A Collaborative Visual Database

by

Imed Adel

ESSTHS

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 [29] have passed since the launch of the Apple Macintosh—one of the first commercially successful [44] 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 dot-com 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 [59], 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 to our work and it is intended to add the right context for our project.
- Analysis and specification of needs starts with exploring the existing solutions and proposing a better alternative.
- Conceptual study is our third chapter. It starts with a deep dive into real-time collaboration and its algorithms, followed by an exploration of the

common architectural patterns of modern applications, and eventually, presents our choices. Finally, within this chapter we introduced our detailed architecture of the application, with class and sequence diagrams to the rescue.

- Implementation is where we went deeper into our technology stack. We objectively compared our options and picked the right tools for the job. Finally, we brought our application into life.

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. 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, it must support recent technological advancements on 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] Content Management Software (CMS). It offers a flexible CMS editor and a configurable API. It also comes with multiple SDKs 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

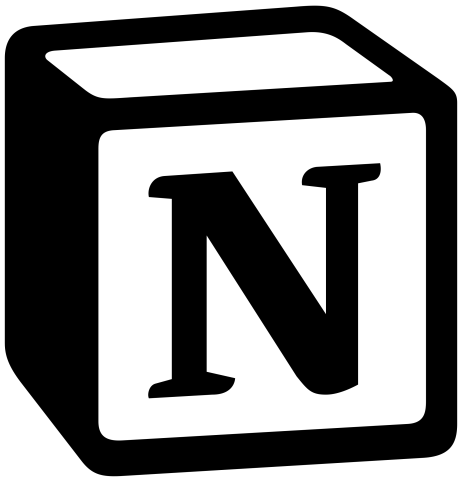

Figure 1.6: Notion logo

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. 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.

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. The code row specifies whether the software requires any technical knowledge to operate. The rating row is based on Capterra.

	Firebase	Airtable	Contentful	Sanity	Webflow CMS	Notion
Released	2016	2012	2016	2016	2012	2012
Type	Database	Spreadsheet	CMS	CMS	CMS	Wiki
Code	Yes	No	No	No	No	No
API	Yes	Yes	Yes	Yes	Yes	Beta
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, 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 [4]. It also doesn't have an API, although one is being developed at the time of writing [2]. 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 [3]. Furthermore, it doesn't have the same rich text capabilities as Notion. Finally, it lacks a real-time API, and it is relatively expensive [3].

[2] Notion's API was recently launched in beta.

Case study: Notion

Due to the popularity of Notion among various types of users, and due to the surge of sign-ups that it had seen, it seems to be the perfect product for a case study. While it is neither a database nor a CMS, it is often used as a limited content management solution for blogs.

Notion is a front-end React application, which means that the whole application has to be downloaded, parsed, and executed by the browser before it can load. Unfortunately, this happens nearly every time you visit their website [4]. On a fast 3G connection from Tunisia, the Notion page, depicted in figure 1.9, took 40 seconds to load. For the sake of comparison, the largest page ever on Wikipedia [28] took only 9 seconds to load.

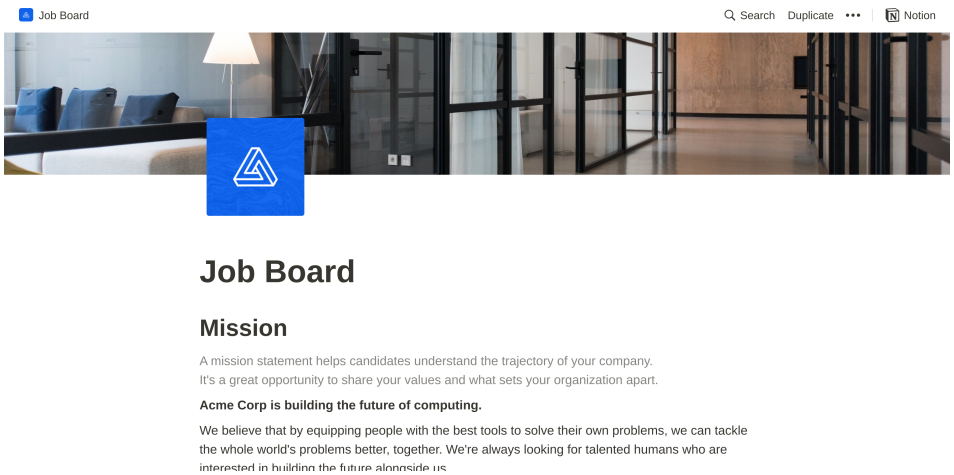

Figure 1.9: Notion page

While fetched resources can be cached for faster subsequent loads, processing JavaScript code and images is not cacheable and will always result in a slow application.

Further analysis shows that not much optimization has been done on Notion's part. The whole JavaScript code is loaded, regardless of whether it is needed on the requested page or not, unnecessary polyfills are loaded even for modern browsers, huge libraries are used—even when a smaller alternative exists—and images are served in their original format without optimization or minification.

The User Experience (UX), aside from the loading issues, is rather simple. Any person can start using Notion without any prior knowledge required.

The takeaway from studying Notion is that performance should not be an afterthought. Instead, it should be part of every decision in the design process of our application. As for the interface, we should strive to keep it as simple and easy as Notion.

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, evolutionary 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.

