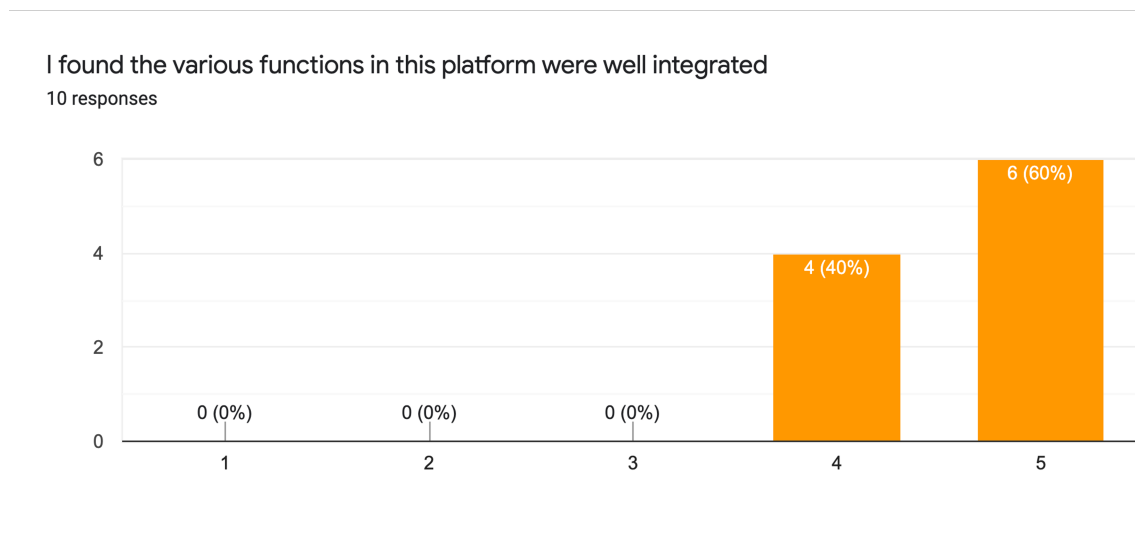


Figure 8.9: Answer to the fifth question of the form regarding the project

All of the participants found the various functions in this platform to be well integrated, being that 60% completely agreed with that statement and 40% agreed with the statement.



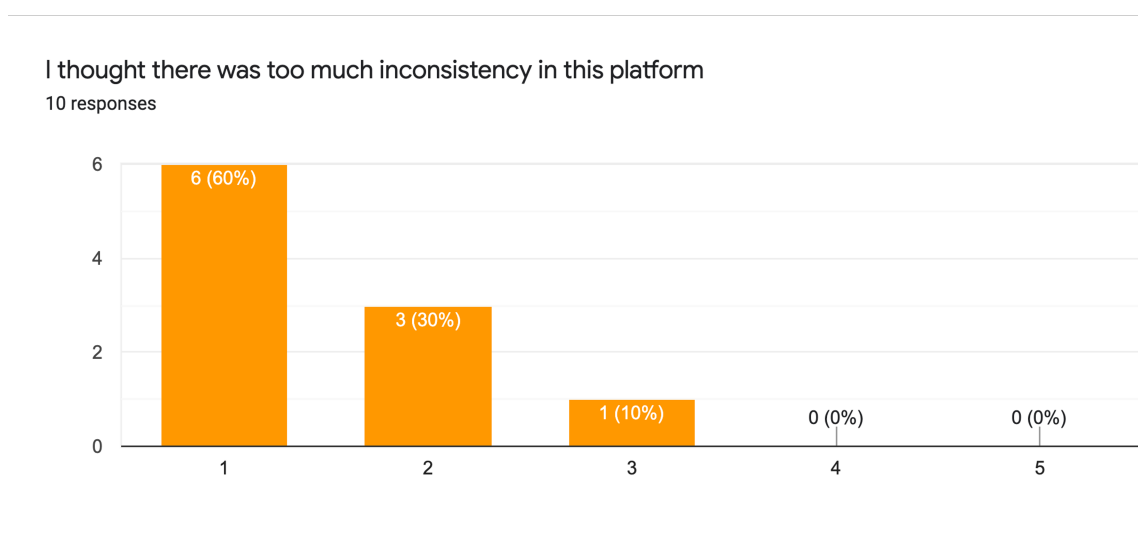


Figure 8.10: Answer to the sixth question of the form regarding the project

Regarding how consistent the platform was, one of the users remained neutral while 60% completely disagreed with there being too much inconsistency and 30% disagreeing with that statement.

