A Collaborative Visual Database

by

Imed Adel

A document submitted in partial fulfillment of the requirements for the degree of

Technical Report

at

Miskatonic University

Contents

List of Figures

List of Tables

General Introduction

Software is either slow, hard, or ugly—and sometimes all three. Nonetheless, since the introduction of the modern computer, software has taken the world by storm. It quickly became an essential component of every business since it paved the way for higher productivity and more automation, and therefore, increased profits.

Nearly forty years [8] have passed since the launch of the Apple Macintosh—one of the first commercially successful [9] mass-produced personal computers featuring a graphical user interface. Yet, software is still as inaccessible and inadequate for most users as ever. Perhaps the best example for such inaccessibility is the fact that this document is being produced using LaTeX—a fractured software system that requires a plethora of tools to be installed for the sake of producing a legible and aesthetically pleasing document.

Along with the domination of personal computers and software companies in the world, another technology was on the rise—the internet. Since the dotcom bubble in the early 2000s, the internet has reshaped our lives. Be it entertainment, communication, education, or work, the internet is the primary and most powerful medium. Therefore, it is no surprise that most software companies switched to Software as a Service (SaaS), that is hosted software served through the medium of the internet [12], which quickly evolved into collaborative software aimed at teams rather than individual users.

The continued sprawl of these technological advancements led to the rise of remote work—a movement that erases any geographical limits and allows businesses and institutions to expand well beyond their headquarters. This movement has been recently magnified to unprecedented levels due to the global pandemic. The main traits of work and education instantly changed and non-collaborative software fell behind to give room to collaborative SaaS.

Within these changing dynamics, managing data is still an unsolved problem. Setting, managing, and securing a database is still one of the hardest tasks of building a business. Connecting the database to the rest of the business' applications is not as easy as one might expect. Keeping all employees on-board and managing access to the database, while allowing everyone to seamlessly collaborate is not easily achievable. Requiring all the above while keeping the costs low is impossible. Software geared towards managing data and content is either hard to configure and hard to use or slow to load and slow to on-board.

Based on the belief that software must be accessible, collaborative, fast, and hopefully enjoyable to use, we set out to develop a modern alternative. Using the latest technological innovations, our project is pushing the limits for what is possible with collaborative SaaS for managing data and content. Merebase—our project—is a collaborative visual database SaaS that challenges the norms, democratizes access to data management software, and fills the need for a no-code and low-cost database software.

Our work is discussed in four chapters.

- Presentation is an introduction.
- Analysis and specification of needs is an analysis.

I Presentation

1.1 Introduction

The aim of this chapter is to contextualize our work. We will start by introducing the hosting institution for the graduation project. Then, we will present the project, its motivations, and its objectives. And finally, we will discuss the development process used throughout the making of this project.

1.2 Host

This project is done as part of the final graduation project with the goal of obtaining the License Degree in Computer Science within the Higher School of Sciences and Technology of Hammam Sousse.

1.3 Project presentation

Our project's main idea and design choices stem from the problems we faced while trying to accomplish certain tasks using other tools.

1.3.1 Problematics

The continuous shift to Software as a Service (SaaS), coupled with the rise of remote work, uncovered a gap in the field of data and content management software. The gap is further exacerbated due to the accelerating adoption of web applications, which are mostly client-side applications without any server requirements. Nowadays, businesses are looking for easy and collaborative ways to allow stakeholders to manage data and content, and to connect the data to their different applications. The solution must respond to the needs of businesses from different backgrounds, with varying budgets, and minimal technical knowledge. The solution must also be easily integrable with other tools that these businesses might rely on. Furthermore, the solution must support recent technological advancements in the web, such as real-time collaboration and real-time queries. To ensure these requirements, we need to answer the following questions:

- How to support real-time collaboration?
- What level of collaboration is required for optimal productivity?
- How should we organize and share data between multiple users?
- What interface structure ensures the most accessible software?
- What data types should we support?
- How important is speed?
- How can we ensure a fast user experience?
- What are the bottlenecks of the existing solutions and how can we solve them?
- How can we ensure a fast and easy on-boarding?

1.4 Preliminary study

Before starting the development process of our projects, it is of utmost importance to research the existing solutions in the field of data and content management that our potential users are currently relying on. It is necessary to understand what problems users are facing while using these solutions and what kind of tricks and shortcuts do they have to depend on to achieve their desired outcome. We should also focus on the points that users admire about their current choices, as these are the features keeping them from looking for another solution in the meantime.

With this goal in mind, we went on to research multiple applications and software systems with varying degrees of features and requirements. While some might require deep technical knowledge of databases, servers, and programming, others are more straightforward and require little to no technical knowledge. However, while some applications may require no programming skills, they still require some time for on-boarding and getting familiar with the software. This can be a significant roadblock for many enterprises that are already stuck with some other software system.

From this wide pool of data and content management software, we selected the most used and loved ones and put them to comparison. In particular, we chose to focus on Notion, Airtable, Contentful, Sanity, Webflow CMS, and Firebase.

1.4.1 Existing solutions

We will start by presenting the selected solutions.

Firebase

Figure 1.1: Firebase logo

Firebase is a platform developed by Google for creating mobile and web applications. It was initially released in 2012. It offers, among its products, a real-time database. In which, data is stored in JSON format and synced between all the connected clients. The database was not developed with non-technical users in mind, however, its real-time capabilities offer an example of what's desired in real-time database software. Firebase Realtime Database has been successfully used to develop highly demanding mobile applications.

Airtable

Figure 1.2: Airtable logo

Airtable is a visual database app inspired by the ease of spreadsheets and the wide adoption of software like Microsoft Excel. The company behind the app was founded in 2012.

Airtable comes with team collaboration out of the box. It also automatically generates a REST API from each database.

Pricing is done per team member. There are several limits to the size of storage and uploads.

Contentful

Figure 1.3: Contentful logo

Contentful is a headless[1] CMS (Content Management Software). It offers a flexible CMS editor and a configurable API. It also comes with multiple SDKs (Software Development Kits) in multiple programming languages to make its integration easier.

Pricing is offered per package, with the lowest premium package starting at US$489 per month.

Sanity

Figure 1.4: Sanity logo

[1] Content is decoupled from the main application. It's made accessible through a set of APIs.

Sanity is another headless CMS. It competes directly with Contentful, offers an even more configurable editor, and its pricing starts at US$199 per month. It comes with real-time collaboration, a feature that Contentful lacks.

Webflow CMS

Figure 1.5: Webflow logo

Webflow is a website builder. It bundles a CMS and an e-commerce management system along with its visual website builder. The CMS is not usable outside of Webflow websites, however, it comes with an intuitive user interface.