2.3.2 "Sign up" use case diagram

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

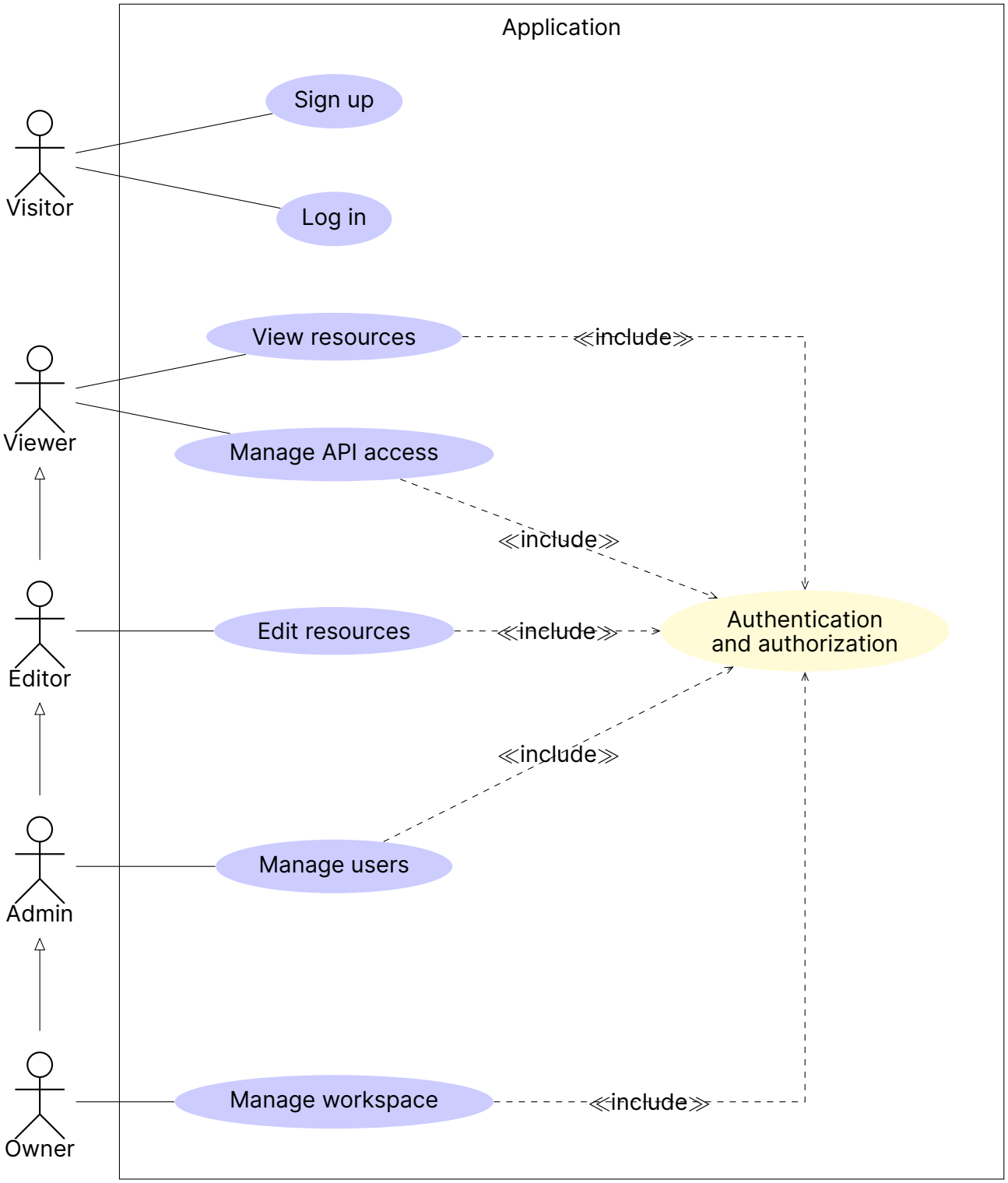

Figure 2.1: General use case diagram

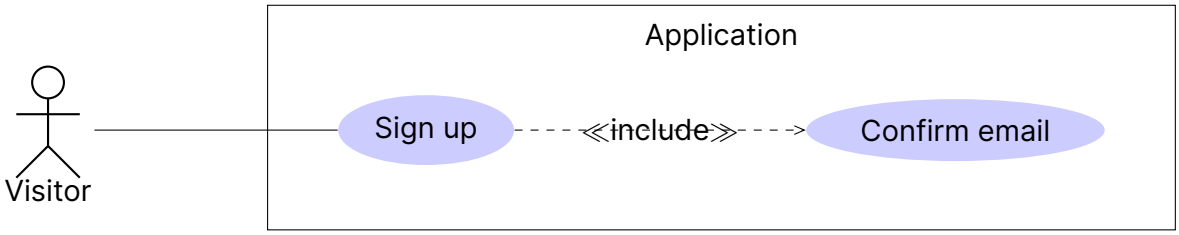

Figure 2.2: "Sign up" use case diagram

2.3.3 "Log in" use case diagram

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

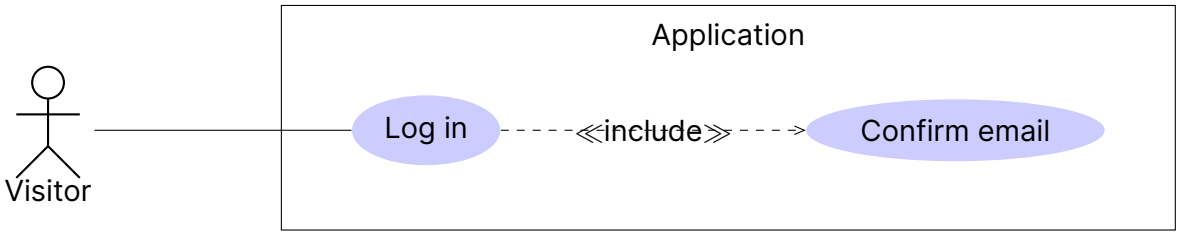

Figure 2.3: "Log in" use case diagram

2.3.4 "View resources" use case diagram

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

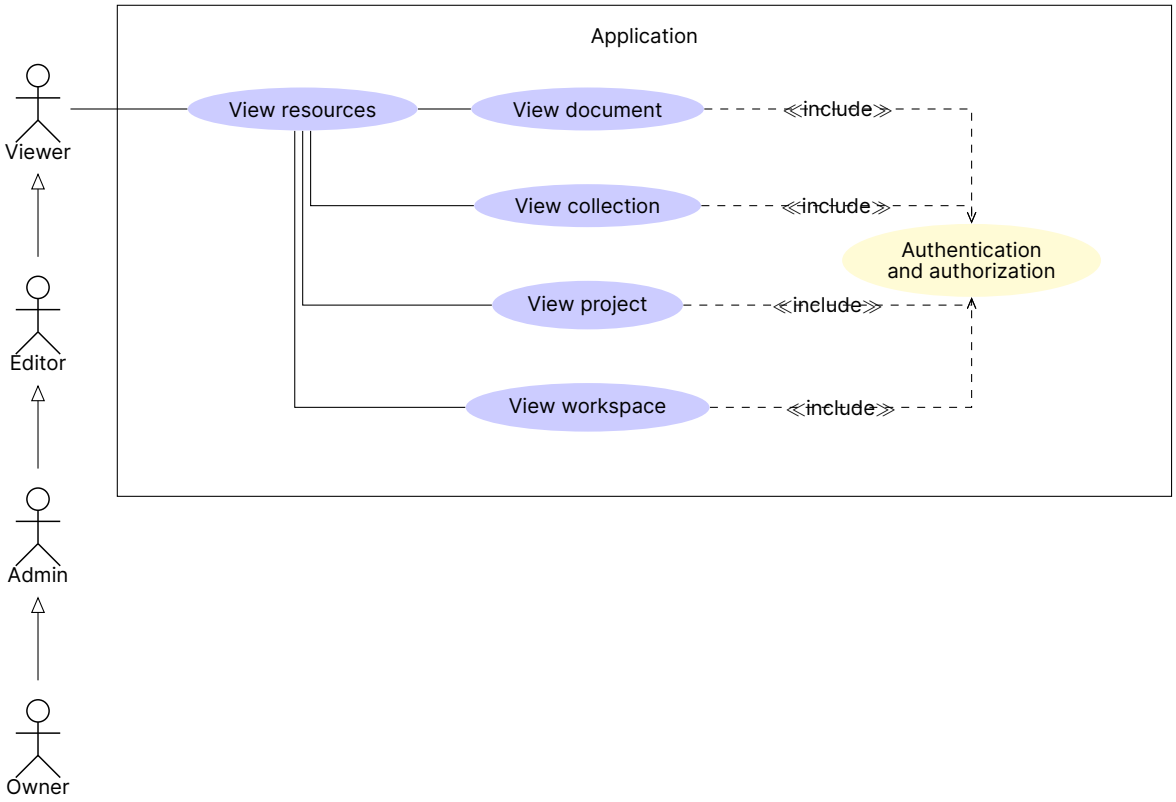

Figure 2.4: “View resources” use case diagram

2.3.5 “Edit resources” use case diagram

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

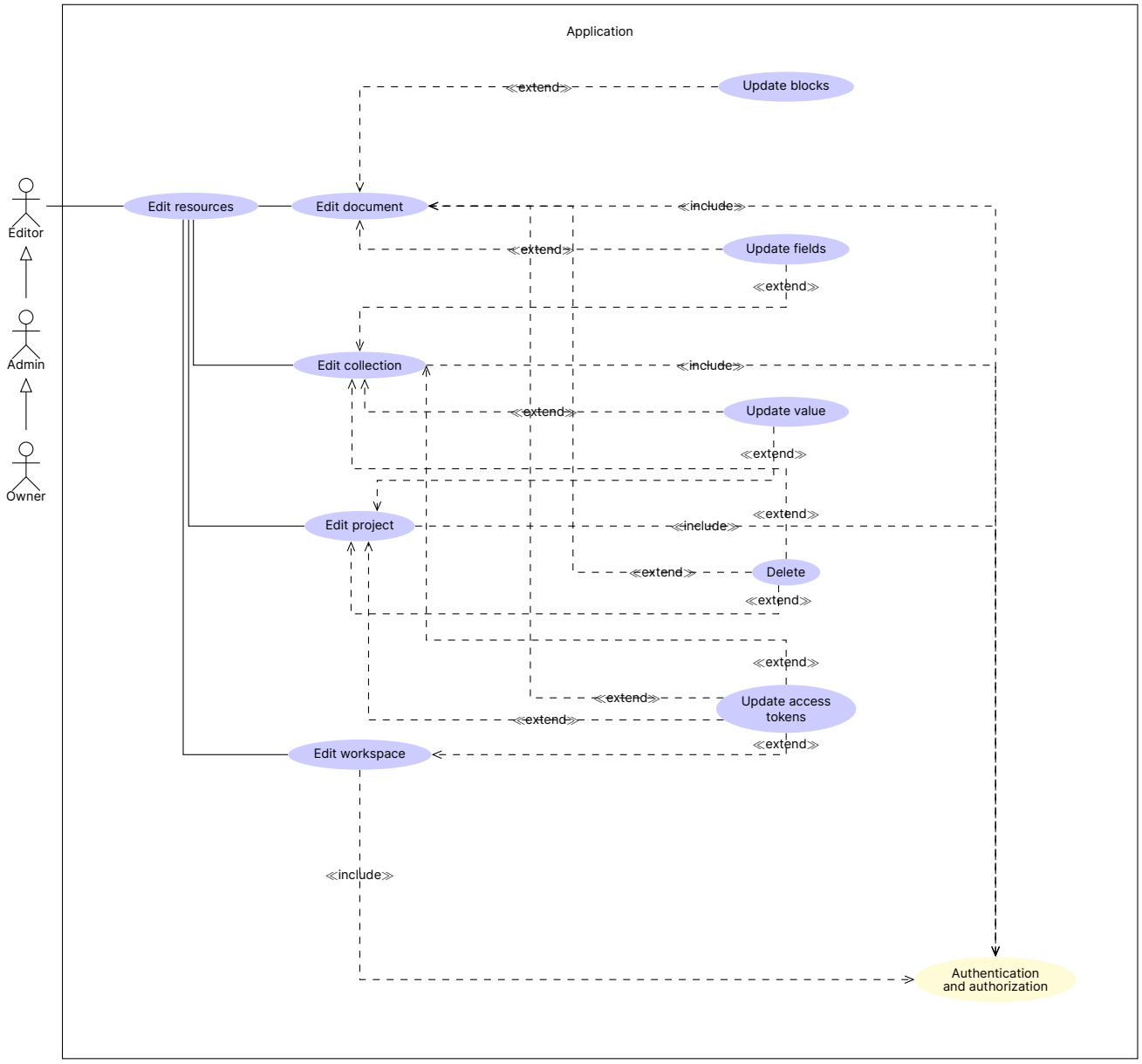