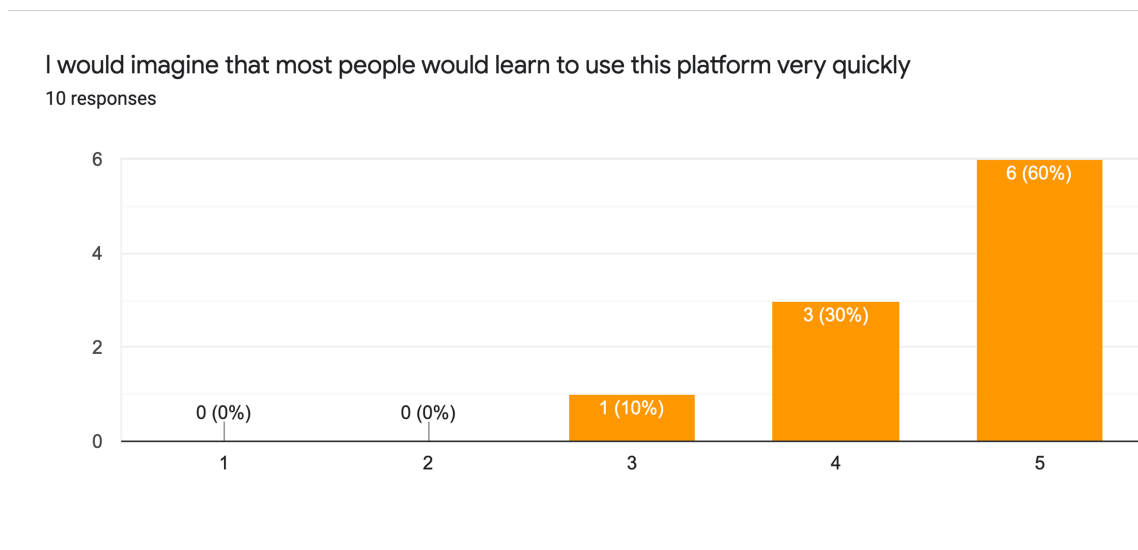


Figure 8.11: Answer to the seventh question of the form regarding the project

In figure 8.11 the answers provide insight into how the participants thought about how other people would use this platform. The majority agreed that most people would learn to use the platform very quickly, with exception for one participant that remained neutral.

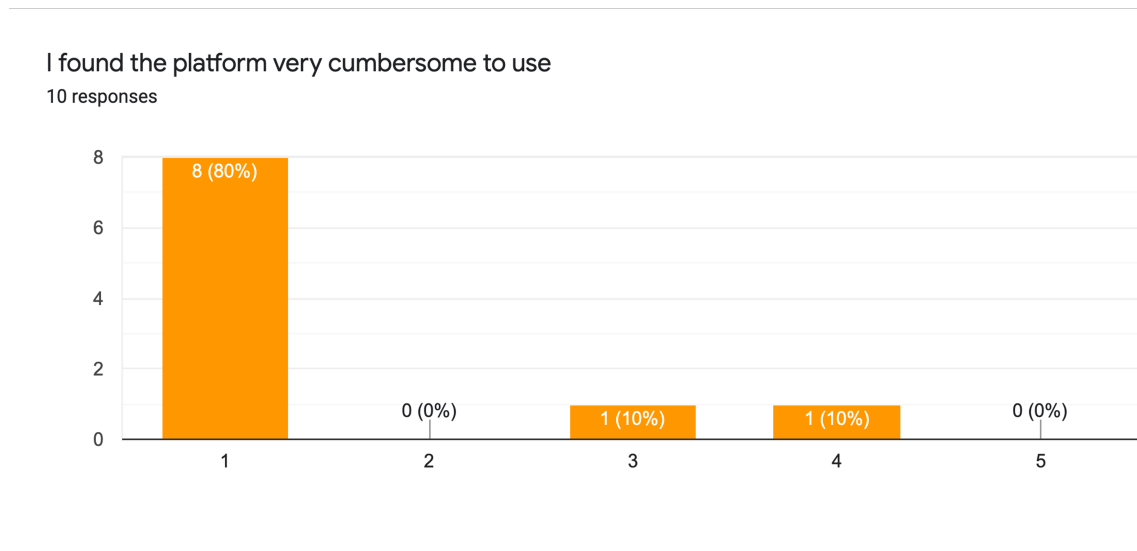


Figure 8.12: Answer to the eighth question of the form regarding the project

80% of the participants did not find the platform to be cumbersome to use, however one of the participants remained neutral and one other participant found the platform to be cumbersome to use. From the answers given so far, it isn't immediately apparent what could have been cumbersome, nevertheless, one of the participants found the platform to be cumbersome (figure 8.12).

