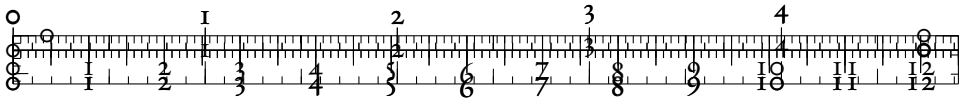
A Collaborative Visual Database

by

Imed Adel

ESSTHS



Contents

1 Analysis and specification of needs **1**

1.1 Analysis of requirements 1

1.1.1 Functional requirements 2

1.1.2 Non-functional requirements 3

1.2 Specification of needs 4

1.2.1 Identification of actors 4

1.3 Use case diagrams 5

1.3.1 General use case diagram 5

1.3.2 “Sign up” use case diagram 5

1.3.3 “Log in” use case diagram 7

1.3.4 “View resources” use case diagram 7

1.3.5 “Edit resources” use case diagram 8

1.3.6 “Manage users” use case diagram 10

1.3.7 “Manage workspace” use case diagram 10

1.3.8 “Authentication and authorization” use case diagram 11

1.4 Wireframes 13

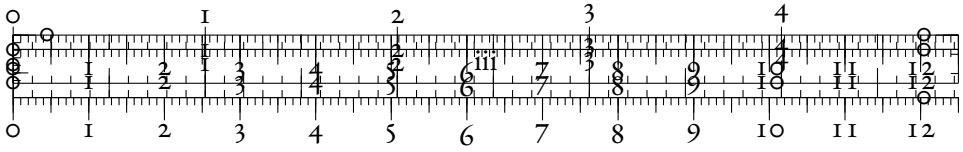
1.4.1 “Projects” wireframe 13

1.4.2 “Collection” wireframe 14

1.4.3 “Document” wireframe 15

1.4.4 “Editing panel” wireframe 15

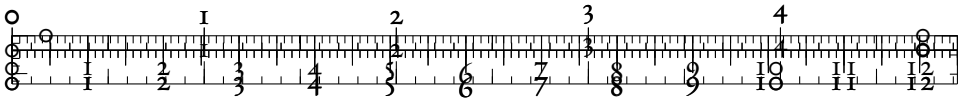
1.5 Conclusion 16



Acronyms 17

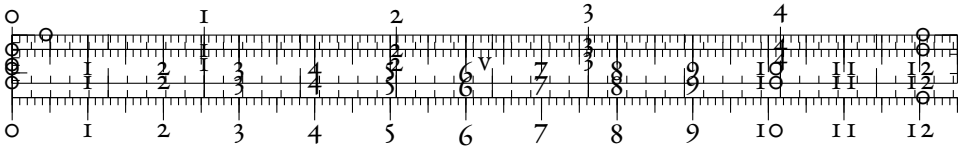
Glossary 19

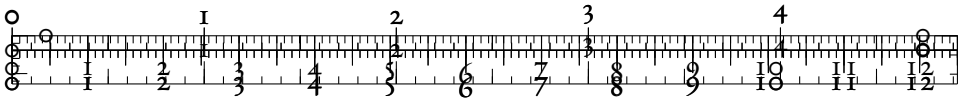
Bibliography 20



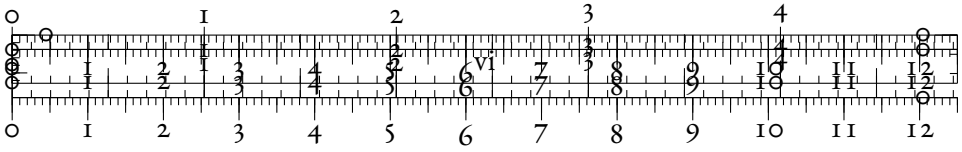
List of Figures

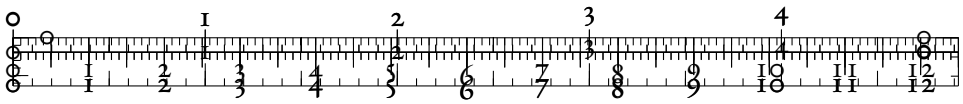
1.1 General use case diagram 6
1.2 “Sign up” use case diagram 7
1.3 “Log in” use case diagram 7
1.4 “View resources” use case diagram 8
1.5 “Edit resources” use case diagram 9
1.6 “Manage users” use case diagram 10
1.7 “Manage workspace” use case diagram 11
1.8 “Authentication and authorization” use case diagram 12
1.9 “Projects” wireframe 14
1.10 “Collection” wireframe 14
1.11 “Document” wireframe 15
1.12 “Editing panel” wireframe 16





List of Tables





I Analysis and specification of needs

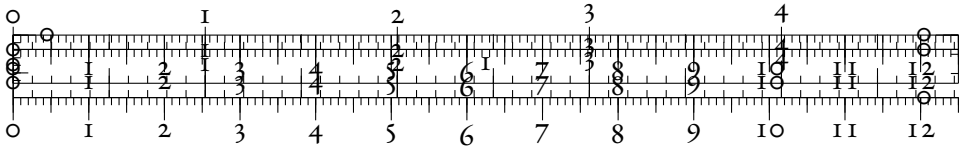
The development process of any application or system requires taking into consideration the needs of the user and their main objectives of using the application.