Figure 2.5: "Edit resources" use case diagram

2.3.6 "Manage users" use case diagram

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

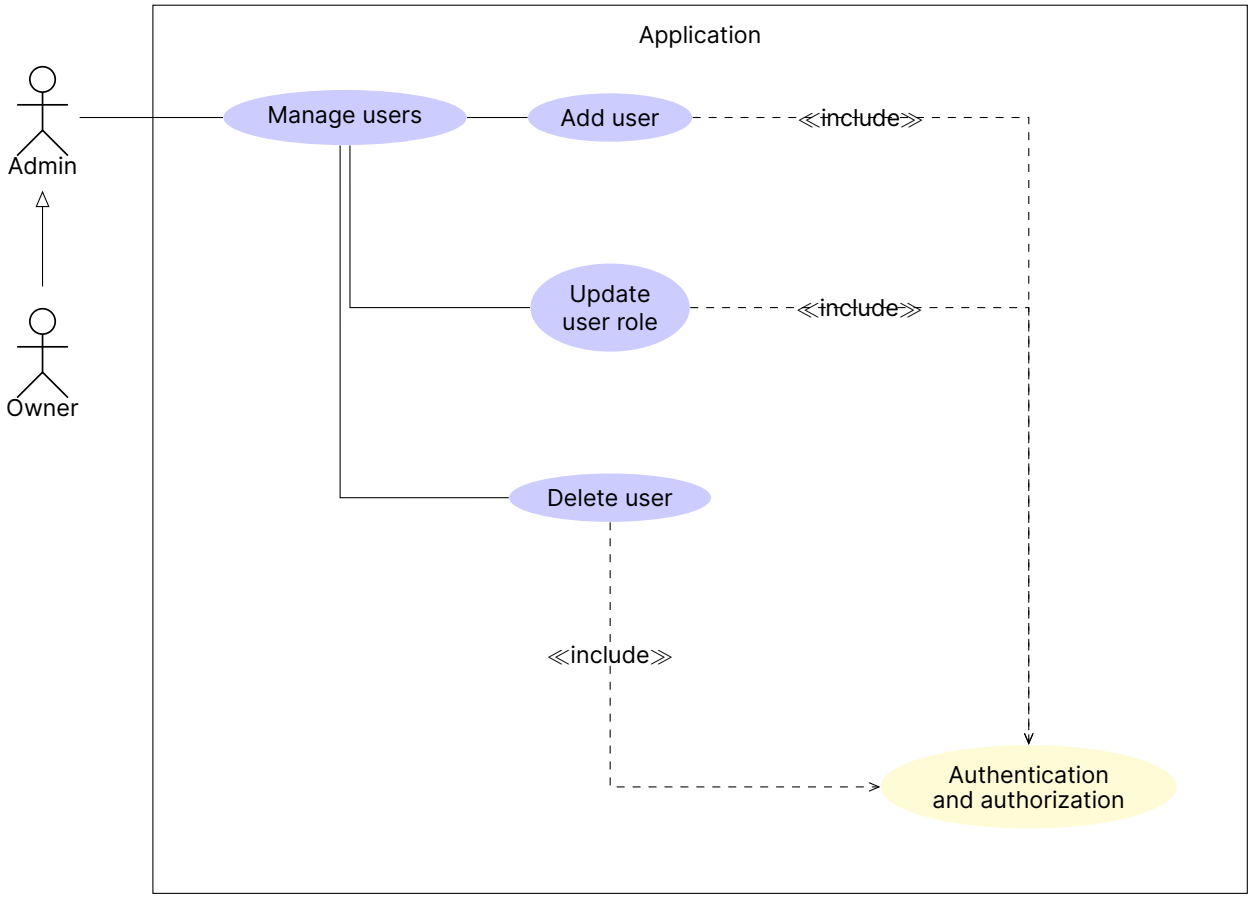

Figure 2.6: "Manage users" use case diagram

2.3.7 "Manage workspace" use case diagram

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

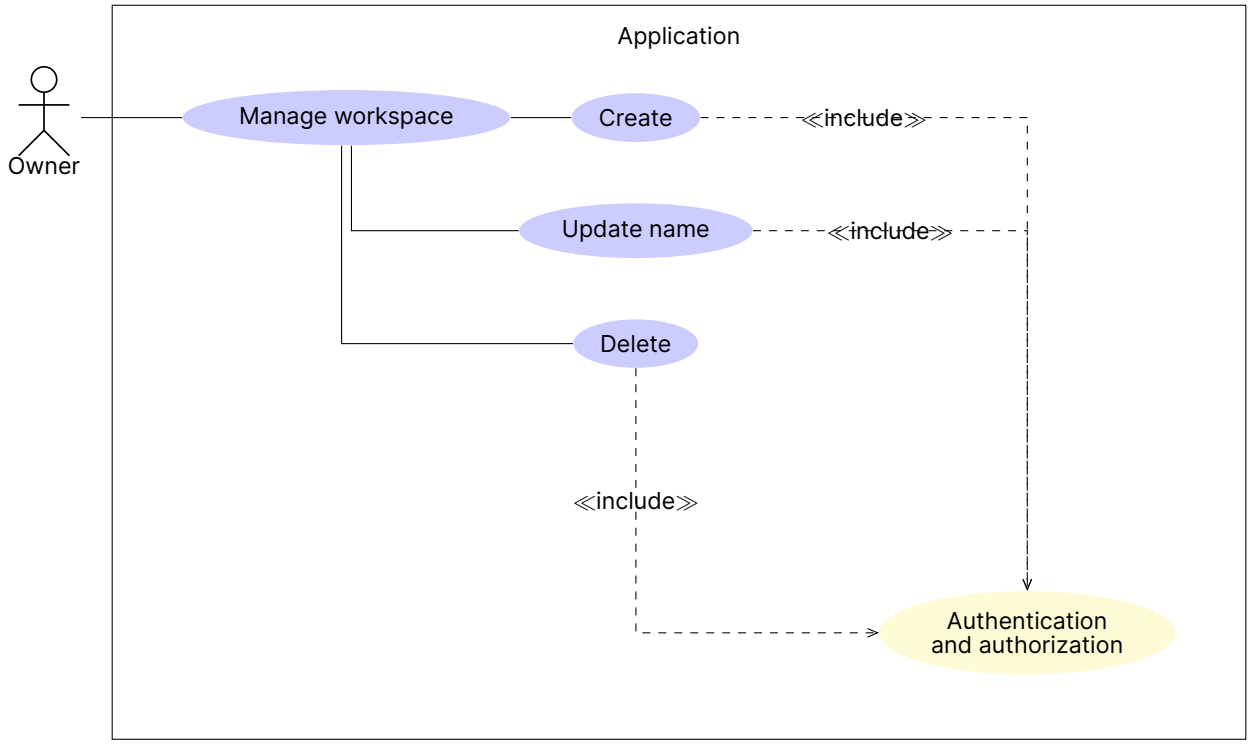

Figure 2.7: "Manage workspace" use case diagram

2.3.8 "Authentication and authorization" use case diagram

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

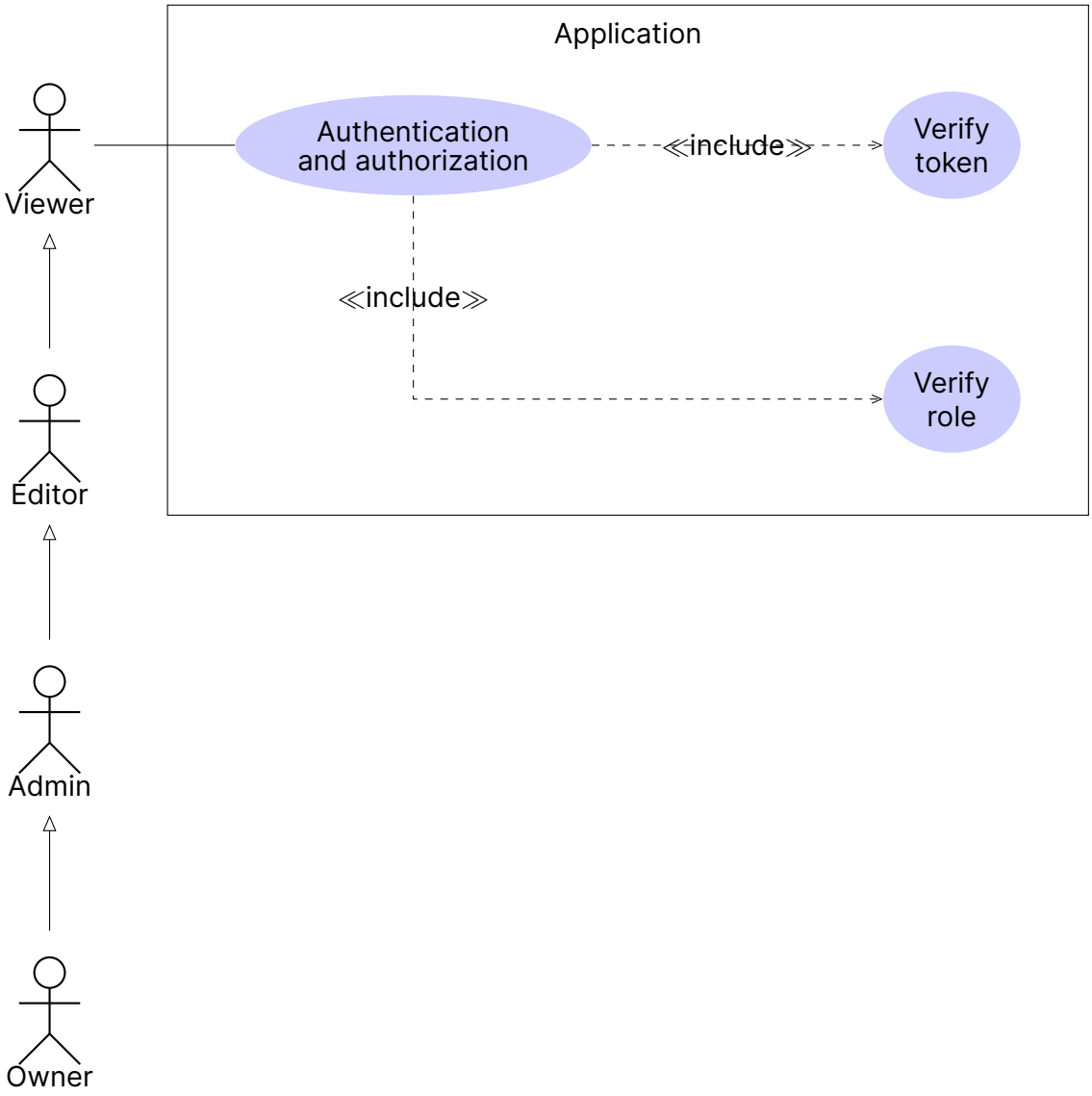

Figure 2.8: "Authentication and authorization" use case diagram

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 [20]. 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 [19].