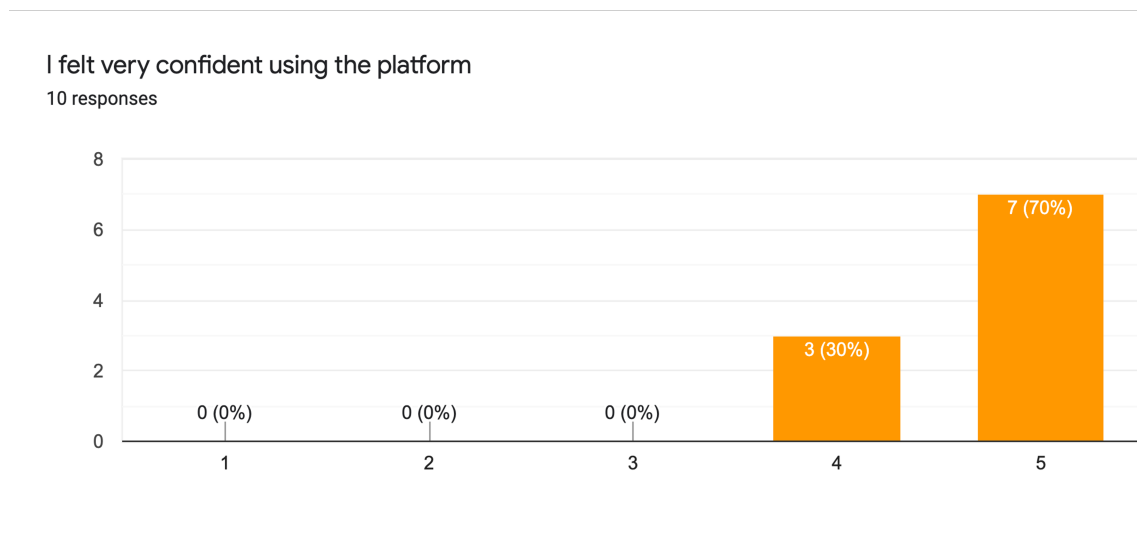


Figure 8.13: Answer to the ninth question of the form regarding the project

All of the participants felt confident using the platform which can be seen as a good sign meaning the platform and its functionalities are easy to understand as well as the process to accomplish a given task.