After researching the current solutions and their drawbacks, we set—within this chapter—to analyze the functional requirements of our users, and therefore our different system actors and their use cases.

1.1 Analysis of requirements

The goal of our project is creating a collaborative visual database that allows users to easily manage their data, be it a list of users, blog posts, or a store inventory, and to easily and securely access this data through an API. Our application is targeted at enterprises and individuals with no or minimal technical skills—a goal that should be kept in mind while structuring our project.

The analysis of requirements phase aims to list a set of both functional and non-functional requirements for modeling and developing our application.



1.1.1 Functional requirements

Functional requirements specify the different tasks of a system. Therefore, in this section, we set the exact tasks that our application must be able to successfully handle, from the most general to the most specific.

- A user must be able to collaborate with other users on the same document at the same time, in real-time.
- A user must be able to organize and share multiple documents at once.
- A user must be able to create multiple workspaces.
- Workspaces must be organized in the following way:
  - Each workspace contains multiple projects
  - Each project contains multiple collections
  - Each collection contains multiple documents
- A user must be able to grant different permissions to different users.
- A user must be able to access their documents and collections through an API endpoint.
- A user must be able to set a defined structure for their documents.
- A user must be able to set predefined filters and queries for each collection, also called a “view”.
- A user must be able to secure access to their API endpoints.
- The API must support real-time updates.
- A user must be able to reference other documents.
- A user must be able to add rich text to documents.
- A user must sign up and login.

- A user's account picture must be fetched automatically from Gravatar
- A user can upgrade their account to a premium one
- A user can cancel their premium subscription

1.1.2 Non-functional requirements

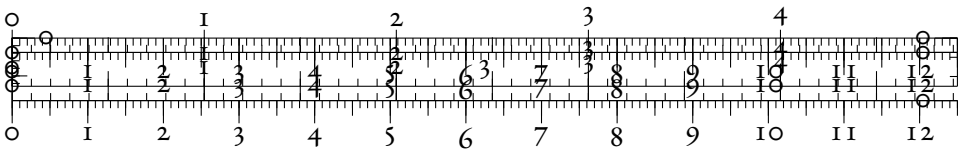
Non-functional requirements define *how* a system performs its various tasks. The goal is to offer the best user experience.

Security Our application should ensure the security of the hosted data. It should respect the permissions and roles set by the user. Therefore, a robust authentication and authorization system must be put in place.

User experience Our goal is to offer the best user experience achievable. Our users will not have the technical knowledge to understand a technically complicated application, and they do not have much time to accommodate themselves with a completely new set of interfaces. Therefore, our application must be simple and familiar.

Speed Our users may not have the best internet connectivity, and they might be sharing a single internet connection to accomplish their work. Therefore, our application must load within seconds. This includes caching resources, minifying them, and predicting and preloading what the user is going to need next.

Performance Our application has to load and display large amounts of data. This can result in an undesirable glitching and huge memory and CPU usage, which, aside from the slowness of the application, increases the temperature



and noise of the computer. Therefore, resulting in an uncomfortable user experience, especially when working within groups.

Accessibility Our users might have some preferences when it comes to navigating the interface, such as relying on the keyboard rather than using the cursor. Other users might rely on different input devices that require special care. This is an important point to consider in order to render our application usable by as many people as possible.

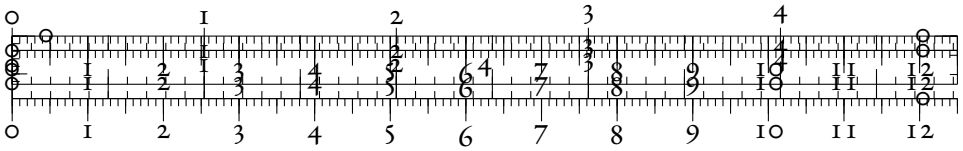
Scalability Our application must be developed with scalability in mind. This includes having to increase the number of servers while maintaining the collaborative aspect of the application.

Developer experience Our application must use state-of-the-art technology to offer the best developer experience. This includes using linting and formatting tools, along with deployment platforms such as Docker.

1.2 Specification of needs

1.2.1 Identification of actors

Merebase uses Role-Based Access Control (RBAC) to manage users' access levels and permissions. The role defines what actions the user is allowed to execute. For the time being, it will be assigned per workspace. Based on the discussed requirements, the way our application handles permissions, and the manner in which our services interact, we can identify a set of actors for our use-case models. An actor specifies a role played by a user or any other system that interacts with the application.



Visitor They are an unknown user to our application. They can either sign up or log in.

Viewer They can view documents in the workspace without being allowed to modify them.

Editor Along with viewing documents, they can also edit them.

Admin They are editors with elevated privileges: along with viewing and editing documents, they can invite new users and assign roles.

Owner They are the creator of the workspace and they have the similar privileges to admins. They own the workspace and they can delete it.

1.3 Use case diagrams

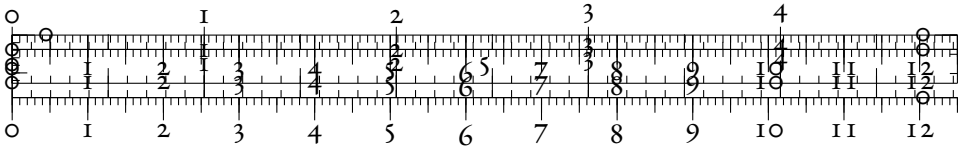
Use case diagrams represent a more in-depth analysis of users' interaction with the application by highlighting the different services and functionalities. Therefore, in this section, we will explore the interactions between our application's different components and services, and the user.