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, taking into consideration our choice of collaboration algorithms. 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 [13]. 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 [16].

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 [8]. It was later acquired by Google and renamed to Google Docs [24]. 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 [18].

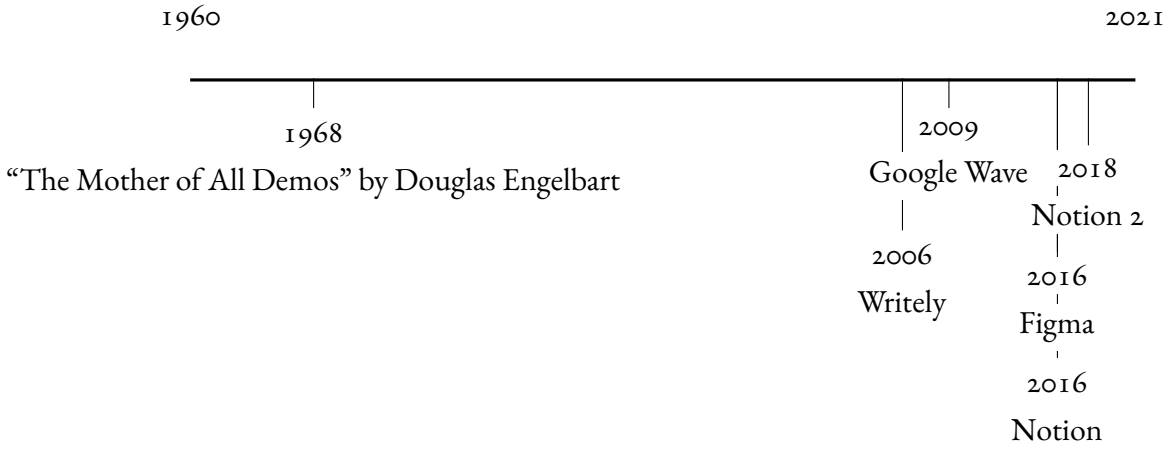

Figure 3.1: Timeline of real-time collaboration

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.

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.

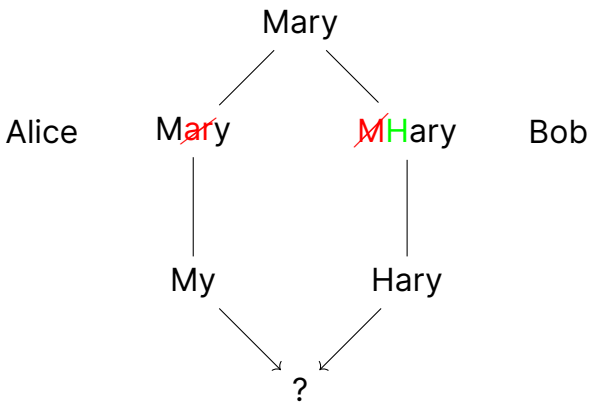

Figure 3.2: Example of a common problem in real-time collaboration

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 [55]. Google Docs has been using OT since at least 2009 [65].

The model of Operational Transformation works by calculating all possible transformations for a text and using them to transform received changes according to the local state, thus eliminating any possible mistakes of synchronization. Figures 3.3 and 3.4 explain the algorithm followed by OT for resolving conflicts. Figure 3.5 illustrates the solution proposed by OT for the problem proposed in 3.2.

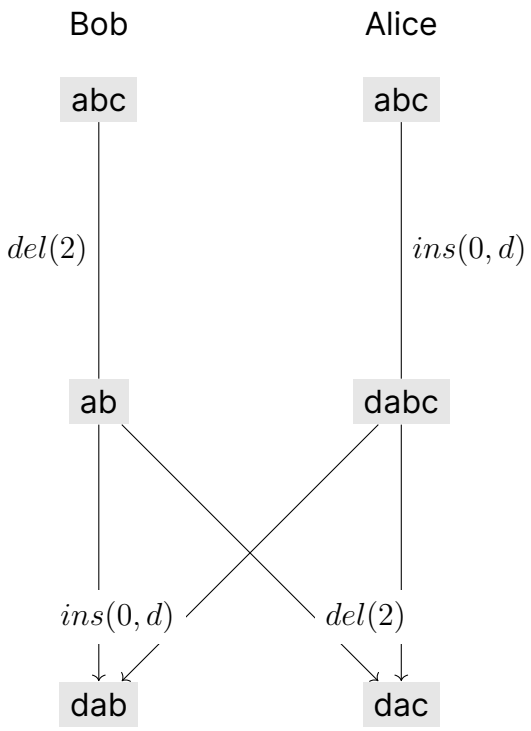

Figure 3.3: Without OT

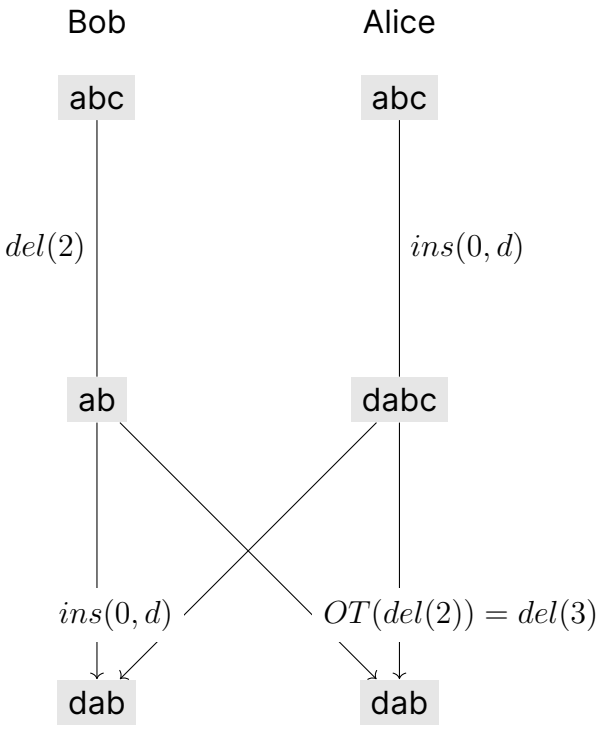

Figure 3.4: With OT

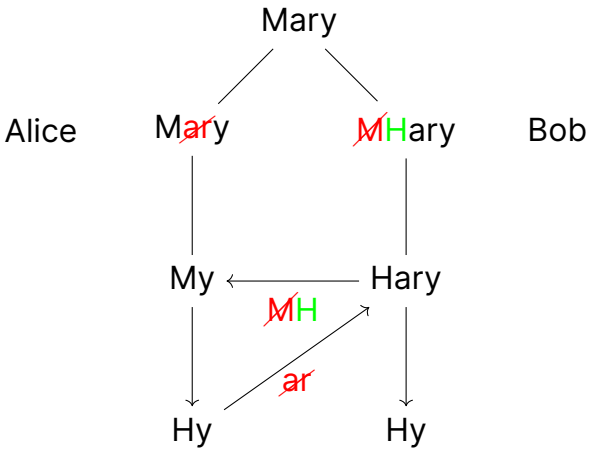

Figure 3.5: OT's solution for the problem in figure 3.2

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) [55]. 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 [52].

In 2011, CRDT was formally defined [52], 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 establish a solution that perfectly fits their needs.

Figma

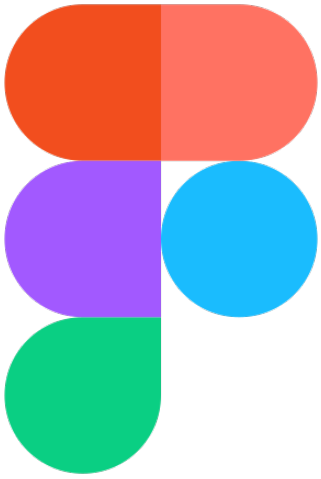

Figure 3.6: Figma logo

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 [58].

The engineering team behind Figma avoided following the same footsteps of Google Docs in using OTs, which, while performant and required less memory, were too complicated to implement and scale [63]. Instead, they decided

to implement their real-time collaboration features using a system inspired by CRDT. Since they already had a centralized server and a database acting as the source of truth, they were able to remove much of the complexity of CRDTs while maintaining their original concept [63]. In particular, the server decides which changes are applied and which ones are discarded based on the timing of these changes and their order. This mode of operation can be viewed as a modified version of CmRDT.

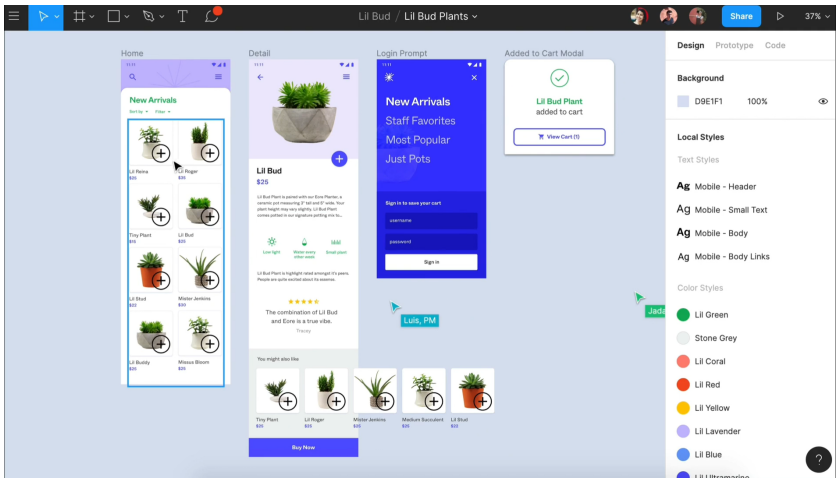

Figure 3.7: Figma collaborative interface

Excalidraw

Figure 3.8: Excalidraw logo

Excalidraw is an open-source virtual collaborative whiteboard tool that lets you easily sketch diagrams that have a hand-drawn feel to them [50]. Contrary to Figma, Excalidraw is Peer-To-Peer (P2P), that is, the server is not the source of truth and used merely for connecting the different clients and propagating changes [6]. On top of that, the exchanged data is encrypted end-to-end, meaning that the server has no access to clients' changes. Excalidraw uses a variant of Convergent Replicated Data Type (CvRDT) with the whole state being sent over and compared to the local state before applying changes.

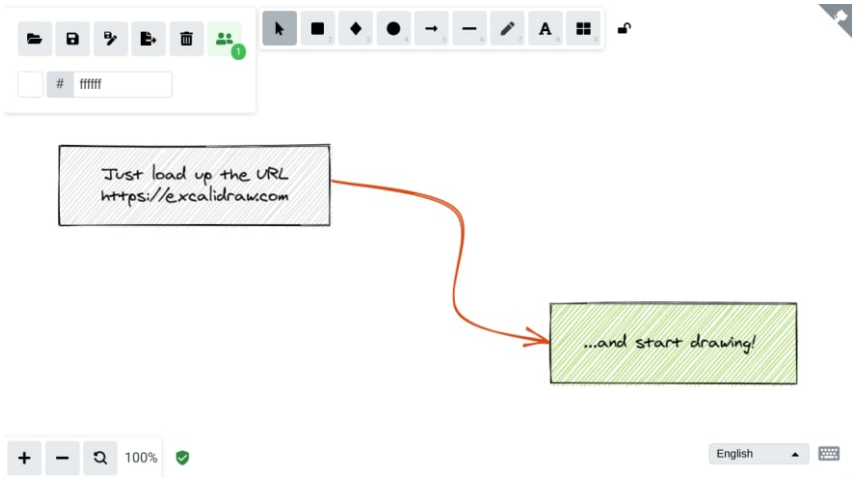

Figure 3.9: Excalidraw collaborative interface

What is peculiar about Excalidraw's example is the way they handle concurrent changes of the same element. Each element has a version attached to it and a hash of that version, which are compared to know which changes to apply and which ones to discard. If two clients edit the same element at the same time, Excalidraw does not try to merge those changes, instead, it picks whichever change came first [6]. Figure 3.10 illustrates the algorithm implemented by Excalidraw for merging conflicting states.