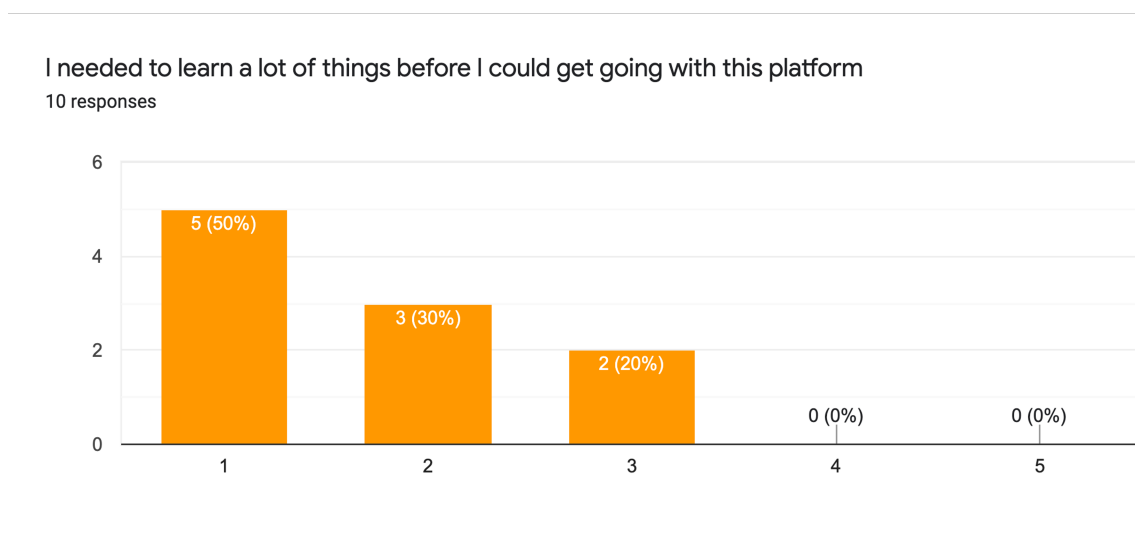


Figure 8.14: Answer to the tenth question of the form regarding the project

Lastly, most of the participants disagreed that they needed to learn a lot of things before they could get going with this platform, except for 2 of the participants that remained neutral.

Regarding the answers given during the usability test to the question "what do you interpret the Public checkbox does?", most users understood that it meant the mosaic was going to be saved as a public mosaic. However, certain participants had the idea that when a project is public, it would be immediately shown in the homepage of the Mosaic Sharing Platform. Even though only 2 out of the 10 participants gave that answer it is important that with future work a new solution is created that allows for a better understanding of how to save mosaics publicly or privately.

The times spent by each participant were also measured, this helps to better understand how fast do these new users navigate the platform and perform the tasks, the times obtained were:

- 7:05 minutes for the whole test (1:10 minutes to create a collection and add a mosaic)
- 6:10 minutes for the whole test (45 seconds to create a collection and add a mosaic)
- 6:54 minutes for the whole test (57 seconds to create a collection and add a mosaic)
- 6:09 minutes for the whole test (46 seconds to create a collection and add a mosaic)
- 6:50 minutes for the whole test (1:01 minutes to create a collection and add a mosaic)
- 7:00 minutes for the whole test (1:08 minutes to create a collection and add a mosaic)

- 6:01 minutes for the whole test (1:11 minutes to create a collection and add a mosaic)
- 5:57 minutes for the whole test (58 seconds to create a collection and add a mosaic)
- 6:25 minutes for the whole test (55 seconds to create a collection and add a mosaic)
- 5:48 minutes for the whole test (39 seconds to create a collection and add a mosaic)

As we can see, the times vary between 7:05 minutes and 5:48 minutes, this times include the time spent reading the script, the time spent by the participants drawing and choosing the colors as well as the participants time spent picking descriptions for their mosaics and collections. This times do not account with the time spent filling the form provided in the end.

The average time achieved is approximately 6:43 minutes. Taking into account the extra times that were accounted that were not directly related with the fulfillment of the tasks at hand as well as the number of tasks performed, the users were quick to navigate the platform and executing the tasks at hand.

### 8.2.3 Discussion

From the results seen, some conclusions can be gathered, namely, since this is a very specific platform, probably not all participants would see themselves using it frequently (figure 8.5). However, this does not take away from the fact that it is easy to use, as seen in the remaining questions. An analysis on the time taken to perform all the tasks that were proposed, leads to the point that since the time taken to accomplish all the tasks was short, the tasks must have been easy to perform.

Furthermore, as previously mentioned in the Usability quality attribute section 5.6.1, the users should take less than a minute to create a new collection and add a mosaic to that collection. From the usability tests performed, the participants took in average 57 seconds to perform this task. Taking into consideration the time spent on naming the new collection and writing the description, the time spent should go down by at least 10 seconds. This means that the average time spent on creating a new collection and adding a mosaic is 47 seconds, which is below one minute. Therefore, it is assumed that this quality attribute was accomplished with success.

This gives us a conclusion that the usability aspect of the project was achieved. Even though some users didn't understand, as previously mentioned, exactly what the "Public" checkbox does, the overall flow of all the participants through the platform was intuitive and fast, leading to the conclusion that the usability factor is assured.

## 8.3 Maintainability

The other quality attribute that was important to be respected in this project is the maintainability aspect. This relates to the qualities of being extensible and flexible. This entails that a given project can be altered in the long run without running into issues related to how is the code structured and how are the functionalities implemented. Furthermore, this also means that in the future, another developer can add new features to the project, being the Mosaic Sharing Platform or the mosaic editor.

The question that remains is how does this quality attribute was achieved?