1.3.1 General use case diagram

Diagram 1.1 describes the general use case of our application. In the following diagrams, we will further analyze and dissect each use case.

1.3.2 "Sign up" use case diagram

Diagram 1.2 describes the "sign up" use case of our application.



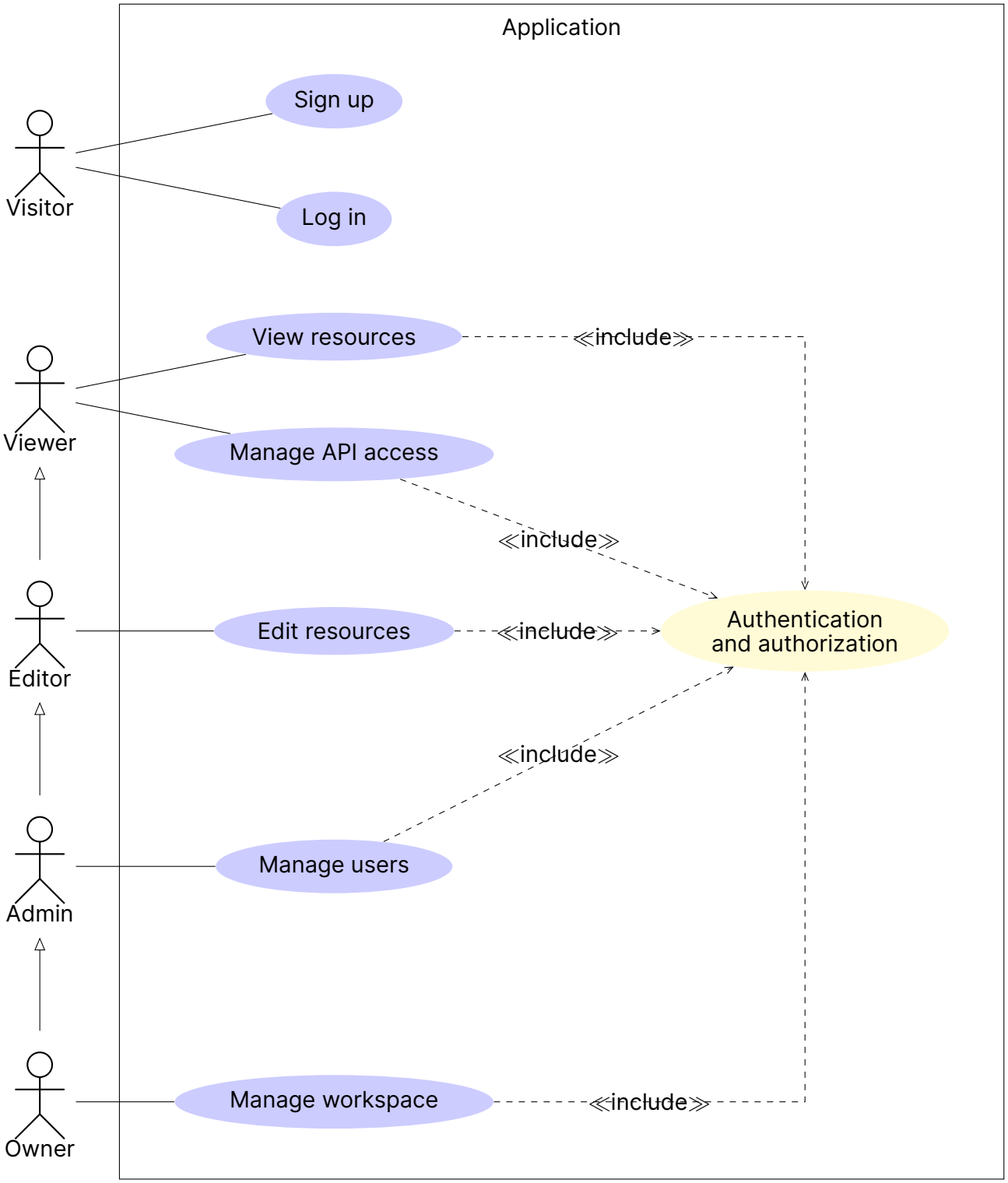


Figure 1.1: General use case diagram

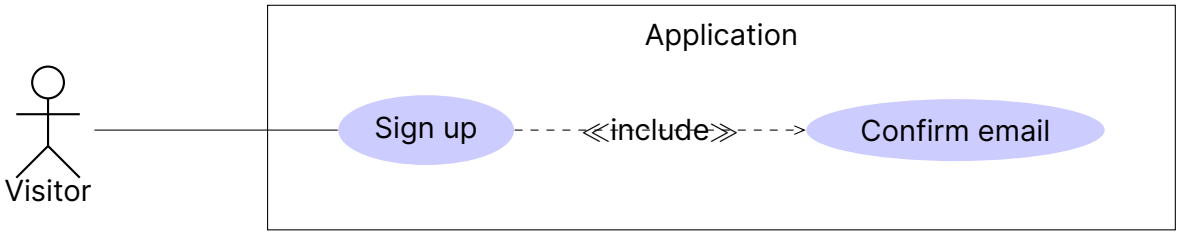


Figure 1.2: “Sign up” use case diagram

1.3.3 “Log in” use case diagram

Diagram 1.3 describes the “log in” use case of our application.