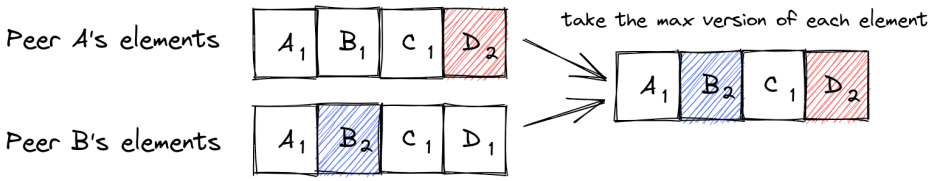

Figure 3.10: Excalidraw collaboration algorithm as explained on their blog [6]

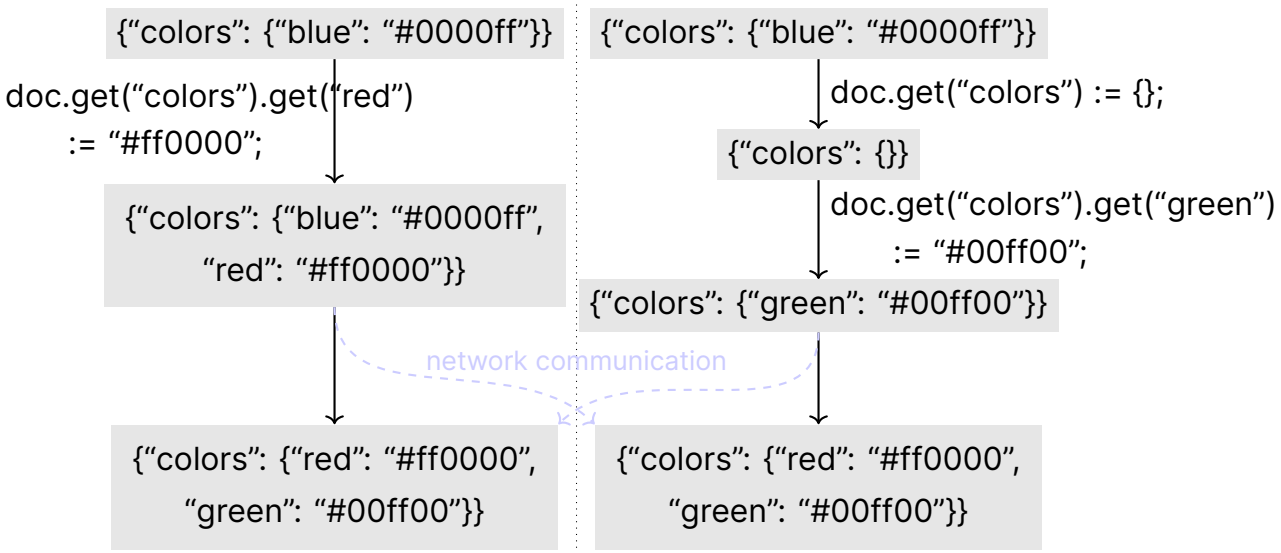

Figure 3.11: Example of Automerge algorithms

Automerge

Automerge is a set of algorithms for applying CRDT concepts to a JavaScript Object Notation (JSON) data structure without having prior knowledge of the inner working of the data structure or the infrastructure used to transmit changes. It works by registering all the changes applied locally to the state and sending these operations to remote clients, while employing a variety of CRDT algorithms for resolving conflicts [27]. Automerge also comes with a ready-to-use JavaScript library. Figure 3.11 illustrates one of the algorithms provided by Automerge.

Compared to Figma, Automerge does not require any central source of truth. On top of that, it tries to resolve conflicts rather than picking one of the changes based on the version or timestamp, as it is the case in Figma and Excalidraw.

Centige

Centige is a discontinued collaborative web app builder by the author. It was mainly a research project, started in late 2019 and developed throughout 2020, around the same time as Excalidraw. The web application used a collaboration model very similar to that of Figma, as it relied on a centralized server acting as the source of truth and used Operation-based CRDT for transmitting changes to the other clients. Each client had a Conflict-free Replicated Data Type (CRDT) and on ever change, the new local state would be compared with the older one, in a model similar to that of State-based CRDT, before generating a set of operations that are sent to the server. The latter would compare the operations' hash before propagating them to rest of the connected clients. All of these computations happened within milliseconds and supported offline editing to some extent.

3.1.4 Our choice

After reviewing the literature on real-time collaboration and having researched the different implementations of the feature in varying products, it is time to choose a fitting design pattern for real-time collaboration in Merebase. It is important to keep in mind that the design of the collaborative aspect of our application greatly influences the rest of the architecture. Considering the ease-of-use of Conflict-free Replicated Data Type (CRDT) and the sheer volume of the scientific literature around it, coupled with the community around it and our knowledge of the algorithm, we chose to implement a variant of CRDT.

In particular, we are going to implement Operation-based CRDT (CmRDT) in approach similar to that of Excalidraw with each element having a version attached to it. However, in contrast to Excalidraw's implementation, we are going to rely on a server a source of truth. This choice eliminates any complexities

with P2P technology and keeps the performance smooth as we do not have to perform any comparisons or calculations on the client's device. Therefore, we achieve one of our non-functional requirements—speed.

3.2 Architectural patterns

To better define our technical requirements for the implementation in the upcoming chapter, it is necessary to set a list of architectural patterns to follow. While such patterns are not set in stone, they are helpful guidelines for developing the application and avoiding any traditional pitfalls.

An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture and thus, various architectures can be mixed and used together.

3.2.1 Common patterns

As architectural patterns aim to solve various problems in different contexts, the term can be vaguely used to designate differing concepts. On one hand, there are patterns such as Model-View-Controller (MVC) and Model-View-ViewModel (MVVM), which define the relation between the data and the presentation layer or the User Interface (UI). On the other hand, there are patterns such as the client-server architecture and the monolithic architecture that define the relation between the different access layers of an application.

The MVC pattern, illustrated in figure 3.12, tends to get paired with a monolithic architecture, illustrated in figure 3.13, with a single server acting both as the data access layer and the presentation layer. While this pattern is the most common in the software world, it is starting to fall into disuse with the proliferation of UI libraries, and microservices.

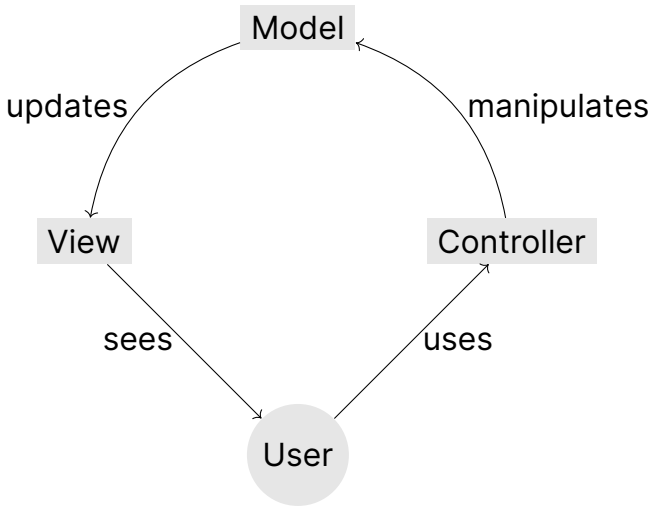

Figure 3.12: The MVC pattern

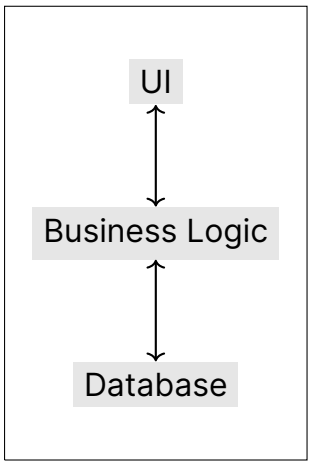

Figure 3.13: The monolith architecture

More recent web applications are usually built on the MVVM pattern with a modular, multitiered client-server architecture. This focus on modularity helps in scaling and maintaining software with ease.

Multitier architecture

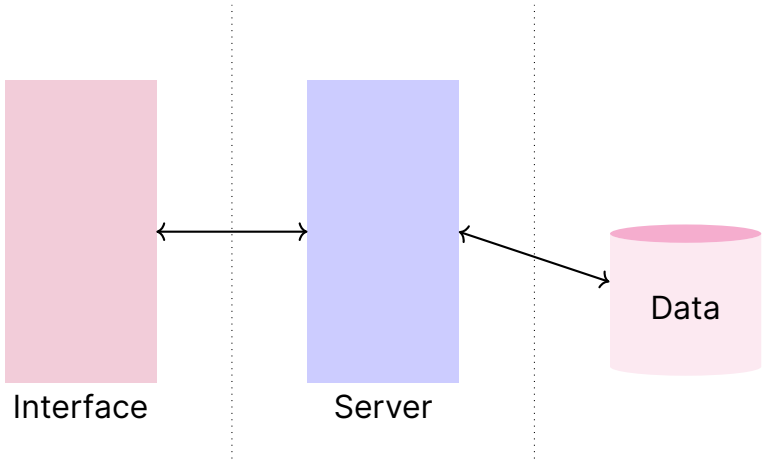

Figure 3.14: The three-tier architecture

Multitier architecture or *n*-tier architecture is a multilayer client-server architecture in which presentation, logic, and data are physically distinct layers. This architecture guarantees flexibility and reusability for developers since layers can be reused, removed, or expanded, without reworking the whole application. Furthermore, it is easily scalable in case of a surge in usage or a system failure.

The most common multitier architecture is the three-tier one, illustrated in figure 3.14. It is composed of:

Client It is often also called "front-end" or the interface. This is the tier that the end user interacts with directly. It is also the one where the MVVM pattern is implemented.

Server It is also called the "back-end". This is where the data processing and business logic happens.

Database This is the data tier. It represents any system used for storing data, be it a database, or a logging storage system.

For more advanced use cases, it is not uncommon to see an even more distributed *n*-tier architecture. If anything, such system are quickly becoming the norm in enterprises, under the name "microservices". In such architectures, each service is a distinct and separate tier on its own. This architecture, however, can quickly transform from being an advantage for quick iteration and scalability to a nightmare in maintenance [42]. Therefore, each application has to find a sweet spot between scalability and flexibility, and maintainability.

MVVM pattern

Figure 3.15: The MVVM pattern

Model-View-ViewModel (MVVM) is an architectural pattern for developing application that separates the development of the User Interface (UI) (the *view*) from the development of the backend logic (the *model*) so the view is independent of the model. The view model has the role of exposing the data of the model to the view, that is, it works as data converter.

MVVM was originally a variation of the Presentation Model design pattern. It was invented in Microsoft to supported their event-driven UIs, and it was incorporated into their Windows Presentation Foundation [53].

Most recent UI frameworks, such as React, Angular, Vue.js, and Svelte, implement some variation or another of MVVM [25].

3.2.2 Our choice

Our choice of architectural patterns is largely guided by the architecture of our real-time collaboration feature and the non-functional requirements defined before, in particular speed, performance, scalability, and developer experience.

In order to achieve these goals, we can identify the following components of our architecture:

- A front-end application that can stay online and functional even when the rest of the system fails;
- A separate view and data layers for a good developer experience;
- A server that can be easily scaled or replaced in case of failure or in the case of a surge in usage;
- Separate servers for our application and for our users' database, since an increase of demand on our application's API does not translate to an increase of demand on the application itself;
- A cloud-hosted database to act as the source of truth for all of our system components;
- An intermediate data store or pub-sub server for communicating between the different servers and keeping them in sync;

These considerations make it clear that an n-tier architecture is the most suitable for our application. Figure 3.16 illustrates our choice of architecture.

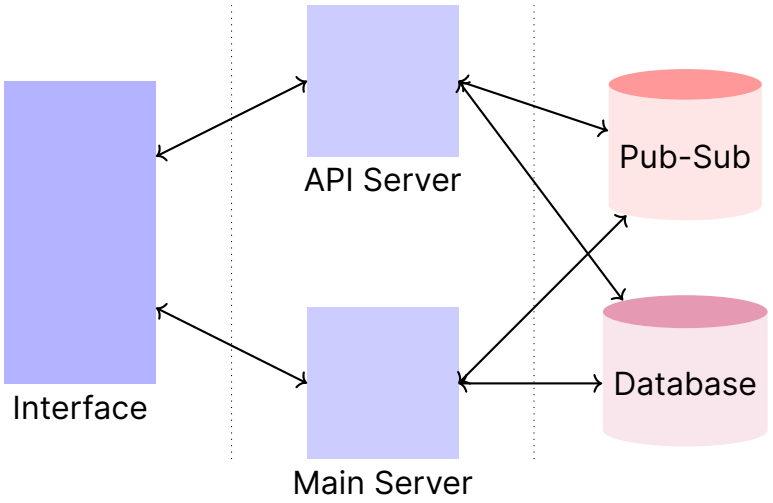

Figure 3.16: Our *n*-tier architecture

We rely on two servers to match our users' needs. The main server handles everything from business logic, data processing, and real-time collaboration, while the API server is used to compile our users' visual databases and serve them on demand in JSON format. This separation of concerns eliminates any system failure or scalability issues. The Pub-Sub store is used to communicate changes between the main server(s) and the API server(s). This is even more important when a single server could no longer match the demand, and we are required to scale our servers up. In such cases, the Pub-Sub store would keep all of our servers in sync.