### 8.3.1 Mosaic Sharing Platform

For the Mosaic Sharing Platform the process of evaluating the simplicity of the code is done by directly looking into the project folder, represented in figure 7.1.

As is seen, the web pages(PHP files) are all on the same level. In folders we have the CSS folder containing all the style elements needed for the platform, the images that contain static images that will be present in the platform at all times. Furthermore we have the lang folder containing the language files and the Mosaic folder containing all files that represent a mosaic (JSON and PNG).

If a new developer was asked to add a new language to the project, they would just need to add a file to the lang folder and translate the words present in the other two folders to the new language.

If any frontend changes need to happen, localizing the web page that needs to be changed can be done instantly by looking at the file names. This ensures that a new developer can track down to the point where they wish to insert new elements.

Now regarding the backend changes that might need to happen, a file with the name "functions.php" contains all the functions that communicate with the backend. The names of the functions were carefully picked in order to be understood immediately what a given function does. Adding a new function is simple and changing an existing is simple as well since all the functions relate to Read/Write operations to the database.

### 8.3.2 Mosaic Editor

For the Mosaic Editor, it is important that future developers can add new drawing/painting functionalities easily. With this concern in mind, the new architectural design pattern used for the functionalities implemented during this semester is perfect. From now on, every time a new tool needs to be added to the editor, the process consists in creating a new file with a class related to the tools class. Everything a new developer needs to do is create a file, understand the tool class

since their new tool will extend that class, implement the method that provides the new tool functionality and a new tool has been created. All the other steps needed to actually make it 100% functional are steps that cannot be changed due to the overall architecture of the project. Steps such as:

- Create a new HTML button for the new tool
- Add the tool to the configuration file
- Add the tool to the sketch class which initiates the command and the tool for a new button
- Creating the command class for the button that will *setActive* the new tool

Are steps that need to be performed regardless of the new functionality as long as it needs a button to work.

By having a simple architecture and a clear division of where the code is for the Mosaic Sharing Platform as well as having added a Chain of Responsibility design pattern to the mosaic editor architecture, the process of adding new features and maintaining this platforms is simple and straightforward. With that conclusion, it is safe to say that the maintainability aspect of this project has been achieved.

## 8.4 Accomplished requirements

At the end of this semester, it is important to look back at what was accomplished from the predefined requirements for the project. This said, in table 8.1 it is possible to see all the functional requirements for this project, highlighted in green are the requirements that were actually implemented.

As is visible in the table, all requirements with the highest priority (Must) were implemented with the exception of the "Draw Elliptic Shape". This tool was not implemented simply for the reason that with the time left for the implementation of the features for the Mosaic editor, the intern thought to be a better opportunity to implement a different tool instead of another shape drawing tool. Therefore, the changing of color occurrences was implemented instead.

In terms of the Mosaic Sharing Platform, all the requirements were implemented, independently of their priority with the exception of using Google or Facebook for authentication purposes. Furthermore, two other features were implemented that were not part of the initial plan. These features are the ability to change the language of the platform and the ability for the user to change their password.

This leads to a conclusion that the project was a success, since the new platform conforms with all the desired requirements. For the mosaic editor, a new architectural pattern was implemented letting future developers easily add new functionalities. Therefore making any future work much easier to carry.

Theme	Requirement	Role	Priority	
Platform	Login and Register Interface	All	(M)	
	Login with Google	All	(C)	
	Login with Facebook	All	(C)	
	Login	All	(M)	
	Register	All	(M)	
	Save Mosaics to Server File System	All	(M)	
	Attribute title to mosaic	All	(M)	
	Share Mosaics	All	(M)	
	Admin Page to create collections	Admin	(S)	
	Create collection	Admin	(S)	
	Attribute description to collection	Admin	(S)	
	Admin Page to add curators to collections	Admin	(S)	
	Attribute roles	Admin	(S)	
	Interface with all public mosaics	Admin/Curator	(S)	
	Collection Interface	All	(S)	
	Add mosaic to collection	Curator/Admin	(S)	
	Private collection page	All	(C)	
	Create private collection	All	(C)	
	Interface with user's private collections	All	(C)	
	Homepage with collections	All	(S)	
	Change the language of the platform	All	(C)	
	Change password	All	(C)	
	Suggest mosaics to add to a collection	Curator/Admin	(C)	
	Open previously built mosaic	All	(M)	
	Editor	Draw Rectangular Shape	All	(M)
		Draw Elliptic Shape	All	(M)
Select tiles by color		All	(S)	
Change color occurrences to another color		All	(S)	
Rectangular Area selection		All	(M)	
Free hand Area selection		All	(W)	
Fill in areas with a certain color		All	(M)	
Insert horizontal guideline		All	(M)	
Insert vertical guideline		All	(M)	
Toggle Borders		All	(C)	
Generate mosaic from image		All	(W)	
Tesselate (from TileMaker 3.1.3)		All	(C)	
Notate on tiles/group of tiles		All	(C)	