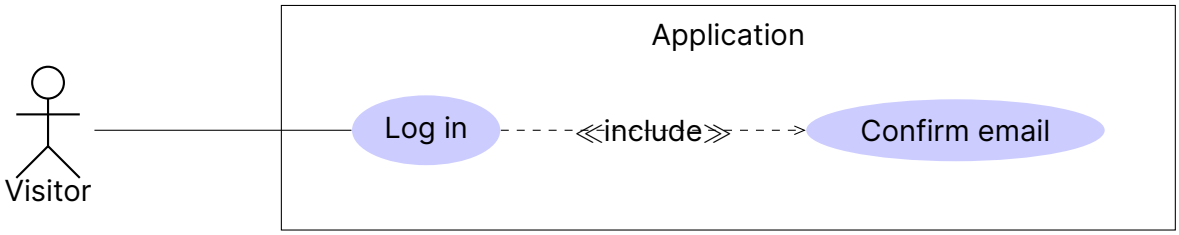


Figure 1.3: “Log in” use case diagram

1.3.4 “View resources” use case diagram

Diagram 1.4 describes the “view resources” use case of our application.

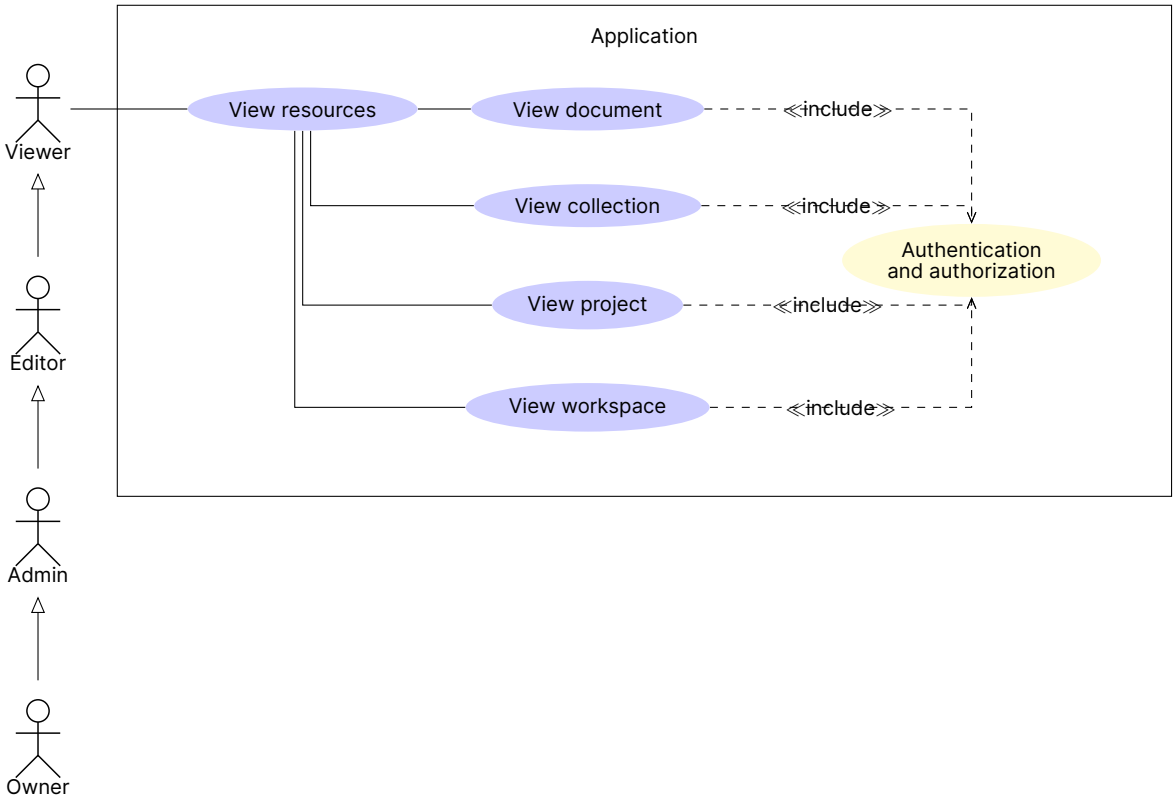


Figure 1.4: “View resources” use case diagram

1.3.5 “Edit resources” use case diagram

Diagram 1.5 describes the “edit resources” use case of our application.

