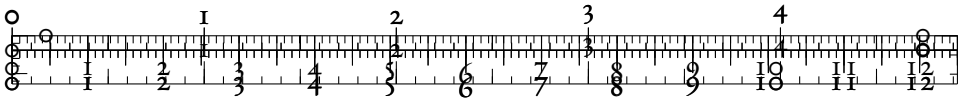
A Collaborative Visual Database

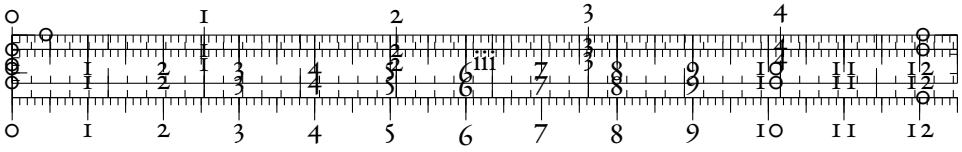
by

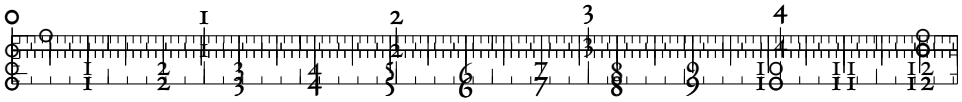
Imed Adel

ESSTHS

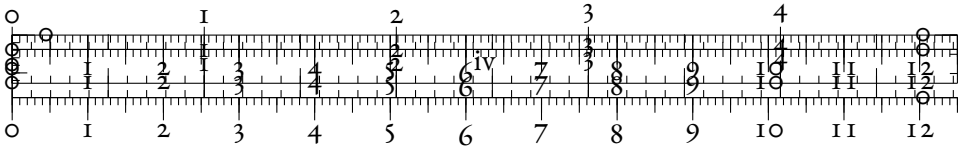


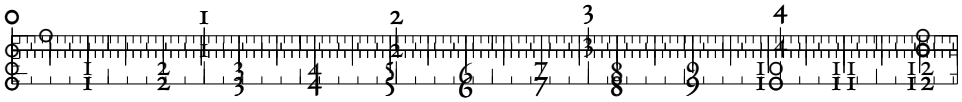
Contents



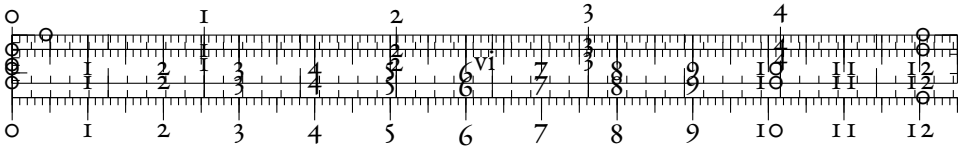


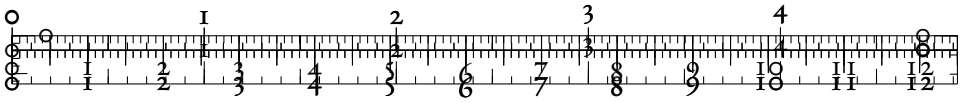
List of Figures





List of Tables





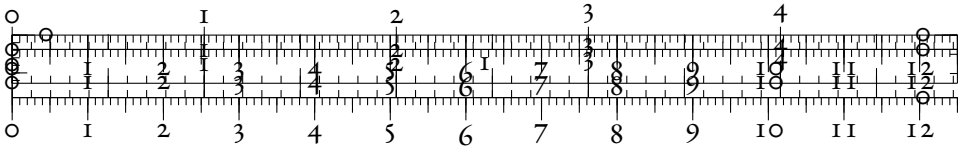
I Conceptual study

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.

1.1 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 [4]. Google Docs is another notorious contender in the space of collaborative editing, with products such as Google Docs and Google Sheets.

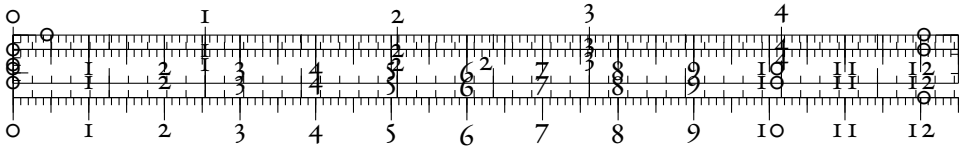
The interest in collaborative software has seen a resurgence since 2020, mainly due to the move to remote work, with companies like Microsoft offering ready-to-use APIs to enable this feature.