Table 8.1: Requirements accomplished from the already existing requirements table





# Chapter 9

## Conclusion

In this final chapter, we will discuss the work done during this year, the challenges found along the way, the lessons learned from the challenges. Furthermore, some suggestions will be given to further improve the mosaic editor and/or the Mosaic Sharing Platform. At the end, some final remarks will be given about the internship and the project.

### 9.1 Work Done

During this year, a lot of work was put into creating a new platform as well as adding new functionalities to an existing one.

During the first semester, the work was focused on preparing for the implementation phases. A study on the existing system's architecture was conducted, several platforms that related to the objectives and requirements of this project were analysed and the requirements were defined. Some work was allocated into actually implementing a new feature. The database model for the project was drawn out and a mock up of the homepage for the future platform was also designed. These steps allow for a compensation of work that won't need to take place during the second semester.

During the second semester, the work was focused on creating a functional new platform that allows users to share the mosaics they develop as well as add functionalities to the existing mosaic editor.

When creating the new platform, some studying took place as how to achieve the mock up designed in the first semester. Once that step was taken and the implementation began, the developer became more and more comfortable with HTML. This helped throughout the project since different web pages needed to be created, sometimes a web page that was already created needed to be altered in some way. On the server side of this project, the connection with the database was created which then was changed to a remote database on MosaicoLabs' server. Several functions were written that allowed for easily read and write information to the database.

For the mosaic editor, a new architectural model was studied and put into practice with the Chain of Responsibility. Several tools were also implemented to allow the users to easily and quickly perform actions that previously would demand painting every single tile to get the same result.

The final product of this internship complies with the success criteria defined in the threshold of success 4.2.6. Since all the higher priority requirements for the Mosaic Sharing Platform were implemented and more, as well as the new architectural model being created and several new tools being implemented in the mosaic editor. From the tests performed which validated the functionality of the tools and the platform as well as the usability tests. It is safe to say that the platform's usability was assured. Through the architecture and the code structure of the entire project, it becomes apparent that adding or changing the code base developed through this project should not pose a problem.

## 9.2 Challenges

Some of the challenges associated with this project were mostly associated with the mosaic editor. Working with a code base that already exists makes it difficult to figure out where to apply the needed changes. Once the code base was understood and the behaviour of the editor was clear, changing the code base didn't feel as threatening.

One of the challenges was how to convert the idea of the Chain of Responsibility design pattern into code. An architectural model makes sense on paper, but until it is implemented, it can be quite frustrating imagining how it is actually supposed to be implemented. However, through some studying and meetings with the supervisor, it was achieved.

Another challenge was understanding how quadtree works. When implementing the Fill in tool, the developer thought the tool to be completed with the first iteration, however that solution did not work for all cases. Therefore, the need for a new solution involving quadtree arose. Once again, to achieve the new solution, the quadtree data structure was studied as to better understand how it works. Once it was clear, the implementation was done.

The last challenge to be mentioned regarding the mosaic editor is the use of the *fetch* API. This API was used to communicate with the Mosaic Sharing Platform, since the intern was firstly not aware of the existence of this API, it took some time to fully understand how it works. Once it was understood, that API was used several times without difficulty.

Regarding the Mosaic Sharing Platform, some challenges arose as well. Namely, the need to incorporate language files, that would allow the user to change the language of the webpage. Even though it was easy to understand how it was supposed to be achieved, it was still a demanding task as it required to remove all the static text from all the already created webpages and creating two different files with the translation for those texts.

These challenges however, provided a lot of learning opportunities, not only on the available technologies but also regarding problem solving skills. By learning how to use the *JavaScript* fetch API, the developer can now use this in future projects. The process of thinking and reaching a solution for the painting tools was a very satisfying and motivational process. Learning how to put into practice architectural models of software engineering is also very important for future works and to better understand the code structure from an image of the architecture.