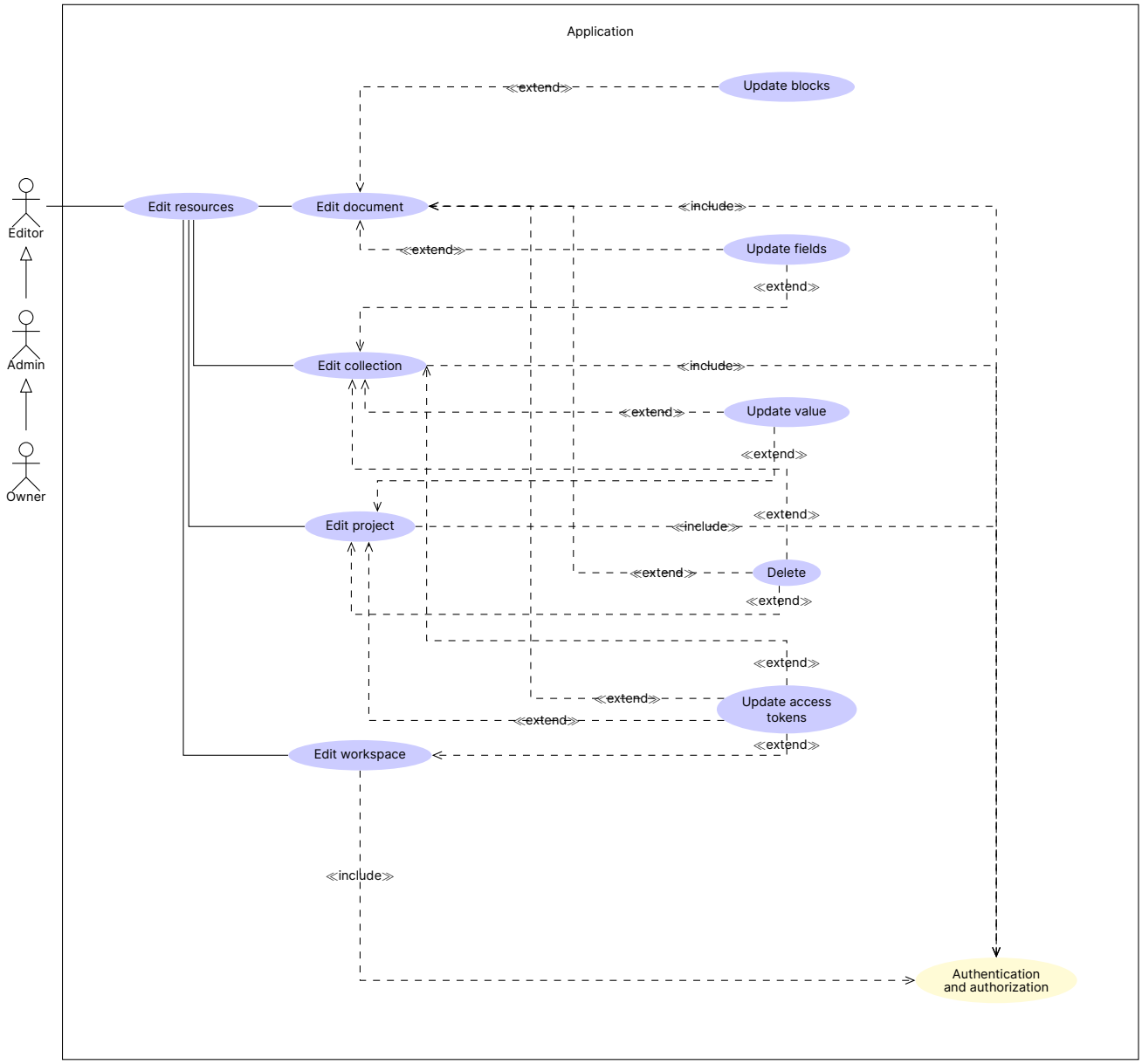


Figure 1.5: "Edit resources" use case diagram

1.3.6 "Manage users" use case diagram

Diagram 1.6 describes the "manage users" use case of our application.