Notion

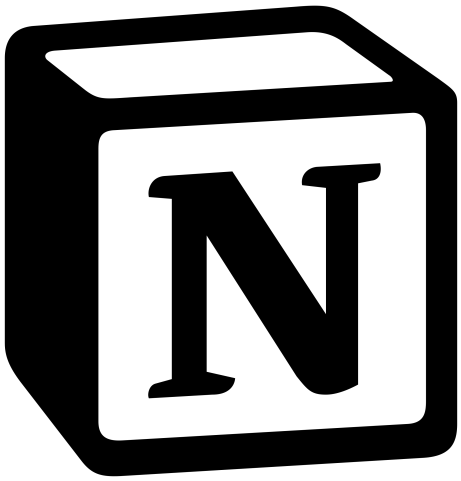

Figure 1.6: Notion logo

Notion is a new contender in the space of content management. It presents itself as a collaborative workspace for teams. Its use cases vary from product management and team documentation to note-taking and personal organization. The initial version of Notion was released in 2016. The second version, which received a lot of praise and media coverage, was released two years later in 2018. However, the largest surge in sign-ups happened during the pandemic, with 40% of sign-ups occurring from December 2020 to January.

Notion is built on the concept of blocks: A block is any single piece of content you add to your page, like a to-do item, an image, a code block, an embedded file, etc. [2] This makes it easy to build complex pages and move content around.

Notion is also built as a collaborative web app—eliminating the need for saving and figuring out how to share one's documents as is the case in other apps.

[2] citation needed, see Notion FAQ

Pricing is done per workspace member with unlimited storage starting from the free plan.

1.4.2 Comparison of the existing solutions

In order to have a better understanding of the different offerings of the selected solutions, and their features and shortcomings, we have to compare them side by side.

Methodology

For a fair and objective comparison of the different solutions, we will rely on Web.Dev and Google Chrome Lighthouse for measuring performance, accessibility, speed and security, and Capterra for aggregating user reviews and forming a consensus about the main strain points in the existing tools.

Web.Dev Web.Dev is a web application developed by Google that uses the Lighthouse tool to measure different websites and web applications metrics. It can only audit public web pages, however, it offers metrics totally unbiased by our browser setup.

Figure 1.7: Lighthouse logo

Google Chrome Lighthouse Google Chrome Lighthouse is an extension available by default in Google Chrome browsers. It can measure different website and web application metrics. It can audit both public and private web pages. However, the results can be affected by any installed browser extensions. Which is why we run this tool in an isolated browser installed for this particular use case.

Figure 1.8: Capterra logo

Capterra Capterra is a free resource that helps businesses of all kinds compare available software and find the right software for their needs. It offers software ratings, reviews and buying guides.

Comparison table

Table 1.1 is an objective side-by-side comparison of our selected solutions.

	Firebase	Airtable	Contentful	Sanity	Webflow CMS	Notion
Released	2016	2012	2016	2016	2012	2012
Type	Database	Spreadsheet	CMS	CMS	CMS	Wiki
Requires	Yes	No	No	No	No	No
API	Yes	Yes	Yes	Yes	Yes	Partial
Rating	4.6	4.7	4.5	—	—	4.7

Table 1.1: Comparative table of the existing solutions

1.4.3 Critique

Multiple solutions are trying to focus on various use cases, however, all of them suffer from noticeable performance issues, a bad UX (User Experience), and inadequate pricing for small and medium-sized businesses.

Notion is known for its slow performance and long loading times. Pages take on average between six and 12 seconds to load. [3] It also doesn't have an API, although one is being developed at the time of writing. Furthermore, Notion is less structured than products like Airtable or Firebase.

Airtable is notable for its complexity, even for experienced users. It also suffers from some performance issues when loading large documents. Furthermore, it doesn't have the same rich text capabilities as Notion. Finally, it lacks a real-time API and it's relatively expensive.

[3] citation needed

Case study: Notion

TBD.

1.4.4 Proposed solution

Merebase is a collaborative visual database that can be used for data and content management. It's built with real-time collaboration, performance, and intuitiveness in mind. Thanks to years of innovation in the field of browser apps and high-performance real-time servers, it should be able to load instantaneously, while offering a smooth user experience with no glitching or slowdowns when loading large documents, and with the ability to effortlessly collaborate with other users.

1.5 Development process

To ensure the optimal use of time and energy, we chose to follow a development process throughout this project. That is, dividing work into smaller chunks according to a certain set of rules. In particular, we followed the principles of Agile software development, which is an umbrella for multiple methodologies and frameworks. Of these methodologies, we adopted Feature-Driven Development (FDD) and the Kanban method for their practicality and ease-of-use, especially for projects with small teams.

1.5.1 Agile software development

Agile software development is an umbrella term for a set of frameworks and practices based on the set of principles popularized by the Manifesto for Agile Software Development in 2001. It advocates adaptive planning, evolution-

ary development, early delivery, and continual improvement, and it encourages flexible responses to change. It derives its values from a range of software development frameworks and methodologies.

Values

The Manifesto for Agile Software Development proclaims the following values:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Principles

The Manifesto for Agile Software Development is based on twelve principles:

1. Customer satisfaction by early and continuous delivery of valuable software
2. Welcome changing requirements, even in late development
3. Deliver working software frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the primary measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design

10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Best architectures, requirements, and designs emerge from self-organizing teams
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

1.5.2 Feature-Driven Development

Feature-Driven Development (FDD) is an Agile method for developing software iteratively and incrementally. It encourages planning, design, and development based on features.

1.5.3 Kanban

Kanban is an Agile method to manage work by balancing demands with available capacity, while uncovering bottlenecks. Work is divided into smaller tasks that are visualized on top of a Kanban board.

1.6 Conclusion

In summary, throughout this chapter, we have presented our project, the hosting institution, the motivations behind our choices, and our objectives. We also researched the existing solutions and we started drawing the picture for what our project strives to achieve.

Within the next chapter, we will be going into more details of this picture by focusing on the analysis and specification of needs for out project.

2 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.

2.1 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.

2.1.1 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

2.1.2 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.

2.2 Specification of needs

2.2.1 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.

2.3 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.

2.3.1 General use case diagram

Diagram 2.1 describes the general use case of our application. In the following diagrams, we will further analyze and dissect each use case.