3.3 Detailed architecture

With our architectural patterns selected and our requirements fully defined, we can, now, proceed to producing detailed diagrams of our application. In the following sections, we will present our application's class, and sequence diagrams.

3.3.1 Class diagrams

Figure 3.17 represents our application's general class diagram.

3.3.2 Sequence diagrams

Sign up

In order to mitigate the security issues of storing passwords, we follow the model of passwordless sign up and log in. Instead of asking our users for a password, which may not be as secure and unique as one might hope, we generate a One-Time Password (OTP) and email it to the user. The OTP is stored for a limited time in our database and it is removed once used. The user, therefore, has to click the confirmation link in their email in order to verify their identity.

Figure 3.18 represents the sequence diagram for "sign up".

Log in

Figure 3.19 represents the sequence diagram for "log in".

Confirm email

Upon receiving a confirmation email, the user has to click the confirm button, which sends a request to the server with the OTP. The server validates the password and either accepts it and authenticates the user, or refuses it and alarms them about the error. Since having users check their email every time is a poor User Experience (UX), the server uses time-limited browser cookies for storing the user's authentication state.

Figure 3.20 represents the sequence diagram for "confirm email".

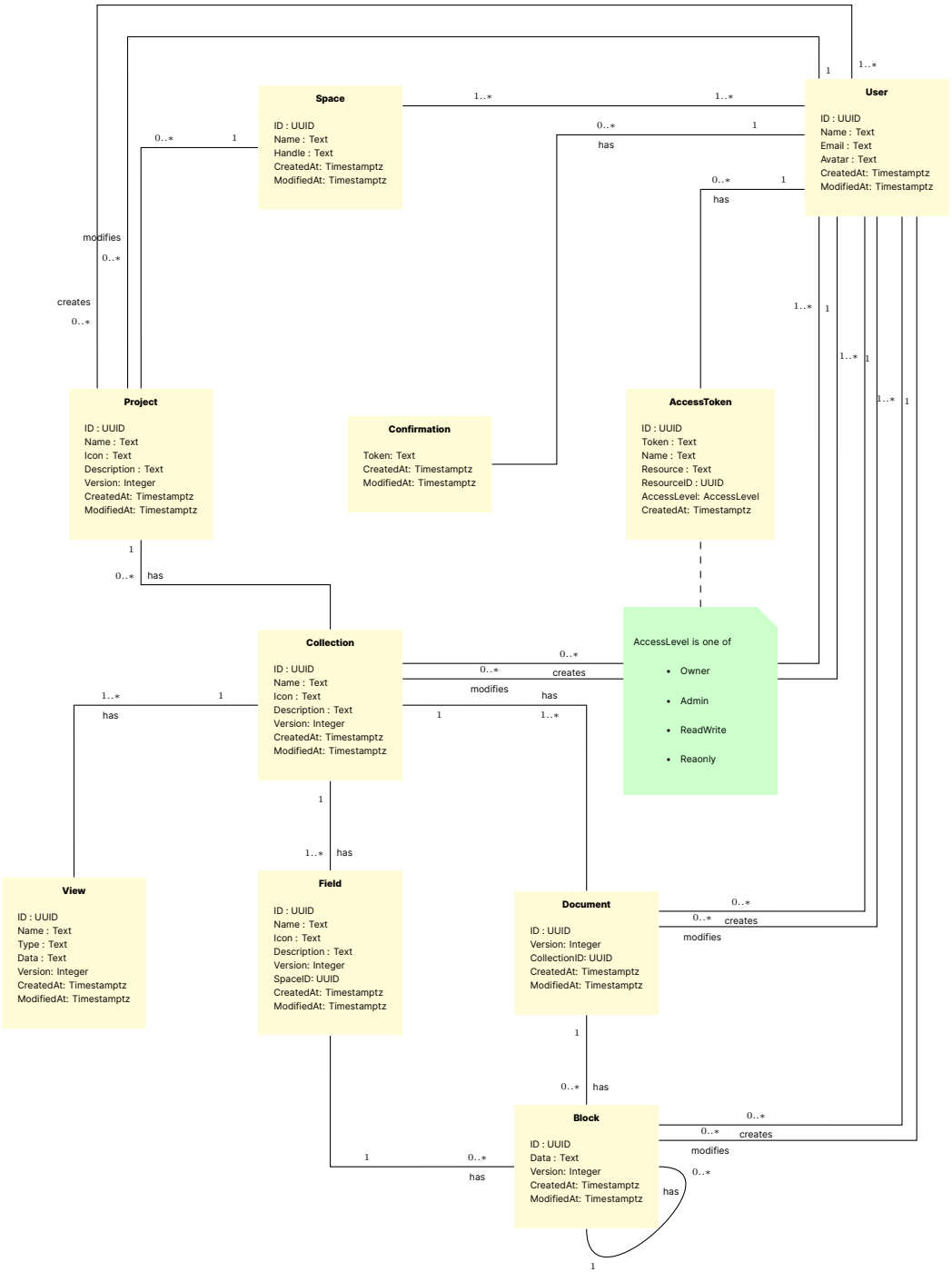

Figure 3.17: General class diagram

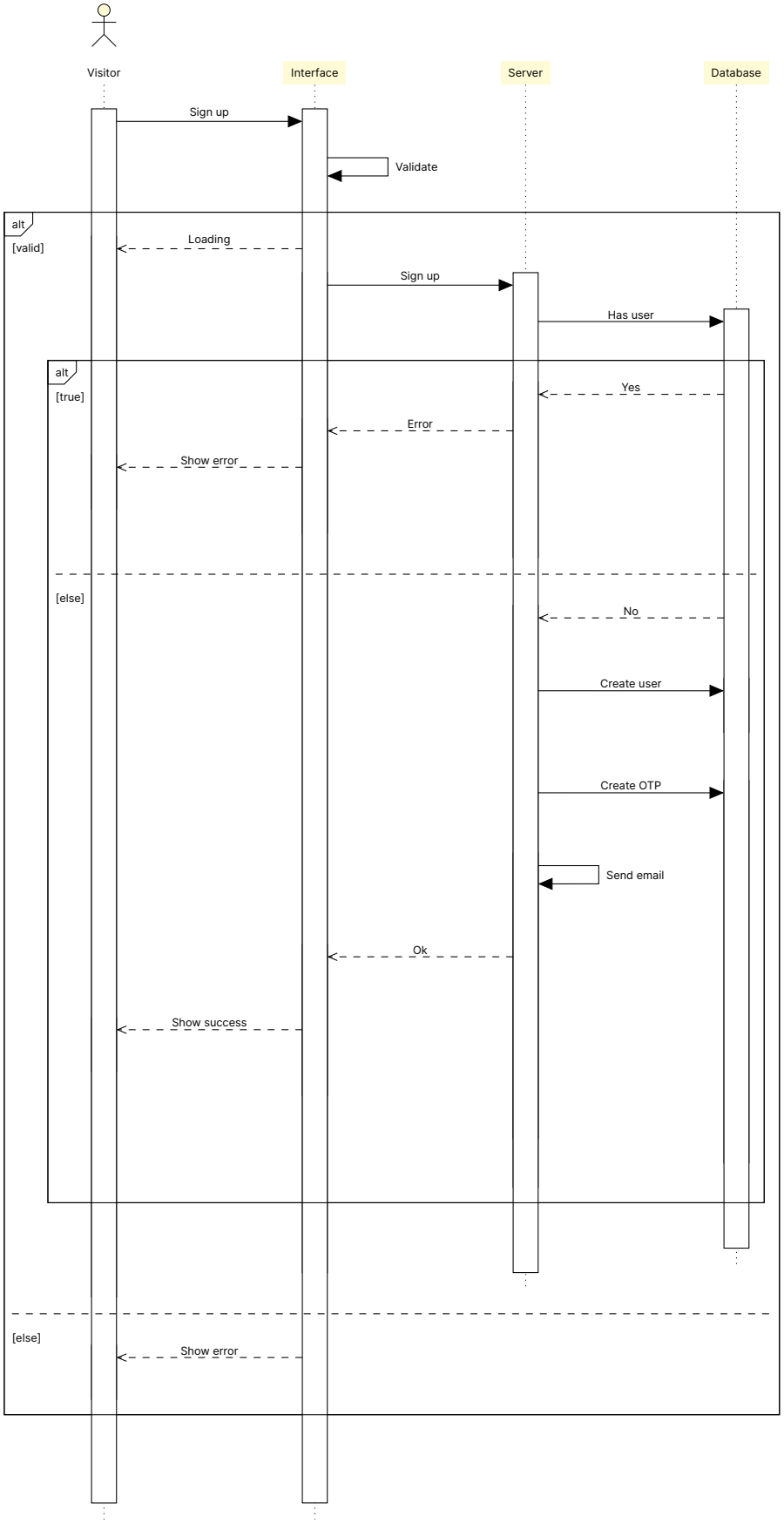

Figure 3.18: “Sign up” sequence diagram

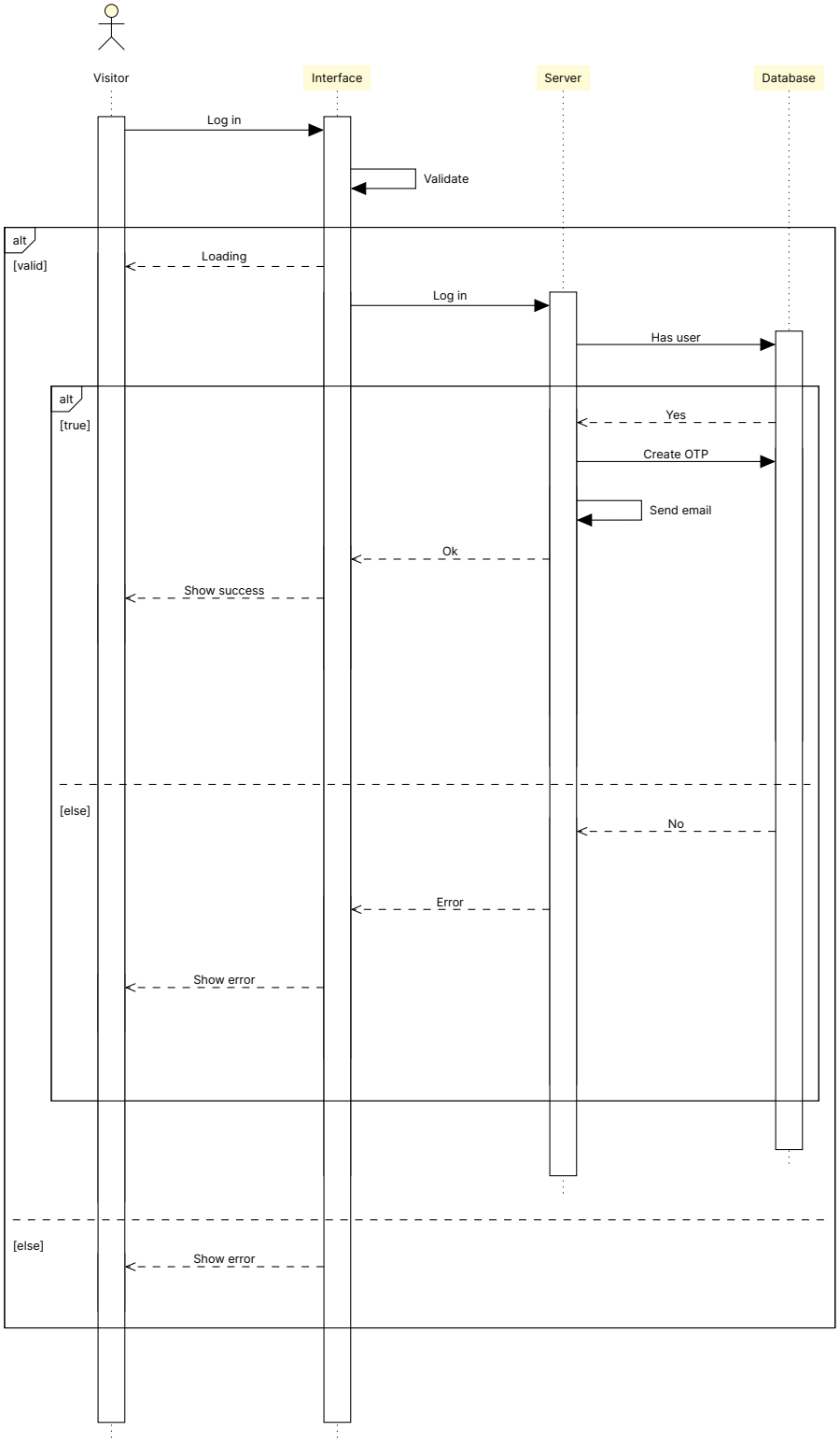

Figure 3.19: “Log in” sequence diagram

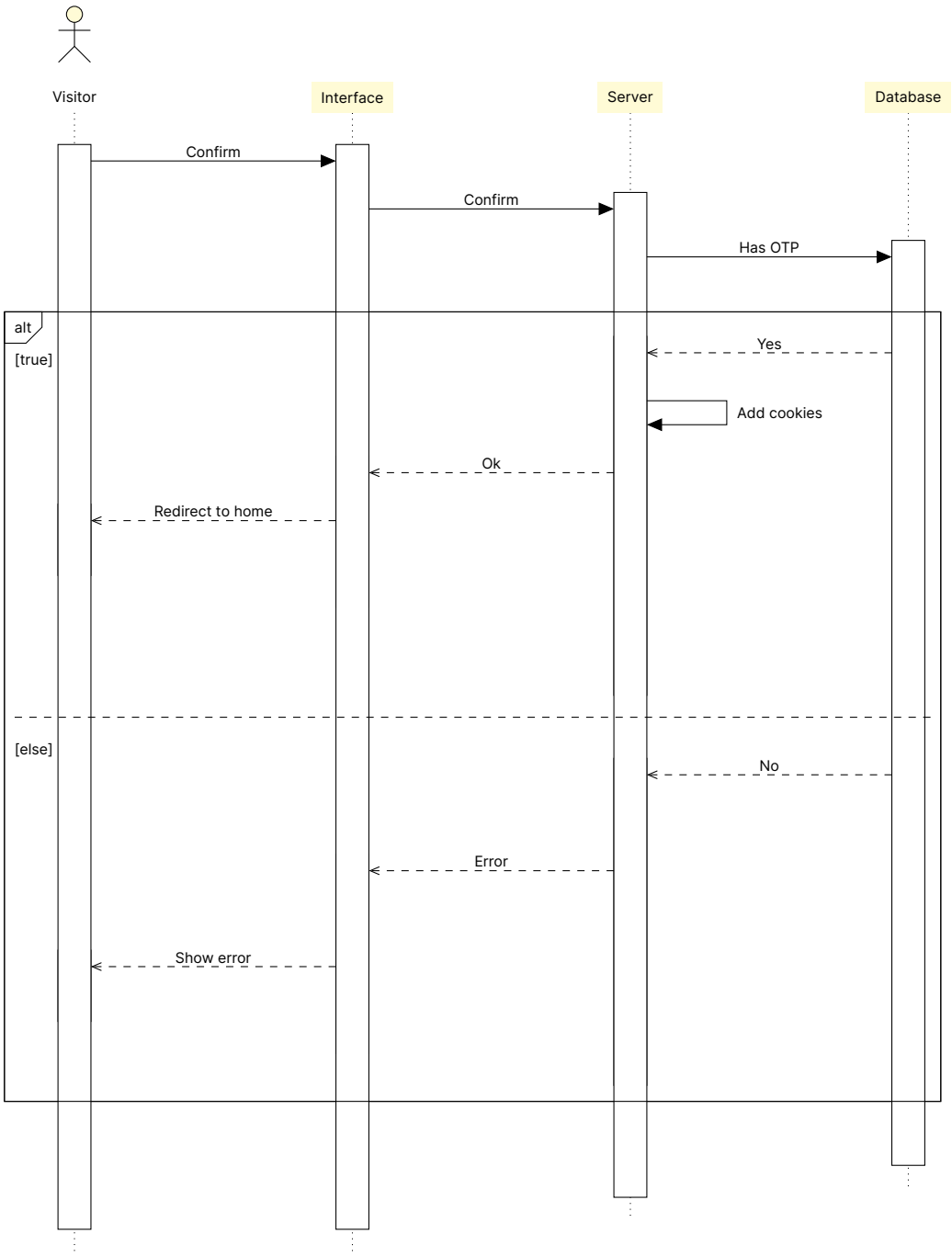

Figure 3.20: “Confirm email” sequence diagram

Create workspace

Whenever a user creates a new workspace, we only ask for a name, and we intentionally create everything they may need to start working. This is because an empty workspace is intimidating and can slow on-boarding process for new users. The same process is followed whenever the user creates a project, a collection, or a document.

Figure 3.21 represents the sequence diagram for "create workspace".

Create collection

Figure 3.22 represents the sequence diagram for "create collection".

Create field

Figure 3.23 represents the sequence diagram for "create field".

Update block

Updating a block is the most common task in our application. It happens within milliseconds and it is quickly propagated to all the connected collaborators. This work by first updating the local state in order to give the effect of an immediate change. Then, the updated block is sent to the server, which would compare its version with the one stored in the database—our source of truth. The equality of versions means that the user has an up-to-date local state, and therefore, their change is accepted and sent to the other users. However, the unequality of versions indicates that the user's local state is out-of-date. Since they are online, they will eventually get the missing updates. So, their change is refused. The same method applies to updating fields too.