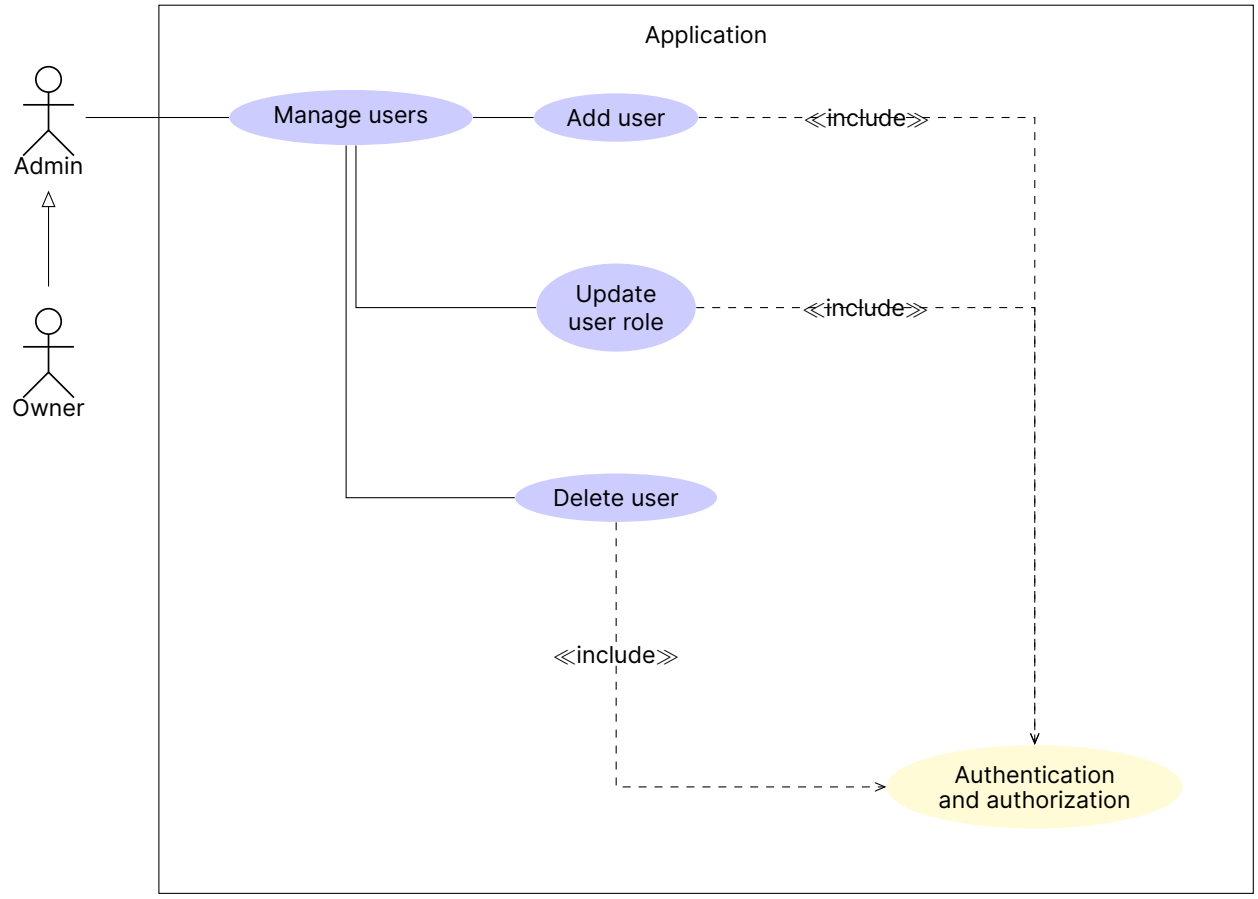


Figure 1.6: "Manage users" use case diagram

1.3.7 "Manage workspace" use case diagram

Diagram 1.7 describes the "manage workspace" use case of our application.

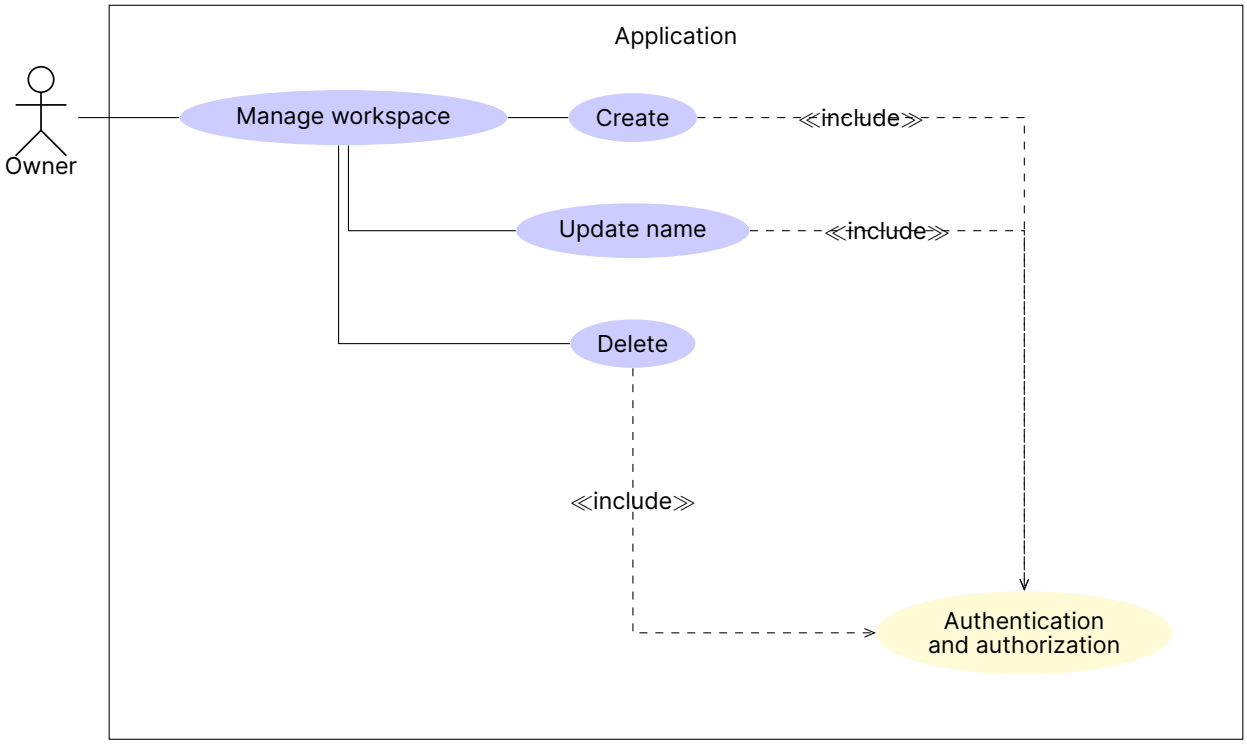


Figure 1.7: “Manage workspace” use case diagram

1.3.8 “Authentication and authorization” use case diagram

Diagram 1.8 describes the “authentication and authorization” use case of our application.