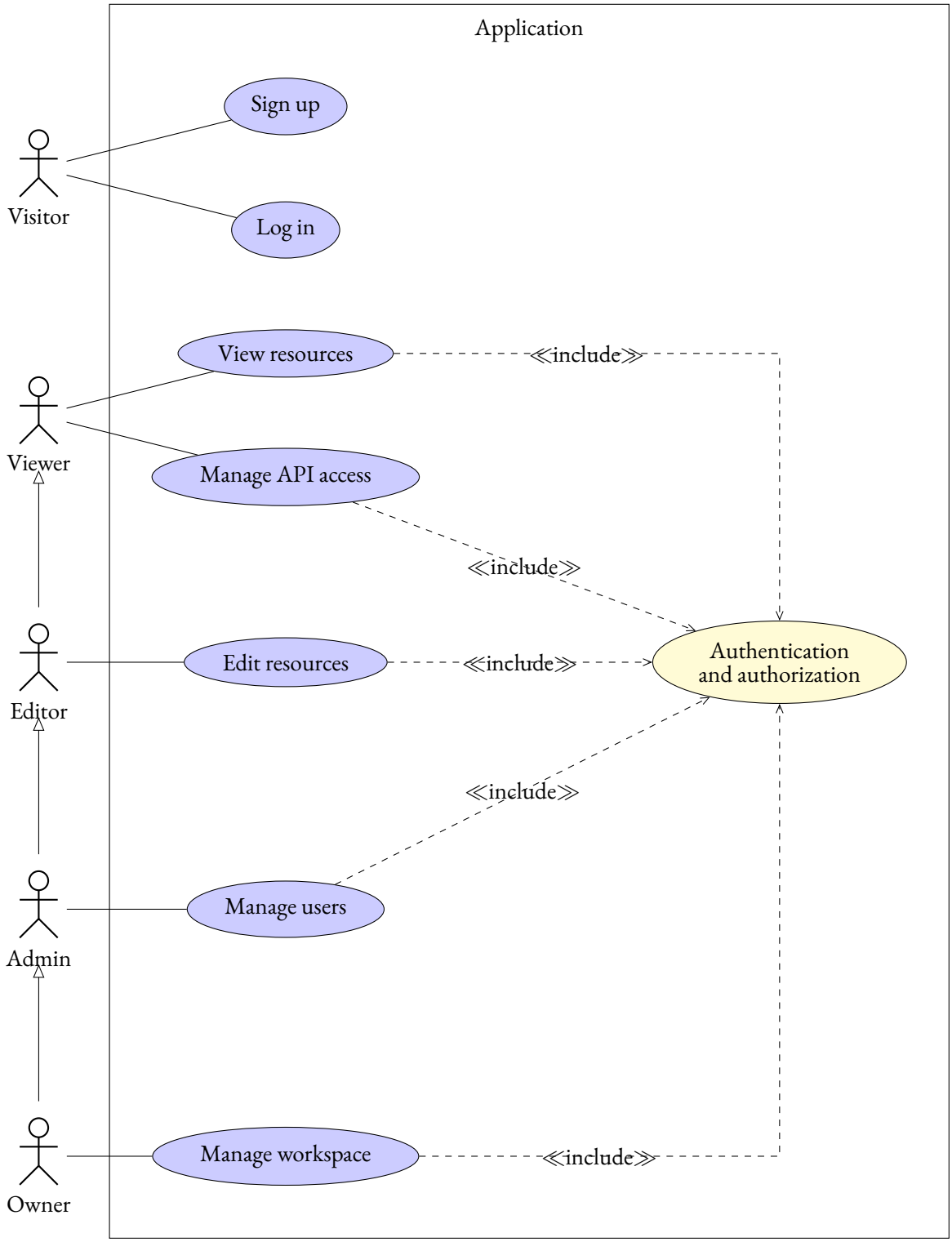

Figure 2.1: General use case diagram

2.3.2 "Sign up" use case diagram

Diagram 2.2 describes the "sign up" use case of our application.

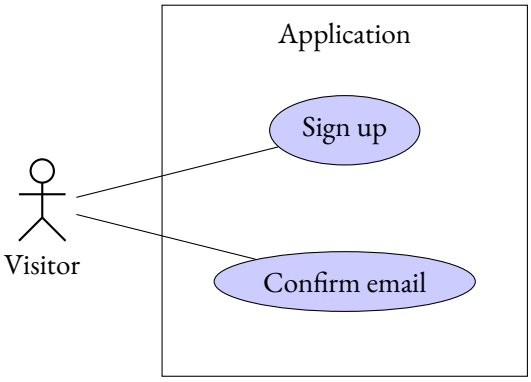

Figure 2.2: "Sign up" use case diagram

2.3.3 "Log in" use case diagram

Diagram 2.3 describes the "log in" use case of our application.

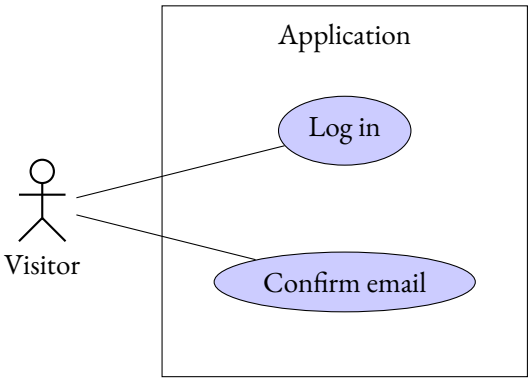

Figure 2.3: "Log in" use case diagram

2.3.4 "View resources" use case diagram

Diagram 2.4 describes the "view resources" use case of our application.

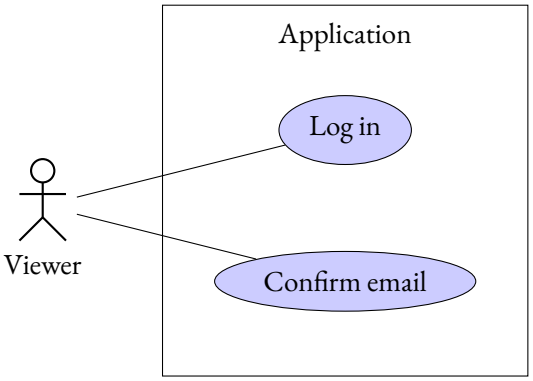

Figure 2.4: "View resources" use case diagram

2.3.5 "Edit resources" use case diagram

Diagram 2.5 describes the "edit resources" use case of our application.

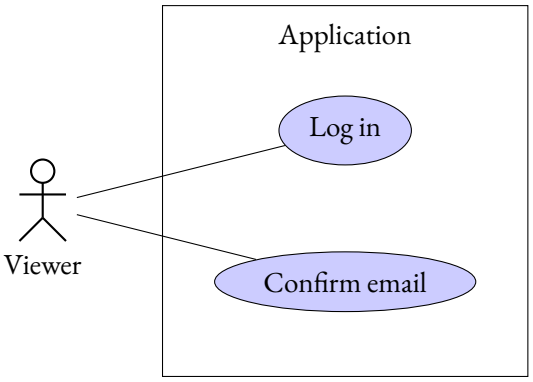

Figure 2.5: "Edit resources" use case diagram

2.3.6 "Manage users" use case diagram

Diagram 2.6 describes the "manage users" use case of our application.

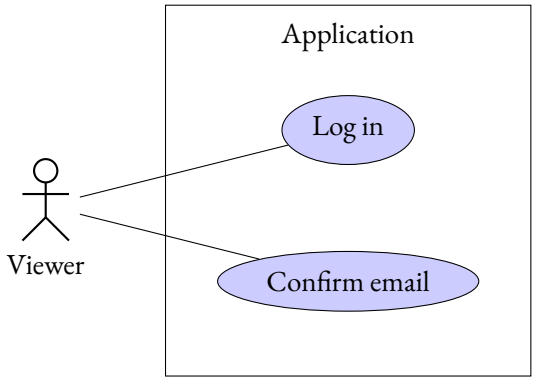

Figure 2.6: "Manage users" use case diagram

2.3.7 "Manage workspace" use case diagram

Diagram 2.7 describes the "manage workspace" use case of our application.