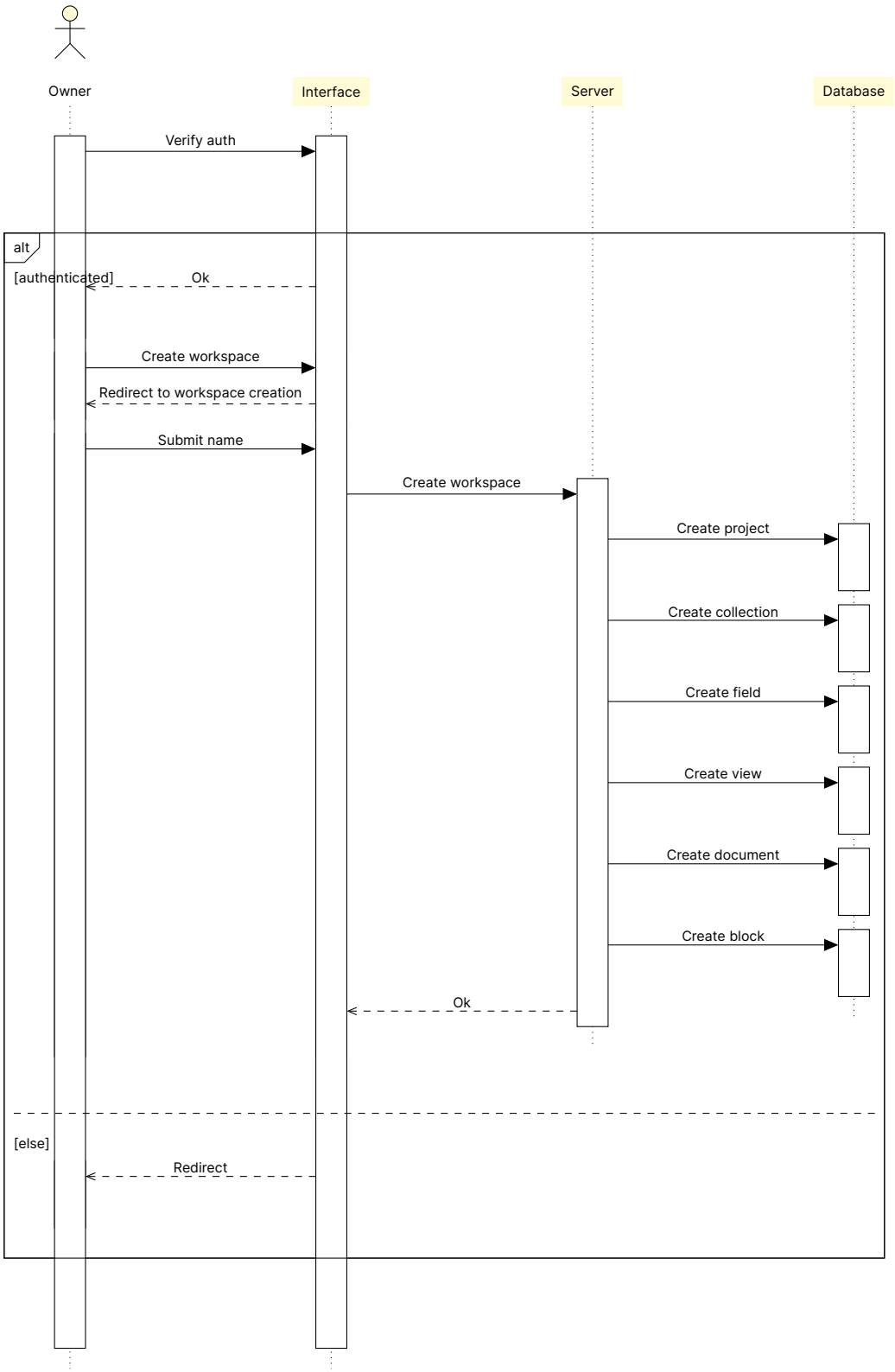

Figure 3.21: “Create workspace” sequence diagram

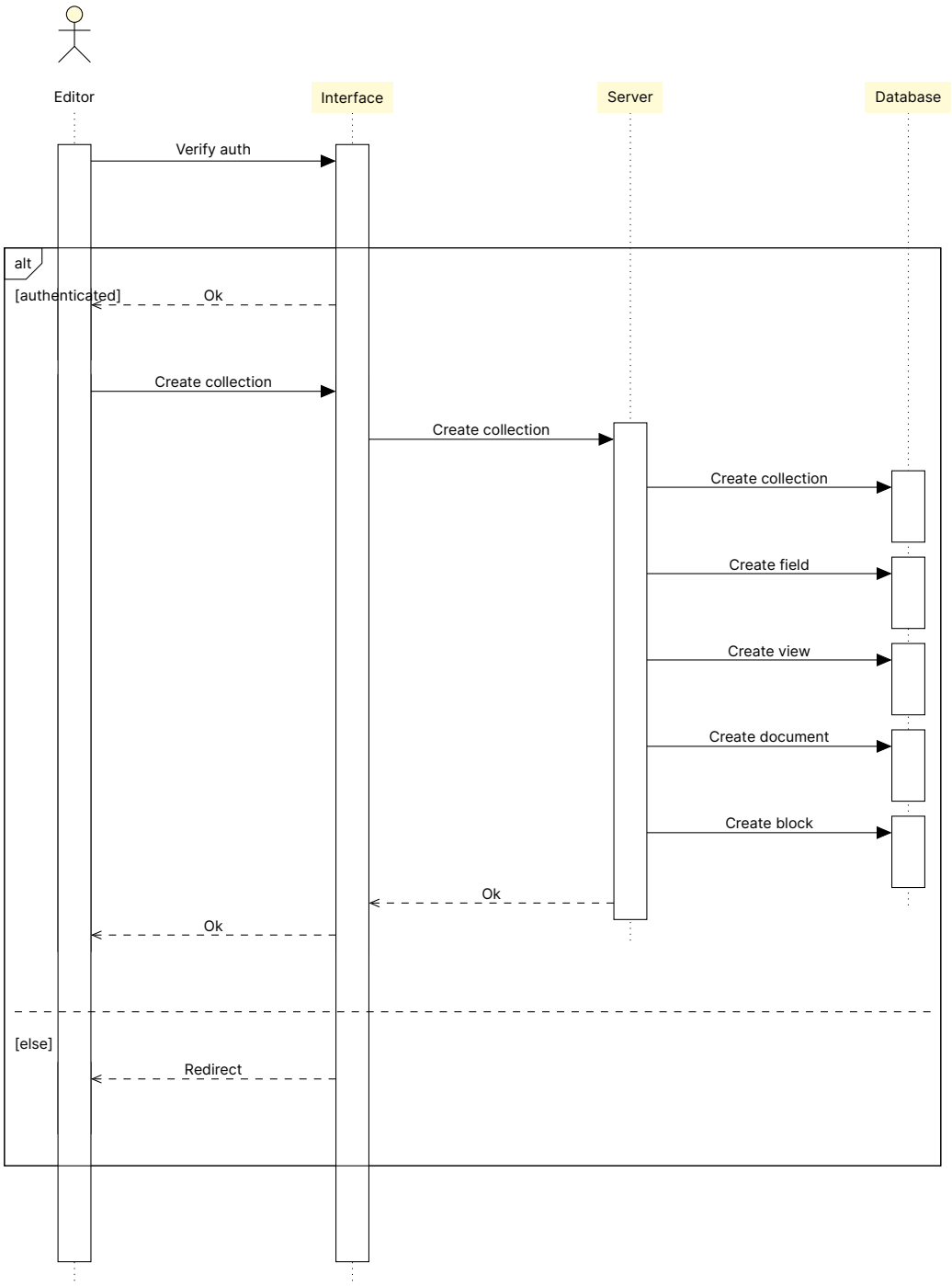

Figure 3.22: “Create collection” sequence diagram

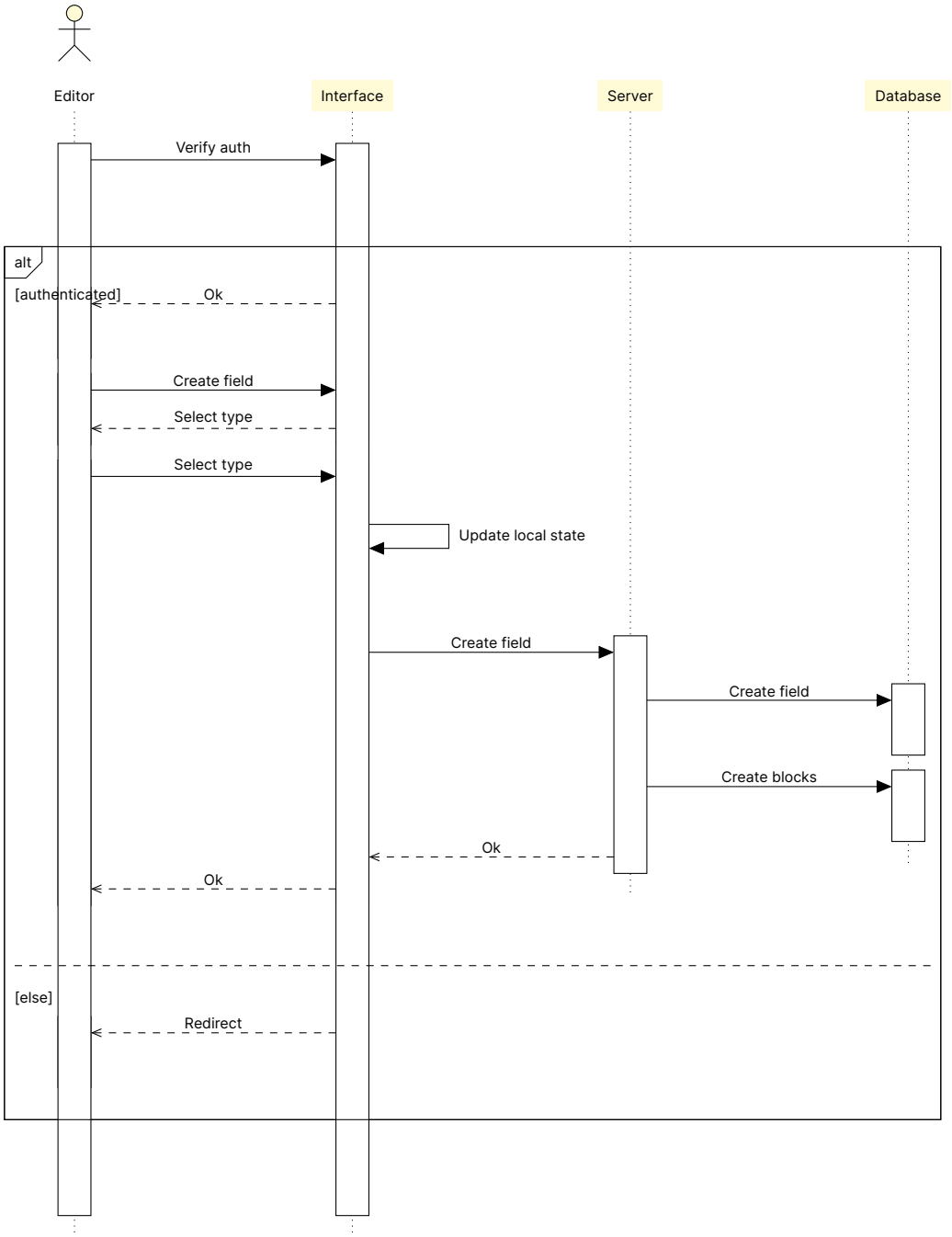

Figure 3.23: “Create field” sequence diagram

Figure 3.24 represents the sequence diagram for “update block”.

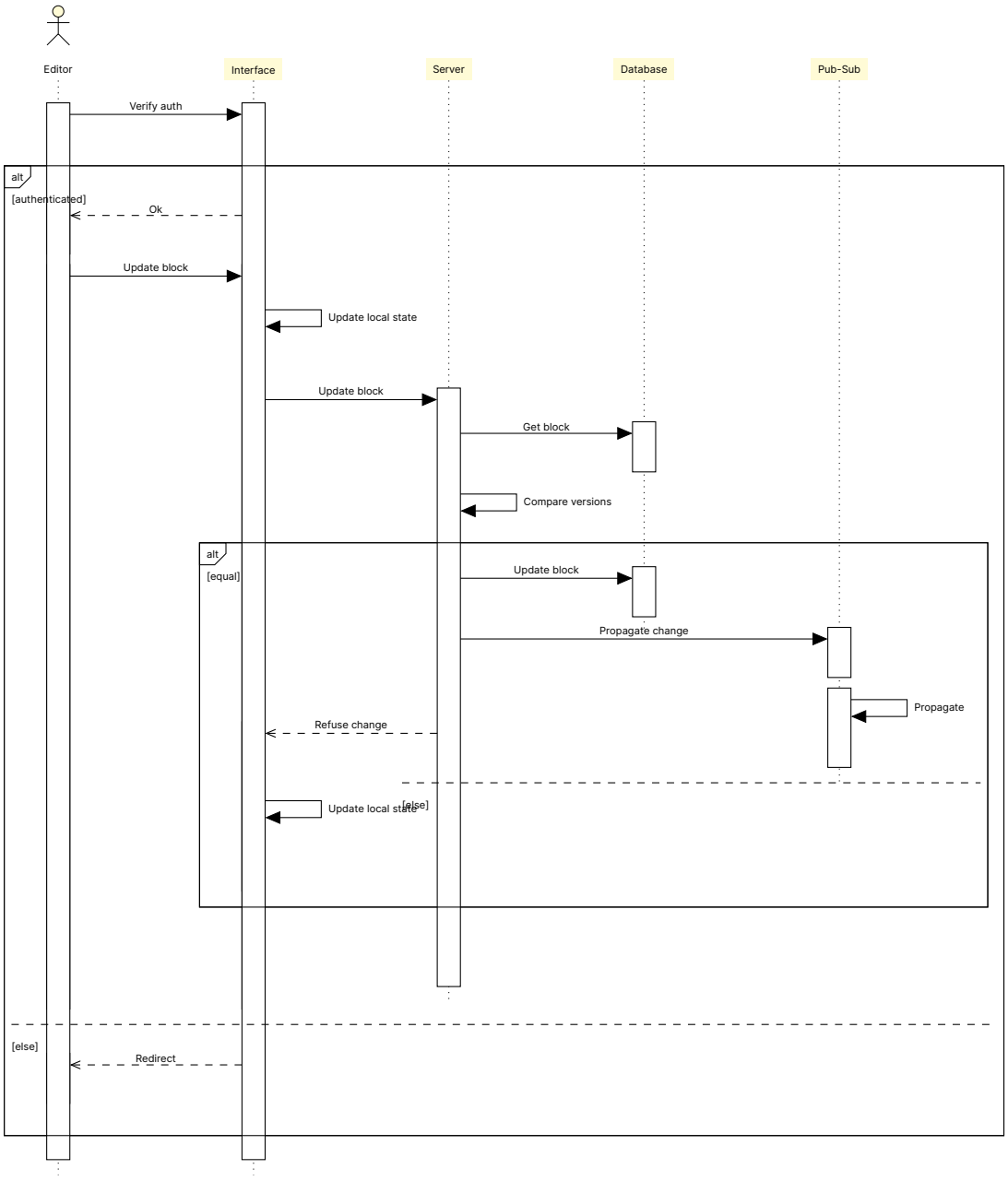

Figure 3.24: “Update block” sequence diagram

3.4 Conclusion

In this chapter, we analyzed the possible architectures of the different aspects of our applications, compared them, and made an objective choice. Then, we proceeded to present, in details, the architecture of our application.

Within the next chapter, we are going to glue all the pieces of research and theory together, and bring our application to life.

4 Implementation

As exciting as ideas may be, they are worthless if they never leave the books to the real world. In this chapter, we are finally bringing our application to life. We will start, as always, by exploring the possible choice for our implementation, proceed to justify our choice, and finally present our application.

4.1 Technology stack

So far, within the last three chapters, we only made a choice after objectively analyzing the existing possibilities, comparing them, and picking the one that suited our needs the best. This chapter is no different.

In the last chapter, we decided that our application will follow an *n*-tier architecture and a MVVM client. Therefore, we need to start by picking the right technology for each tier. Our requirements, defined in the second chapter, should always be our main focus. That is, speed, performance, accessibility, security, and scalability.

4.1.1 The *n*-tier stack

Client stack

With the MVVM pattern in mind, we have multiple options to pick from. The selection of these options is based on our past-expertise in them.

React An open-source front-end JavaScript library for developing UIs created by Facebook starting in 2013 [66].

It advocates declarative, componenet-based views and relies on a virtual Document Object Model (DOM) to selectively update the browser's DOM based on state changes, thus, achieving better performance and a greater developer experience [49].

Vue.js An open-source front-end JavaScript framework for developing UIs created by Evan You starting in 2014 [31].

In a similar fashion to React, Vue.js also relies on a virtual DOM and advocates a component-based architecture.

Svelte An open-source front-end JavaScript *compiler* created by Rich Harris in 2016 [38]. Contrary to React and Vue.js, Svelte is *not* a run-time library. Instead, it operates like a language on its own that gets compiled down to efficient JavaScript code. Therefore, it does not rely on concepts such as the virtual DOM and minimizes the final size of the application.

While React and Vue.js are similar, Svelte has a completely different approach that can offer better performance. However, speed is not everything—the size of the community matters much more. On top of that, Svelte's performance metrics can be achieved using React too, albeit with more configuration. More concretely, React's monthly downloads were 54 times Svelte's downloads during the same month and 5 times Vue.js' downloads [9]. Library-wise, one can

find up to 160,000 open-source libraries for React, compared to 50,000 for Vue.js and 3,000 for Svelte [39].

Taking all of these points into consideration, React seems the perfect choice for our application.

Server stack

Since they have to handle thousands of computationally expensive requests with millisecond-latency, our servers' technology stack has to be as performant as possible. However, while performance is the top priority, it is also important to pick a stack with large community support, otherwise our implementation might have a poor developer experience, and it might face maintainability issues in the future.

The choice of the technology is also limited by our expertise. A performant language or framework with no expertise would as good as or maybe even worse than a slow language with experience in using it.

Node.js An open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine[1] and executes JavaScript code outside a web browser. It has an event-driven architecture that aims to optimize throughput and scalability in web applications, especially ones with real-time requirements [37].

Go A statically typed[2], compiled programming language created at Google. Its syntax is similar to C, but with memory safety, garbage collection, structural typing, and concurrency [32]. Being a compiled, concurrent programming language, Go usually offers better performance than Node.js [23].

[1] The JavaScript engine used in Google Chrome
[2] Type safety is checked during compilation

Rust A multi-paradigm[3] programming language designed for performance and memory safety, which it achieves without garbage collection [17]. It is notable for its extreme speed and low usage of server resources even when compared to Go [51].

When comparing our options, Rust wins by sheer performance, while Node.js wins for its large community and the ability to share code between our front-end and our back-end. However, since our application is technically a database on top of another database, performance is an important metric. This is why Go, with its performance and community, is the best choice for our use case.

Database

Merebase is essentially a no-code collaborative database on top of another low-level database. Aside from being regularly queried by our users, our database will act as the source of truth for real-time collaboration. Therefore, the choice of our application's data store must primarily rely on the performance of such store.

MongoDB A source-available[4] cross-platform NoSQL database program [64]. MongoDB uses JSON-like documents with optional schemas. It is developed by MongoDB Inc. and licensed under Server Side Public License (SSPL) [64]. It was initially released in 2009 [33].

MySQL A widely used open-source Relational Database Management System (RDMBS) initially released in 1995 [34].

PostgreSQL A free and open-source RDMBS emphasizing extensibility and SQL compliance initially released in 1985 [1, 46, 48]. It offers various features, including transactions, automatically updatable views, materi-

[3] It supports object-oriented, and functional programming, among others

[4] A type of a licensing in which the source code can be viewed, but with no or restricted modification

alized views, triggers, and stored procedures [1]. It is designed to handle a wide range of workloads, from single machines to data warehouses or web servers with numerous concurrent users. Some notable products built on top of Postgres include Amazon Redshift, a data warehouse service.

Since each user can shape their database however they like, it might be tempting to choose a NoSQL database, such as MongoDB, for our application. However, benchmarks show that SQL databases, such as Postgres and MySQL, perform significantly better [5].

Finally, when it comes to choosing between SQL databases, MySQL is the most taught one in our schools, and it is the most used in enterprise software. However, Postgres or PostgreSQL offers more concurrency features [35, 47], and it was demonstrated to be more performant than MySQL [26]. Furthermore, the latter's open-source, non-commercial, liberal nature makes it more popular among developers online, thus, providing it with more libraries and extensions.

Based on these points, we decided to choose Postgres as our database.

Pub-Sub

Contrary to the choice overload of the other tiers of our stack, the pub-sub tier is easily dominated by Redis.

Redis is an in-memory store [15]. It can be used as a distributed, in-memory key–value database, cache, or message broker. It supports multiple data structures, such as strings, lists, maps, and sets.

4.1.2 Detailed technology profile

TypeScript

TypeScript is a programming language developed and maintained by Microsoft since 2012 [54]. It is a strict superset of JavaScript that adds optional static typing to the language [60]. The use of TypeScript type checking significantly reduces the number of bugs in JavaScript applications.

Yarn

Yarn is a fast package manager for JavaScript initially developed by Facebook [68]. It allows developers to easily use and share open-source JavaScript packages. One of Yarn's features is Workspaces, which allows us to develop multiple packages locally and manage them using a single command.

Next.js

Next.js is an open-source React front-end framework created by Vercel that enables functionalities such as server-side rendering and generating static websites for React based web applications [36]. This eliminates any performance slowdowns caused by client-rendered JavaScript applications. Furthermore, Next.js provides prefetching links, thus enabling faster navigation between pages, along with code-splitting, which only sends the JavaScript code needed to run a page instead of sending the whole application [36].

Zustand

Zustand is a small, fast and scalable state-management library for React. It features an observer pattern, reduces unnecessary React re-renders, and keep the application performant [43].

Reakit

Reakit is a lower level component library for building accessible high level UI libraries, design systems and applications with React. It strictly follows Web Accessibility Initiative – Accessible Rich Internet Applications (WAI-ARIA) 1.1 standards, which meets our accessibility requirements.

Tailwind CSS

Tailwind CSS is a utility-first CSS framework [56]. Instead of writing CSS styles in separate files, Tailwind CSS provides developers with a set of predefined CSS classes. This makes the development process faster and more consistent. On top of that, this ensures a minimal CSS file size, thus reducing the time needed for loading our application.

A concept similar to that of Tailwind CSS, sometimes labeled Atomic CSS, is currently used in many leading companies, such as Facebook [14].