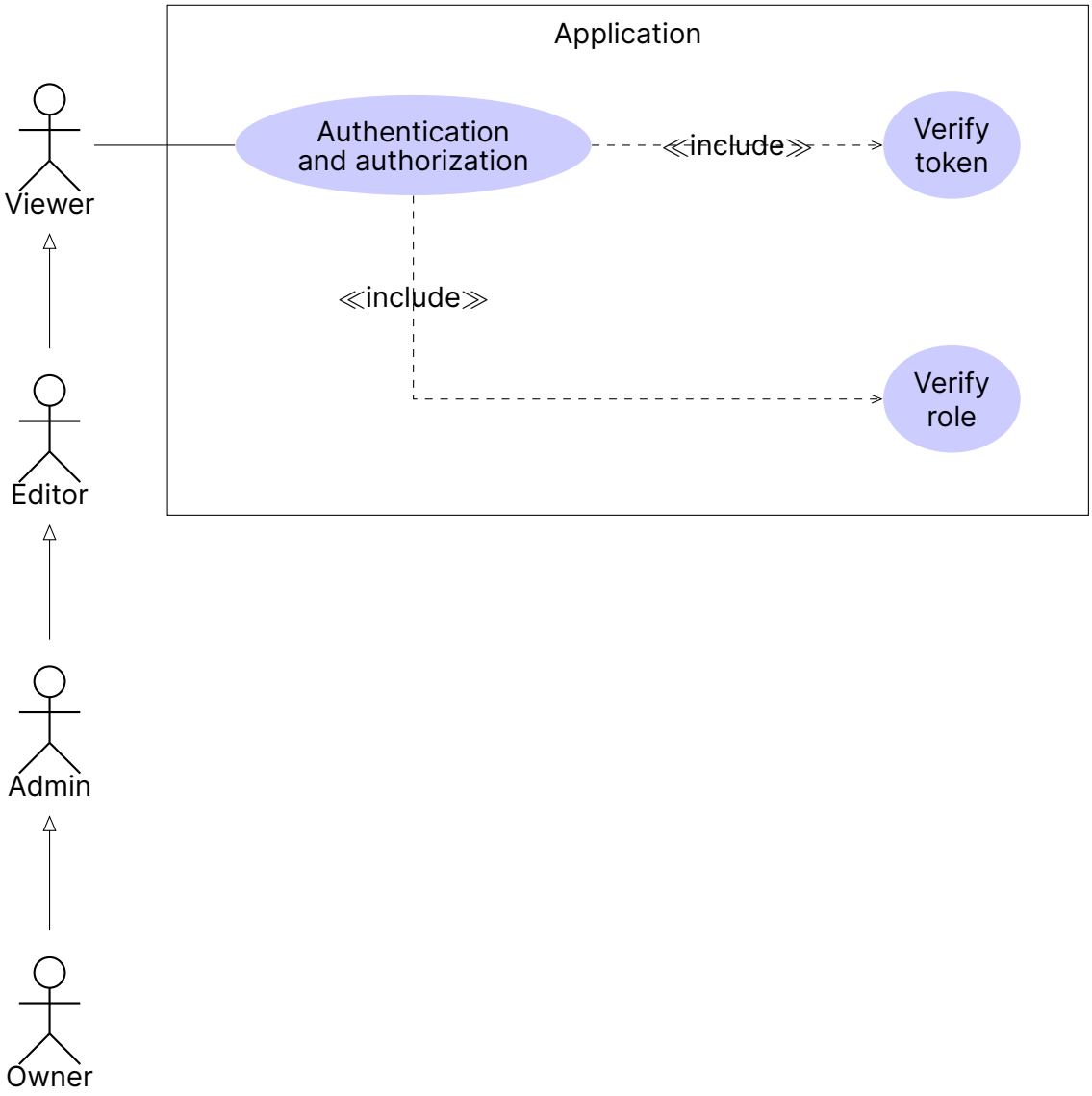


Figure 1.8: "Authentication and authorization" use case diagram

1.4 Wireframes

Use case diagrams provide us with an in-depth technical view of the possible interactions between our application and the user. However, they are too abstract and they fail in helping us imagine the format of the application and its interface. Furthermore, this abstraction prevents us from noticing the missing components of our architecture. Therefore, we use wireframes to better understand the User Interface (UI) and to look for any shortcomings in our analysis.

Wireframes are simple visual guides representing the skeletal framework of a website [3]. They depict the page layout or arrangement of the website's content, including interface components and navigational systems, as well as how they operate together. Since the major focus is on functionality, behavior, and content prioritization, the wireframe usually lacks typographic style, color, or images [2].

1.4.1 "Projects" wireframe

Figure 1.9 shows the wireframe of the projects page.

Figure 1.9: ”Projects” wireframe

1.4.2 “Collection” wireframe

Figure 1.10 shows the wireframe of a collection page.

Figure 1.10: ”Collection” wireframe

1.4.3 “Document” wireframe

Figure 1.11 shows the wireframe of a document page.