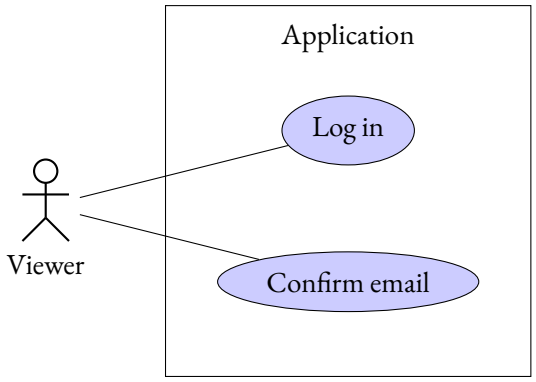

Figure 2.7: "Manage workspace" use case diagram

2.3.8 "Authentication and authorization" use case diagram

Diagram 2.8 describes the "authentication and authorization" use case of our application.

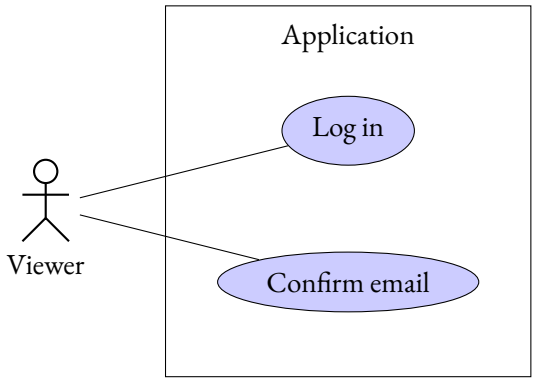

Figure 2.8: "Authentication and authorization" use case diagram

2.4 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 [6]. 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 [5].

2.4.1 “Log in” wireframe

Figure 2.9 shows the wireframe of the log-in page.

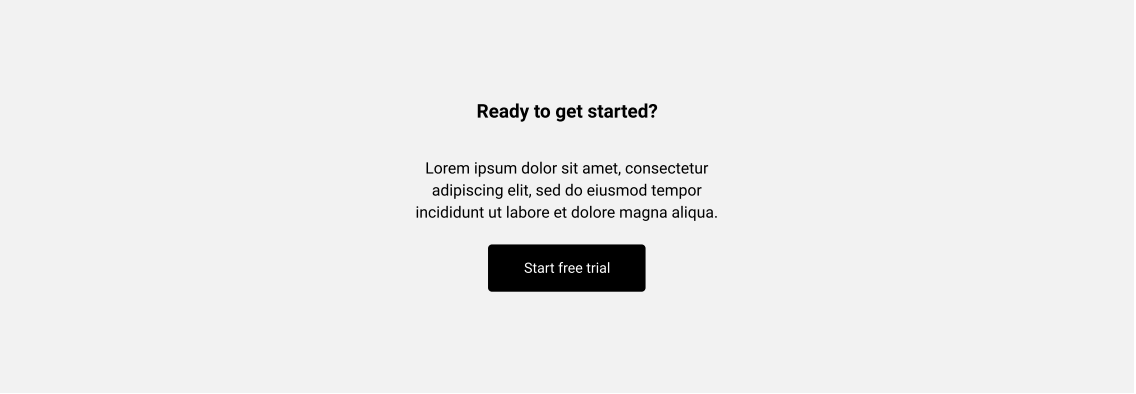

Figure 2.9: ”Log in” wireframe

2.4.2 “Sign up” wireframe

Figure 2.10 shows the wireframe of the log-in page.

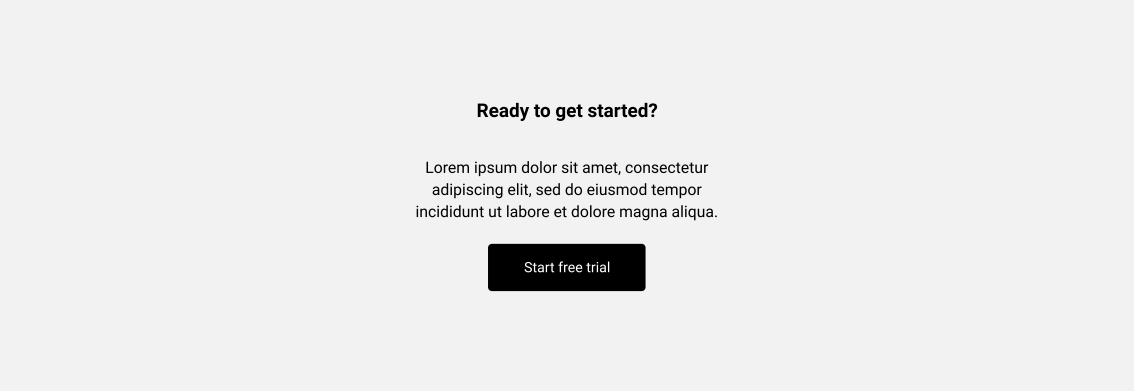

Figure 2.10: ”Log in” wireframe

2.4.3 “Confirmation” wireframe

Figure 2.11 shows the wireframe of the log-in page.

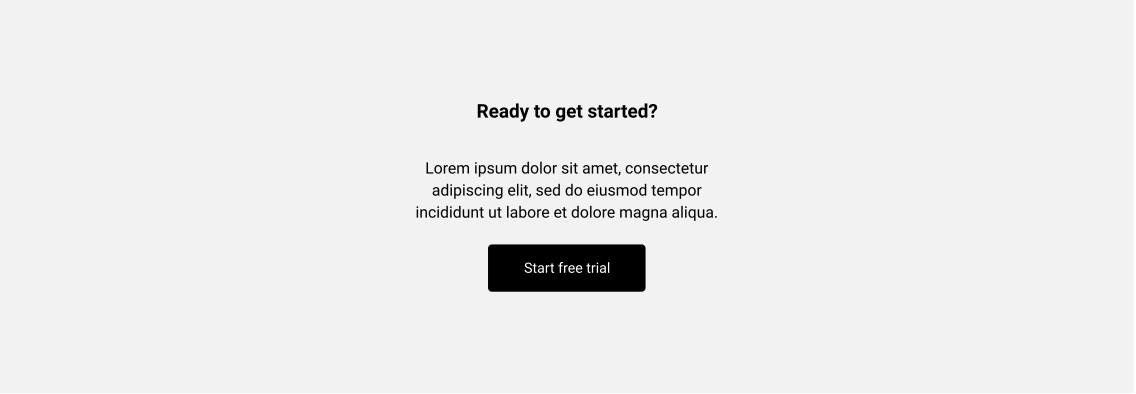

Figure 2.11: ”Log in” wireframe

2.4.4 “Workspace setup” wireframe

Figure 2.12 shows the wireframe of the log-in page.