Windowing

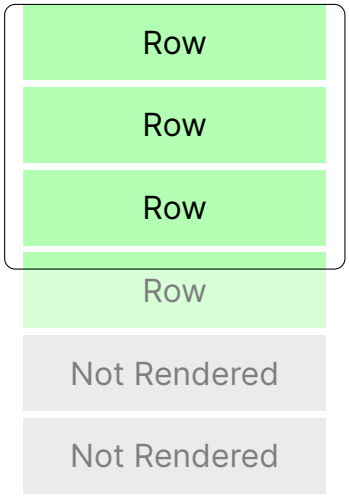

Figure 4.1: Example of the windowing pattern

Windowing or virtual scrolling is a pattern for rendering large lists of data without slowing down the application. It works by only showing the visible elements to the user and instantly rendering any new elements while removing the ones that were hidden [40]. Figure 4.1 shows an example of the windowing pattern on a list of rows. This concept is important in our application since a single page could include thousands of rows of data. In our case, we will be using the library React Virtual.

Websocket

WebSocket is a standardized computer communications protocol that enables interaction between a web browser and a web server with lower latency than HTTP, thus, facilitating real-time data transfer from and to the server [61].

Gin

Gin is a high-performance HTTP web framework written in Go that supports routing, data validation, custom middlewares, and websockets [21].

SQLC

SQLC a command-line tool for generating type-safe Go code from SQL schemas and queries [10]. This is an alternative approach to relying on an ORM library since the latter would result in a network latency that we cannot tolerate.

Dbmate

Dbmate is a command-line tool for performing database migrations [30]. Throughout the development of our application, we had to change the schema a couple of times to accommodate new requirements or findings. Dbmate allows us to safely modify our database.

Docker and Docker Compose

Docker is tool that relies on OS-level virtualization to deliver software in packages called "containers" [11]. Containers are isolated from one another and bundle their own software, libraries and configuration files, making them easily deployable in any environment.

Docker Compose is a tool for defining and running multiple Docker containers at once [41]. With Compose, the application's architecture is defined in a YAML file and easily executed with a single command.

These technologies enable us to quickly set up and launch our different tiers without worrying about compatibility or scalability.

Git

Git is a free and open source distributed version control system, originally developed for tracking changes in the Linux Kernel in 2005 [22]. Compared to other version control system, such as SVN, it is faster, smaller, and easier to use [2].

Visual Studio Code

Visual Studio Code is a source-code editor made by Microsoft [62]. It is tightly integrated with TypeScript since both technologies are developed by the same company.

VS Code's performant code editor inspired part of Merebase's architecture. In particular, the use of windowing.

JetBrains DataGrip

DataGrip is a database management tool developed by JetBrains. It supports a wide range of database engines and offers smart code completion for SQL [12].

XeLaTeX and Tikz

As mentioned in the introduction, this work is typeset using LaTeX. Four chapters later, this is still the case. More specifically, we are using XeLaTeX, which is a typesetting engine derived from the original TeX engine with native support for Unicode [67].

For the diagrams, we used Ti*k*Z, which is a language for producing vector graphics [57].

Conclusion

Our project consists in building a visual collaborative database for small and medium-sized businesses, to be used for content management and to be easily connected to web applications with no required technical knowledge. Three months later, we got there.

Our prior experience with various other applications and our extensive research of similar solutions gave us an idea about what we actually wanted—collaboration, performance, accessibility, and simplicity.

Since we had the freedom of choosing whichever technology we wanted, we made sure to make the right choice that would ensure achieving all of our requirements. We analyzed our options, and we picked what evidently was the best choice for our use-cases. Throughout this, we presented our application's architectural patterns, composition, and conceptual study. We dived deep into collaborative algorithms in theory and practice, and we chose the one that fit our application the most.

Eventually, we explored our technology stack and presented more patterns and ideas that would lead to the desired performance and usability goals.

Finally, and since an idea is worthless unless it is brought to life, we explored our application's implementation with its different screens and features.

No application is ever complete, but we hope that within these three months we were able to create a disruptive solution that would fix one broken part of the internet.

We also hope that this project and this work sets the example for what is possible with the modern web, within a limited timeframe, a single laptop, and an idea.

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

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 [7].

polyfill A piece of code used to provide modern functionality on older browsers that do not natively support it [45].

Bibliography

[1] *1. What Is PostgreSQL?* en. May 2021. URL: https://www.postgresql.org/docs/13/intro-whatis.html (visited on 06/06/2021).

[2] *About - Git*. URL: https://git-scm.com/about/small-and-fast (visited on 06/07/2021).

[3] *Airtable Pricing, Alternatives & More 2021 - Capterra*. URL: https://www.capterra.com/p/146652/Airtable/ (visited on 06/07/2021).

[4] I. Akulov. *Case study: Analyzing Notion app performance*. May 2020. URL: https://3perf.com/blog/notion/ (visited on 06/07/2021).

[5] *Benchmark: MongoDB, PostgreSQL, OrientDB, Neo4j and ArangoDB*. en-US. Feb. 2018. URL: https://www.arangodb.com/2018/02/nosql-performance-benchmark-2018-mongodb-postgresql-orientdb-neo4j-arangodb/ (visited on 06/06/2021).

[6] *Building Excalidraw's P2P Collaboration Feature | Excalidraw Blog*. en. URL: https://blog.excalidraw.com/building-excalidraw-p2p-collaboration-feature/ (visited on 06/05/2021).

[7] *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).

[8] 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).

[9] *Compare npm downloads for react, vue and svelte - npmcharts □*. en. URL: https://npmcharts.com/compare/react,vue,svelte (visited on 06/06/2021).

[10] K. Conroy. *kyleconroy/sqlc*. original-date: 2019-06-21T21:11:35Z. June 2021. URL: https://github.com/kyleconroy/sqlc (visited on 06/07/2021).

[11] *Container Runtime with Docker Engine | Docker*. en. URL: https://www.docker.com/products/container-runtime (visited on 06/07/2021).

[12] *DataGrip: The Cross-Platform IDE for Databases & SQL by JetBrains*. en. URL: https://www.jetbrains.com/datagrip/ (visited on 06/07/2021).

[13] *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).

[14] *Facebook's CSS-in-JS Approach - Frank Yan at React Conf 2019*. en. URL: https://www.infoq.com/news/2020/04/facebook-cssinjs-react-conf-2019/ (visited on 06/07/2021).

[15] *FAQ – Redis*. URL: https://redis.io/topics/faq (visited on 06/06/2021).

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

[17] *Frequently Asked Questions · The Rust Programming Language*. June 2016. URL: https://web.archive.org/web/20160609195720/https://www.rust-lang.org/faq.html#project (visited on 06/06/2021).

[18] 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).

[19] 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.

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

[21] *gin-gonic/gin*. original-date: 2014-06-16T23:57:25Z. June 2021. URL: https://github.com/gin-gonic/gin (visited on 06/07/2021).

[22] *Git - A Short History of Git*. URL: https://git-scm.com/book/en/v2/Getting-Started-A-Short-History-of-Git (visited on 06/07/2021).

[23] *Go vs Node js - Which programs are fastest? | Computer Language Benchmarks Game*. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/go-node.html (visited on 06/06/2021).

[24] *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).

[25] *javascript - MVVM architectural pattern for a ReactJS application*. URL: https://stackoverflow.com/questions/51506440/mvvm-architectural-pattern-for-a-reactjs-application (visited on 06/05/2021).

[26] M. Khushi. "Benchmarking Database Performance for Genomic Data: DATABASE BENCHMARKING." en. In: *Journal of Cellular Biochemistry* 116.6 (June 2015), pp. 877–883. ISSN: 07302312. DOI: 10.1002/jcb.25049. URL: http://doi.wiley.com/10.1002/jcb.25049 (visited on 06/06/2021).

[27] M. Kleppmann and A. R. Beresford. "A Conflict-Free Replicated JSON Datatype." In: *IEEE Transactions on Parallel and Distributed Systems* 28.10 (Oct. 2017), pp. 2733–2746. ISSN: 1558-2183. DOI: 10.1109/TPDS.2017.2697382.

[28] *Long pages*. en. URL: https://en.wikipedia.org/wiki/Special:LongPages (visited on 06/07/2021).

[29] *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).

[30] A. Macneil. *amacneil/dbmate*. original-date: 2015-11-25T18:58:30Z. June 2021. URL: https://github.com/amacneil/dbmate (visited on 06/07/2021).

[31] C. Macrae. *Vue.js: up and running: building accessible and performant web apps*. First edition. OCLC: on1002834117. Sebastopol, California: O'Reilly Media, 2018. ISBN: 9781491997246.

[32] C. Metz. *Google Go boldly goes where no code has gone before*. en. URL: https://www.theregister.com/2011/05/05/google_go/ (visited on 06/06/2021).

[33] *MongoDB System Properties*. URL: https://db-engines.com/en/system/MongoDB (visited on 06/06/2021).

[34] *MySQL System Properties*. URL: https://db-engines.com/en/system/MySQL (visited on 06/06/2021).

[35] *MySQL vs PostgreSQL – Choose the Right Database for Your Project*. en. URL: https://developer.okta.com/blog/2019/07/19/mysql-vs-postgres (visited on 06/06/2021).

[36] *Next.js by Vercel - The React Framework*. en. URL: https://nextjs.org (visited on 06/07/2021).

[37] Node.js. *About*. en. URL: https://nodejs.org/en/about/ (visited on 06/06/2021).

[38] R. H. S. Nov 26 2016. *Frameworks without the framework: why didn't we think of this sooner?* en. URL: https://svelte.dev/blog/frameworks-without-the-framework (visited on 06/06/2021).

[39] *npm*. en. URL: https://www.npmjs.com/ (visited on 06/06/2021).

[40] A. Osmani. *Rendering large lists with react-window*. en. URL: https://addyosmani.com/blog/react-window/ (visited on 06/07/2021).

[41] *Overview of Docker Compose*. en. June 2021. URL: https://docs.docker.com/compose/ (visited on 06/07/2021).

[42] C. Pautasso, O. Zimmermann, M. Amundsen, J. Lewis, and N. Josuttis. "Microservices in Practice, Part 2: Service Integration and Sustainability." In: *IEEE Software* 34.2 (Mar. 2017), pp. 97–104. ISSN: 0740-7459, 1937-4194. DOI: 10.1109/MS.2017.56. URL: https://ieeexplore.ieee.org/document/7888407/ (visited on 06/05/2021).

[43] *pmndrs/zustand*. original-date: 2019-04-09T09:10:06Z. June 2021. URL: https://github.com/pmndrs/zustand (visited on 06/07/2021).

[44] 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).

[45] *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).

[46] *PostgreSQL System Properties*. URL: https://db-engines.com/en/system/PostgreSQL (visited on 06/06/2021).

[47] *PostgreSQL vs MySQL*. en-US. URL: https://www.sumologic.com/blog/postgresql-vs-mysql/ (visited on 06/06/2021).

[48] *PostgreSQL: History*. Mar. 2017. URL: https://web.archive.org/web/20170326020245/https://www.postgresql.org/about/history/# (visited on 06/06/2021).

[49] *React – A JavaScript library for building user interfaces*. en. URL: https://reactjs.org/ (visited on 06/06/2021).

[50] *Rethinking Virtual Whiteboard | Excalidraw Blog*. en. URL: https://blog.excalidraw.com/rethinking-virtual-whiteboard/ (visited on 06/05/2021).

[51] *Rust vs Go - Which programs are fastest? | Computer Language Benchmarks Game*. URL: https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/rust-go.html (visited on 06/06/2021).

[52] 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.

[53] J. Smith. *Patterns - WPF Apps With The Model-View-ViewModel Design Pattern*. en-us. 2009. URL: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern (visited on 06/05/2021).

[54] I. N. S. staff. *Microsoft augments JavaScript for large-scale development*. en. Oct. 2012. URL: https://www.infoworld.com/article/2614863/microsoft-augments-javascript-for-large-scale-development.html (visited on 06/07/2021).

[55] 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).

[56] *Tailwind CSS - Rapidly build modern websites without ever leaving your HTML*. en. URL: https://tailwindcss.com/ (visited on 06/07/2021).

[57] *TikZ package*. en. URL: https://www.overleaf.com/learn/latex/TikZ_package (visited on 06/07/2021).

[58] U. X. Tools. *2020 Tools Survey Results*. en-us. URL: https://uxtools.co/survey-2020/ (visited on 06/05/2021).

[59] 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).

[60] *Typed JavaScript at Any Scale*. en. URL: https://www.typescriptlang.org/ (visited on 06/07/2021).

[61] M. Ubl, E. K. P. October 20th, and 2. C. o. Y. b. m. n. s. t. f. i. t. article. *Introducing WebSockets: Bringing Sockets to the Web - HTML5 Rocks*. en. URL: https://www.html5rocks.com/en/tutorials/websockets/basics/ (visited on 06/07/2021).

[62] *Visual Studio Code - Code Editing. Redefined*. en. URL: https://code.visualstudio.com/ (visited on 06/07/2021).

[63] E. Wallace. *How Figma's multiplayer technology works*. en-US. URL: https://www.figma.com/blog/how-figmas-multiplayer-technology-works/ (visited on 06/05/2021).

[64] *What Is MongoDB?* en-us. URL: https://www.mongodb.com/what-is-mongodb (visited on 06/06/2021).

[65] *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).

[66] *Why did we build React? – React Blog*. en. URL: https://reactjs.org/blog/2013/06/05/why-react.html (visited on 06/06/2021).

[67] *XeLaTeX*. en. URL: https://www.overleaf.com/learn/latex/XeLaTeX (visited on 06/07/2021).

[68] *Yarn: A new package manager for JavaScript*. en-US. Oct. 2016. URL: https://engineering.fb.com/2016/10/11/web/yarn-a-new-package-manager-for-javascript/ (visited on 06/07/2021).