Figure 1.11: ”Document” wireframe

1.4.4 “Editing panel” wireframe

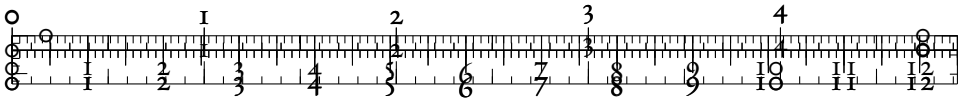
Figure 1.12 shows the wireframe of the editing panel used for editing fields and blocks.

Figure 1.12: "Editing panel" wireframe

1.5 Conclusion

In this chapter, we presented the functional and non-functional requirements of our application, the different system actors, and their use case diagrams. We also explored the User Interface and verified our needs using low-fidelity wireframes.

Within the next chapter, we are going to have a deeper look into the architecture powering our application.



Acronyms

CmRDT Commutative Replicated Data Type

CMS Content Management Software

CRDT Conflict-free Replicated Data Type

CvRDT Convergent Replicated Data Type

DOM Document Object Model

FDD Feature-Driven Development

GPU Graphics Processing Unit

JSON JavaScript Object Notation

MVC Model-View-Controller

MVVM Model-View-ViewModel

OT Operational Transformation

OTP One-Time Password

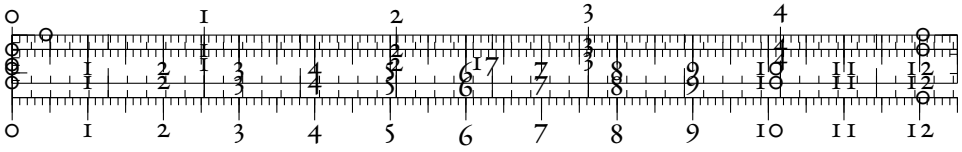
P2P Peer-To-Peer

RBAC Role-Based Access Control

RDMBS Relational Database Management System

SaaS Software as a Service

SDK Software Development Kit



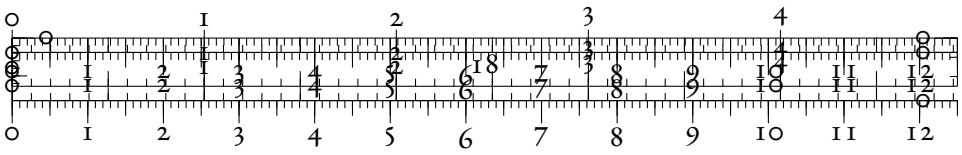
SSPL Server Side Public License

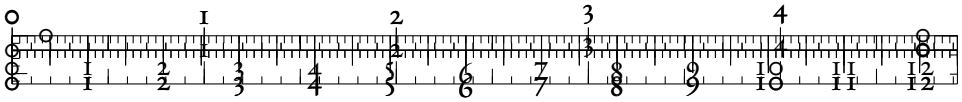
UI User Interface

UX User Experience

WAI-ARIA Web Accessibility Initiative – Accessible Rich Internet Applications

WOOT WithOut Operational Transformation

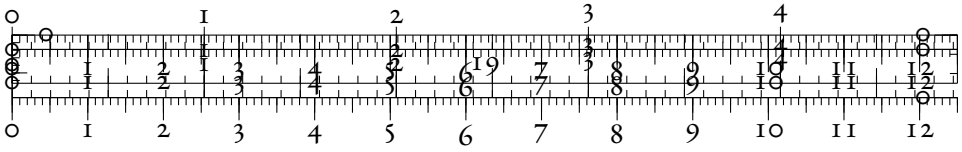


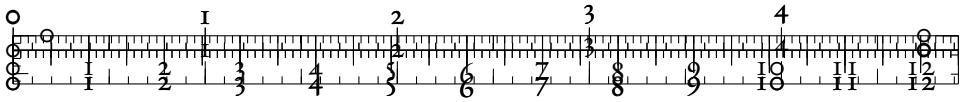


Glossary

cacheable A cacheable response is an HTTP response that is stored to be retrieved and used later, saving a new request to the server [1].

polyfill A piece of code used to provide modern functionality on older browsers that do not natively support it [4].

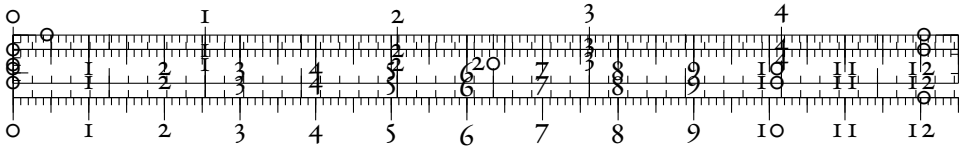




Bibliography

[1] *Cacheable - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. en-US. URL: https://developer.mozilla.org/en-US/docs/Glossary/cacheable (visited on 06/07/2021).

[2] J. J. Garrett. *The elements of user experience: user-centered design for the Web and beyond*. 2nd ed. Voices that matter. OCLC: ocn503049598. Berkeley, CA: New Riders, 2011. ISBN: 9780321683687.

[3] T. Gemayel. *How to wireframe*. en-US. URL: https://www.figma.com/blog/how-to-wireframe/ (visited on 06/04/2021).

[4] *Polyfill - MDN Web Docs Glossary: Definitions of Web-related terms | MDN*. en-US. URL: https://developer.mozilla.org/en-US/docs/Glossary/Polyfill (visited on 06/07/2021).