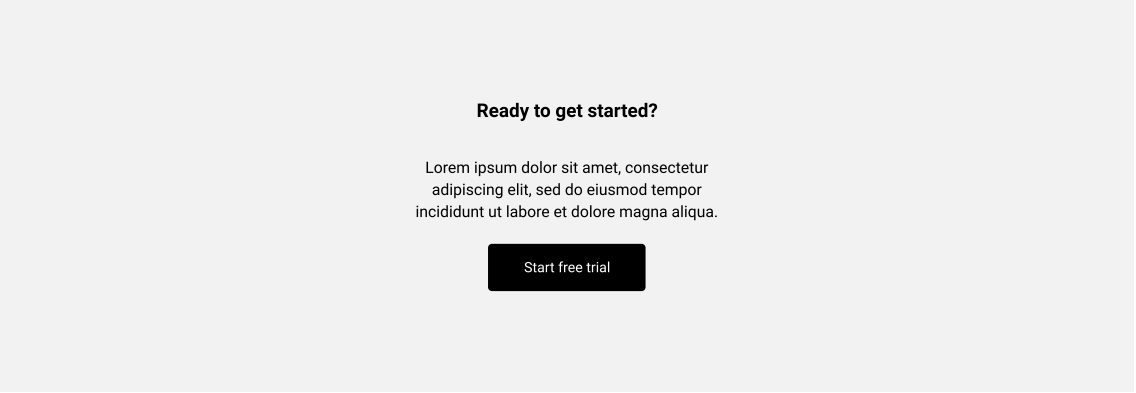

Figure 2.12: ”Log in” wireframe

2.4.5 “Homepage” wireframe

Figure 2.13 shows the wireframe of the log-in page.

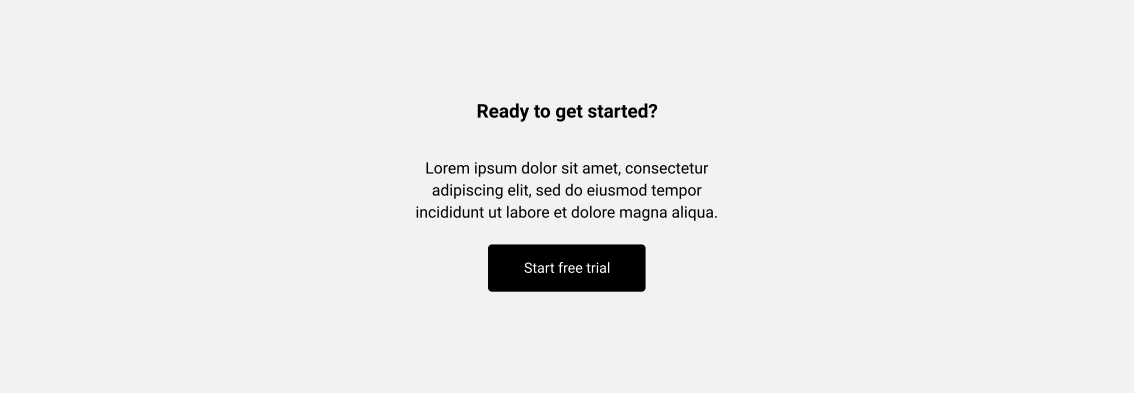

Figure 2.13: ”Log in” wireframe

2.4.6 “Projects” wireframe

Figure 2.14 shows the wireframe of the log-in page.

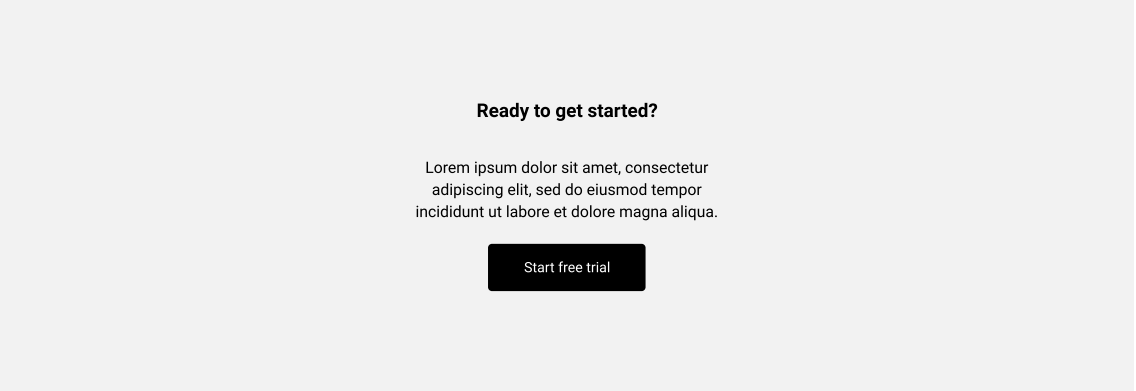

Figure 2.14: ”Log in” wireframe

2.4.7 “Collections” wireframe

Figure 2.15 shows the wireframe of the log-in page.

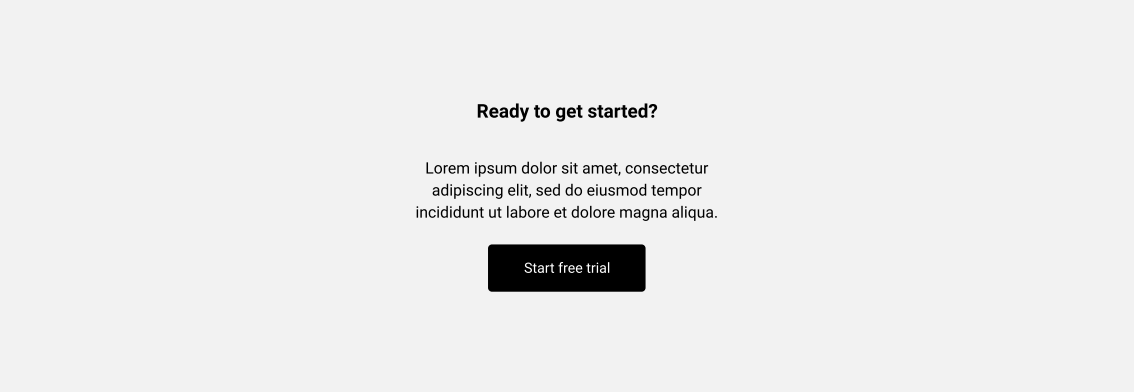

Figure 2.15: ”Log in” wireframe

2.4.8 “Documents” wireframe

Figure 2.16 shows the wireframe of the log-in page.