### 9.3 Suggestions

After working on this project for a year, some ideas regarding what could be implemented moving forward came up. These ideas are not only regarding developing new features but also regarding possible tests that could be conducted.

One of the possible features to be added to the project was to allow the user to select a tile, group of tiles or even the whole mosaic and create a comment on them, as one would on a Google Docs. Even though that feature wasn't implemented, in the future, if it is implemented, a very interesting feature to add would be to create a convolutional neural network that would analyse a mosaic, detect patterns and provide a comment regarding said pattern. A feature like this wouldn't be easy to implement, however, the users would learn a lot from mosaics. If a user draws a mosaic and then receives information regarding a pattern they drew (possibly by accident), it would be a very interesting learning experience that could lead to a greater interest on mosaics.

Some of the proposed requirements that weren't implemented in this project can also be taken into account when thinking of new features to add not only to the mosaic editor but also the Mosaic Sharing Platform. Features such as using Facebook or Google for authentication instead of the regular e-mail, username and password registration and logging in. The capability of generating a private URL to send to friends of a private mosaic drawn by a certain user could also be very interesting to see as it would allow for a wider spread of the platform to people that haven't heard of it.

To better test if the maintainability quality attribute was successfully achieved, some tests can be made in the future, where developers that are not familiar with how the editor or the sharing platform work are called and asked to fix a given bug (placed on purpose for them to find and correct).

### 9.4 Final Remarks

This project allowed a much needed growth in terms of experience and how to look at problems in an engineering perspective.

Through the requirements that were implemented, the users that tested and used

the platform to the personal growth obtained for this project. It is safe to say that this project was success.

# References

- Introduction to blockly | google developers, 2020. URL <https://developers.google.com/blockly/guides/overview>.
- Studio - scratch wiki, 2022. URL <https://en.scratch-wiki.info/wiki/Studio>.
- Muhammad Ovais Ahmad, Jouni Markkula, and Markku Oivo. Kanban in software development: A systematic literature review. pages 9–16. IEEE Computer Society, 2013. ISBN 9780769550916. doi: 10.1109/SEAA.2013.28.
- Catherine Balmelle, Cnrs Paris-Fransa, / France, Jean-Pierre Darmon, Maria De, and Fátima Abraços. Aiema-tÜrkİye scientific committee / bilimsel komite. 2018. URL [www.onikincimatbaa.com](http://www.onikincimatbaa.com).
- H. Frank Cervone. Understanding agile project management methods using scrum, 2 2011. ISSN 1065075X.
- Rogelberg David. 20 top experts share their secrets to agile making the transition to agile or a mixed methodology approach, 2014.
- Dinnie. 7 popular project management methodologies, 2018. URL <https://zenkit.com/en/blog/7-popular-project-management-methodologies-and-what-theyre-best-suited-for/>.
- H. Figueiredo. Festa do mosaico 2020, 2020. URL <https://mosaicolab.pt/pt/2020/12/04/festa-do-mosaico-2020-2/>.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns Elements of Reusable Object-Oriented Software*. 2009.
- Aditya Kamath. Quad tree - geeksforgeeks, 2018. URL <https://www.geeksforgeeks.org/quad-tree/>.
- Jakob Nielsen. Usability 101: Introduction to usability, 1 2012. URL <https://www.nngroup.com/articles/usability-101-introduction-to-usability/#:~:text=Usability%20is%20a%20quality%20attribute,use%20during%20the%20design%20process>.
- Ekaterina Novoseltseva. Software architecture quality attributes - dzone performance, 6 2020. URL <https://dzone.com/articles/software-architecture-quality-attributes>.
- Alexander Sergeev. Roles in waterfall methodology, 2 2016. URL <https://hygger.io/blog/team-roles-in-waterfall-methodology/>.

## *Appendix*

---

Vaibhav Singh. Should i use db to store file ?. are files meant to be stored inside db... | by vaibhav singh | medium, 7 2019. URL <https://medium.com/@vaibhav0109/should-i-use-db-to-store-file-410ee22268c7>.

Annelisa Stephan. A brief introduction to roman mosaics, 2016. URL <https://www.getty.edu/news/a-brief-introduction-to-roman-mosaics/>.

# Appendices





# **Appendix A**

## **Mosaic Editor Software Architecture**