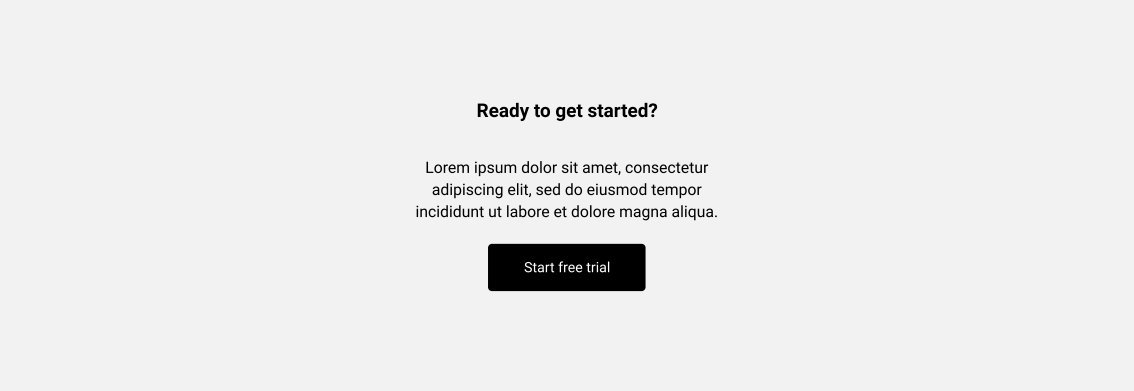

Figure 2.16: ”Log in” wireframe

2.4.9 “Fields” wireframe

Figure 2.17 shows the wireframe of the log-in page.

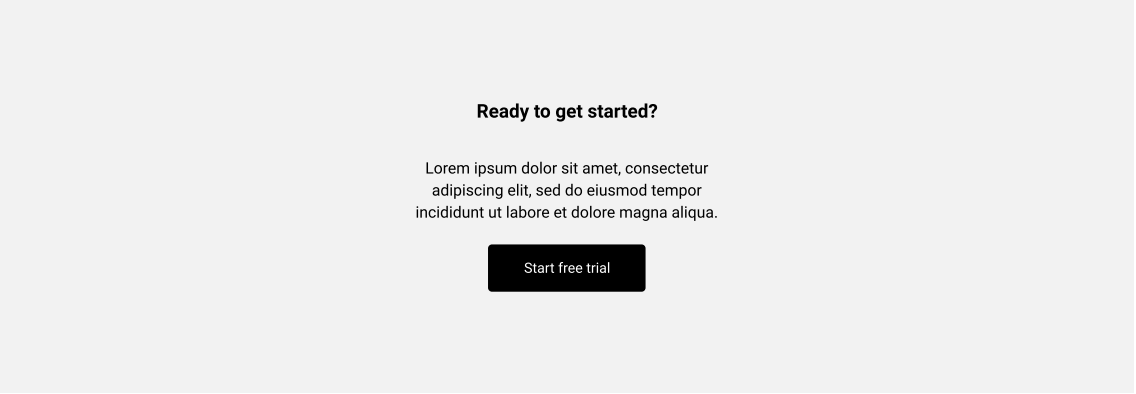

Figure 2.17: ”Log in” wireframe

2.4.10 “Blocks” wireframe

Figure 2.18 shows the wireframe of the log-in page.

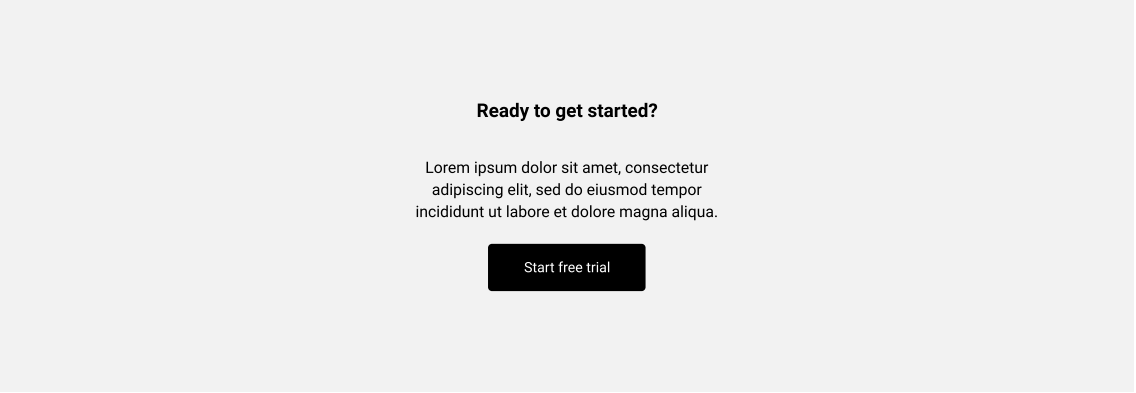

Figure 2.18: ”Log in” wireframe

2.4.11 "Access tokens" wireframe

Figure 2.19 shows the wireframe of the log-in page.

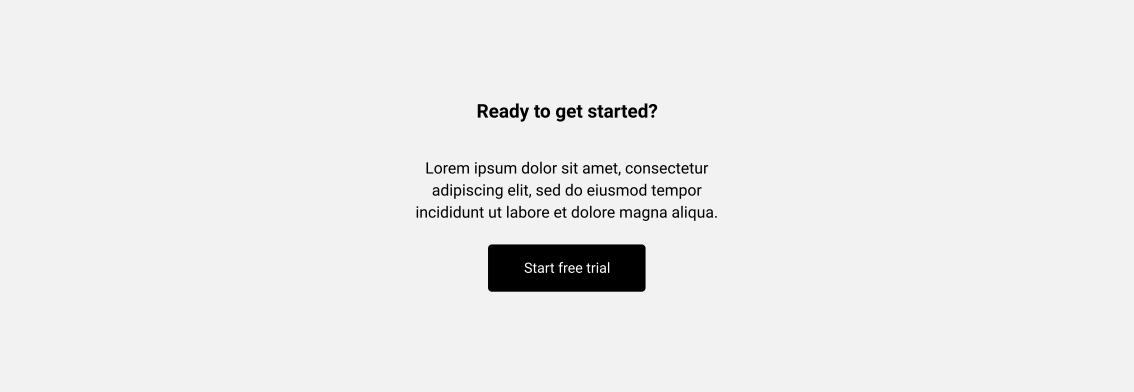

Figure 2.19: "Log in" wireframe

2.5 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.

3 Conceptual study

The deciding factor for the success of one application and the failure of another is the architecture. In order to achieve the requirements we set in the previous chapter, along with solving the issues faced by the existing solutions, we need to carefully design our systems.

In this chapter, we will first dive into the realm of real-time collaboration and read into the existing theoretical research in the field and its practical applications in the real world. Then, we will explore the different architectures for building and scaling our application. Finally, we will present the sequence and class diagrams to better define the implementation of our ideas and the trajectory of our work in the following chapter.

3.1 Real-time collaboration

Real-time collaboration is a type of collaboration used in editors and web applications with the goal of enabling multiple users on different computers or mobile devices to modify the same document with automatic and nearly instantaneous merging of their edits. The document could either be a computer file, stored locally, or a cloud-stored data shared over the internet, such as an online spreadsheet, a word processing document, a database, or a presentation.

Multiple web applications support real-time collaboration under various names. Microsoft, for example, refers to it as “co-authoring” and offers it as part of its

Microsoft Office bundle, including Word, Excel, and PowerPoint [2]. Google Docs is another notorious contender in the space of collaborative editing, with products such as Google Docs and Google Sheets.

The interest in collaborative software has seen a resurgence since 2020, mainly due to the move to remote work, with companies like Microsoft offering ready-to-use APIs to enable this feature.

Real-time collaboration is different from other offline or delayed collaborative approaches, such as Git. While real-time editing performs automatic, frequent, or even instantaneous synchronization of data between all the connected users, offline editing requires manual submission, merging, and resolution of editing conflicts.

3.1.1 History

In 1968, Douglas Engelbart introduced the first collaborative real-time editor in a presentation named "The Mother of All Demos", which also demonstrated many other fundamental elements of modern personal computing including windows, hypertext, graphics, video conferencing, the computer mouse, word processing, and revision control [3].

It took many decades for collaborative software to become mainstream. One of the first editors that offered collaborative capabilities was Writely, a collaborative word processor launched in 2005 [1]. It was later acquired by Google and renamed to Google Docs [7]. Another Google-backed product from the same era was Google Wave, a collaborative email software. However, less than a year later, it was discontinued due to a lack of users [4].

Since 2010, the number of web-based collaborative software has been exploding with successful examples such as Figma and Notion. Figure 3.1 shows the resurgence of products with real-time collaboration.

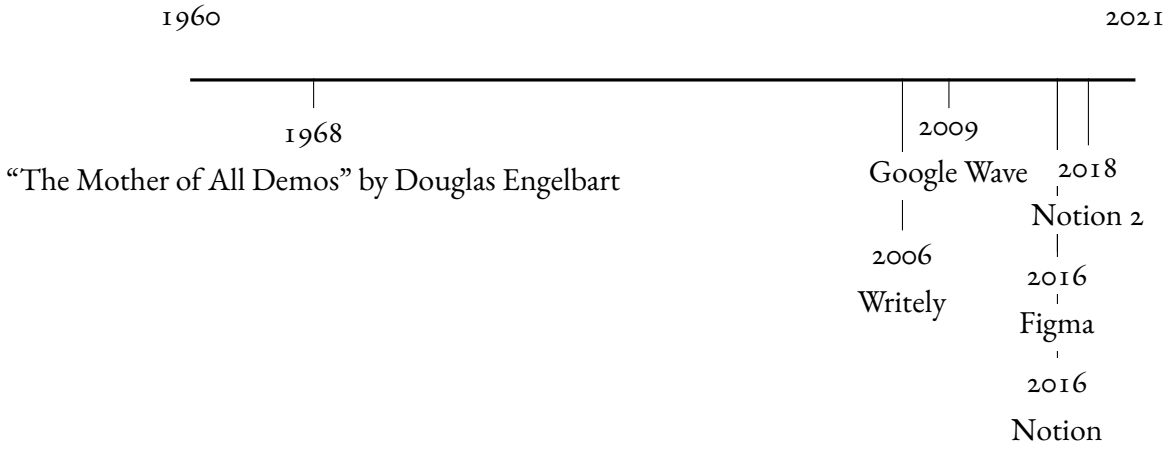

Figure 3.1: Timeline of real-time collaboration

3.1.2 Theoretical grounds

The main challenge of real-time collaboration is keeping multiple clients in sync. An algorithm has to be employed to determine how to apply—often conflicting—edits from the different remote users. Network latency is the main culprit for such a dilemma as modifications can reach the other clients with a certain amount of delay. Figure 3.2 shows a common example of conflicting changes by two users caused mainly by network latency.

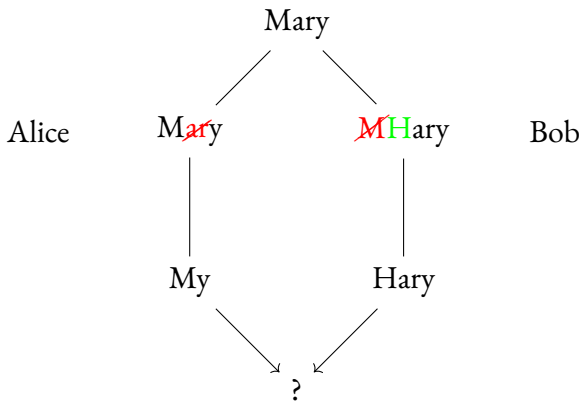

Figure 3.2: Example of a common problem in real-time collaboration

Over the years, two main algorithms emerged for dealing with real-time collaboration, Conflict-free Replicated Data Type (CRDT) and Operational Transformation (OT). Both have their benefits and drawbacks, and each one has numerous variations and implementations.

Operational Transformation (OT)

Operational Transformation was invented for supporting real-time co-editors in the late 1980s and has evolved to become a collection of core techniques widely used in today's working co-editors and adopted in major industrial products [11]. Google Docs has been using OT since at least 2009 [13].

The model of Operational Transformation works by defining all possible transformations for a text and applying those transformations to the received text after applying any needed adjustments to it. Figures 3.3 and 3.4 explain the algorithm followed by OT for resolving conflicts.

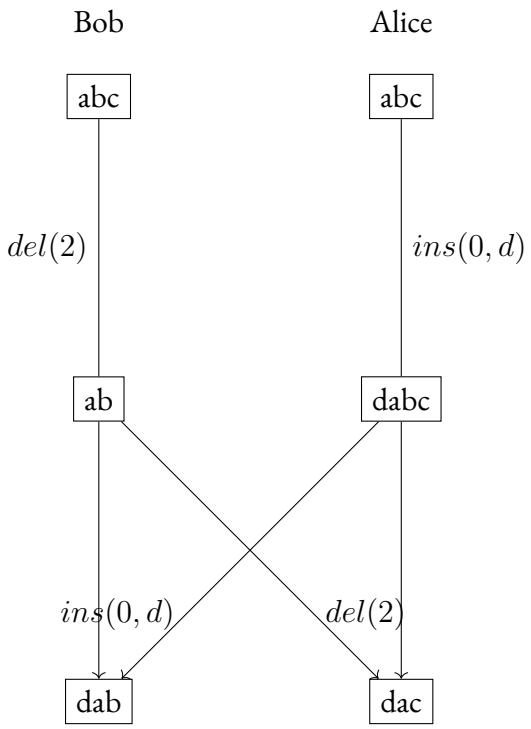

Figure 3.3: Without OT

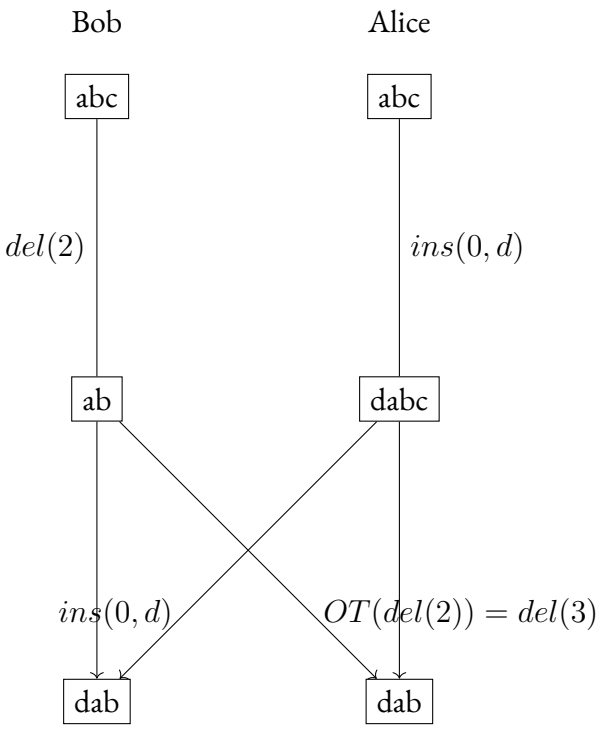

Figure 3.4: With OT

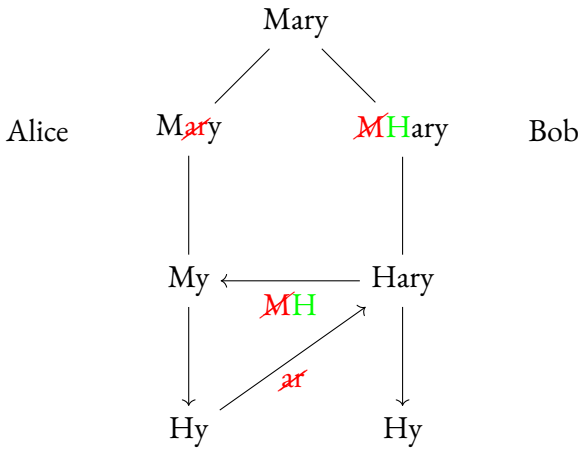

Figure 3.5: OT's solution for the problem in figure 3.2

As seen in the previous examples, OT can be hard and costly to set up, even for text-based documents. When it comes to more complicated and nested data structures, OT is rarely the first choice.

CRDT

Conflict-free Replicated Data Type is a data structure that was first proposed around 2006 under the name of WithOut Operational Transformation (WOOT) [11]. It has the benefit of being able to be duplicated over numerous computers in a network, where the replicas can be updated independently and concurrently without requiring coordination, and where inconsistencies can always be resolved mathematically [10].

In 2011, CRDT was formally defined [10], and while it was initially developed for collaborative text editing, it was eventually adopted for multiple other use cases such as online chat systems and distributed databases.

CRDT is further subdivided into two types based on its implementation—Commutative Replicated Data Type (CmRDT) and Convergent Replicated Data Type (CvRDT). Both of them offer the same real-time capabilities but differ in their design and approach to the concept.

CmRDT Commutative Replicated Data Type or Operation-based CRDT transmits the local state only by sending the update operations required to reach that state. Upon receiving these operations, remote clients must apply them to become in sync with the sender. The transmission server has to avoid duplicating the operations as this algorithm is not idempotent.

CvRDT Convergent Replicated Data Type or State-based CRDT sends over the whole local state to be merged with the receiving clients' states. This method tends to be slow due to the need of sending large amounts of data instanta-

neously over the network. However, the infrastructure does not have to deal with deduplication as it is the case with CmRDTs.

3.1.3 Practical examples

In practice, web applications do not adhere to one algorithm. Instead, they draw from different concepts to a solution that perfectly fits their needs.

Figma

Figma is one of the first web design tools with real-time collaboration. It was released in 2016 and since then it has been hailed as one of the best examples of performant and collaborative software.

Excalidraw

Excalidraw is an open-source white-board and wireframing web application.

Automerge

Automerge is a JavaScript data structure designed specifically for collaborative software.

Centige

Centige is a discontinued collaborative web app builder by the author. It was mainly a learning project, started in late 2019 and developed throughout 2020.

3.1.4 Conclusion

3.2 Design patterns

3.2.1 Common patterns

3.2.2 Comparison

3.2.3 Conclusion

3.3 Detailed architecture

3.4 Class diagrams

3.5 Sequence diagrams

3.6 Activity diagrams

3.7 Conclusion

4 Implementation

Researching the current solutions led us to formulate a set of requirements to ensure that Merebase offers the best experience.

Acronyms

CmRDT Commutative Replicated Data Type

CMS Content Management Software

CRDT Conflict-free Replicated Data Type

CvRDT Convergent Replicated Data Type

FDD Feature-Driven Development

OT Operational Transformation

RBAC Role-Based Access Control

SaaS Software as a Service

UI User Interface

UX User Experience

WOOT WithOut Operational Transformation

Bibliography

[1] E. Chang. *eHub Interviews Writely*. English. Oct. 2005. URL: https://web.archive.org/web/20110722190058/http://emilychang.com/ehub/app/ehub-interviews-writely/ (visited on 06/04/2021).

[2] *Document collaboration and co-authoring*. en-US. URL: https://support.microsoft.com/en-us/office/document-collaboration-and-co-authoring-ee1509b4-1f6e-401e-b04a-782d26f564a4 (visited on 06/04/2021).

[3] *Firsts: The Demo - Doug Engelbart Institute*. URL: https://www.dougengelbart.org/content/view/209/448/ (visited on 06/04/2021).

[4] I. Fried. *Google pulls plug on Google Wave*. en. URL: https://www.cnet.com/news/google-pulls-plug-on-google-wave/ (visited on 06/04/2021).

[5] 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.

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

[7] *Google acquires online word processor, Writely*. en-US. Mar. 2006. URL: https://venturebeat.com/2006/03/09/google-acquires-online-word-processor-writely/ (visited on 06/04/2021).

[8] *Macintosh Product Introduction Plan*. July 2010. URL: https://web.archive.org/web/20100721013724/http://library.stanford.edu/mac/primary/docs/pip83.html (visited on 04/01/2021).

[9] K. Polsson. *Chronology of Apple Computer Personal Computers (1984-1985)*. Aug. 2009. URL: https://web.archive.org/web/20090821105822/http://www.islandnet.com/~kpolsson/applehis/appl1984.htm (visited on 04/01/2021).

[10] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski. "Conflict-Free Replicated Data Types." en. In: *Stabilization, Safety, and Security of Distributed Systems*. Ed. by X. Défago, F. Petit, and V. Villain. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2011, pp. 386–400. ISBN: 9783642245503. DOI: 10.1007/978-3-642-24550-3_29.

[11] C. Sun, D. Sun, A. Ng, W. Cai, and B. Cho. "Real Differences between OT and CRDT under a General Transformation Framework for Consistency Maintenance in Co-Editors." en. In: *Proceedings of the ACM on Human-Computer Interaction* 4.GROUP (Jan. 2020), pp. 1–26. ISSN: 2573-0142. DOI: 10.1145/3375186. URL: https://dl.acm.org/doi/10.1145/3375186 (visited on 06/04/2021).

[12] B. Turner. *What is SaaS? Everything you need to know about Software as a Service*. en. URL: https://www.techradar.com/news/what-is-saas (visited on 04/01/2021).

[13] *What's different about the new Google Docs: Conflict resolution*. en. URL: https://drive.googleblog.com/2010/09/whats-different-about-new-google-docs_22.html (visited on 06/04/2021).