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.

1.1.1 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 [5].

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 [3]. 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 [6].



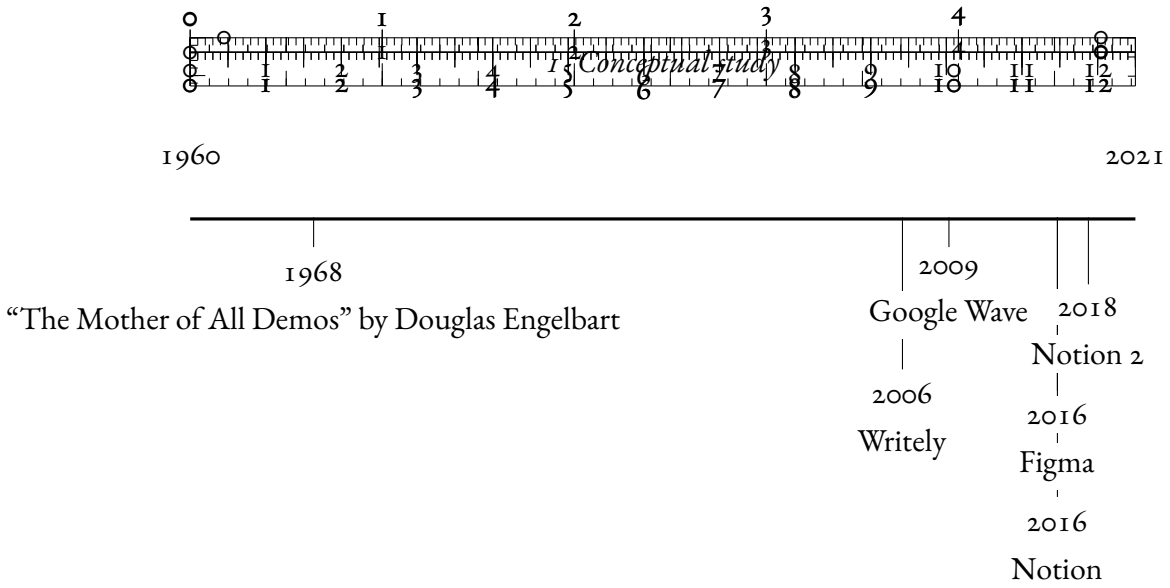


Figure 1.1: Timeline of real-time collaboration

Since 2010, the number of web-based collaborative software has been exploding with successful examples such as Figma and Notion. Figure 1.1 shows the resurgence of products with real-time collaboration.

1.1.2 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 1.2 shows a common example of conflicting changes by two users caused mainly by network latency.

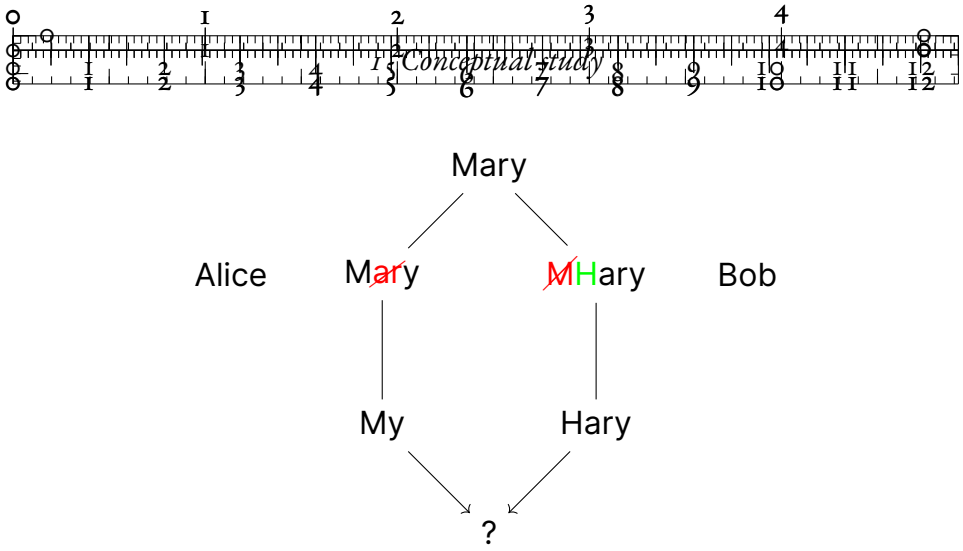


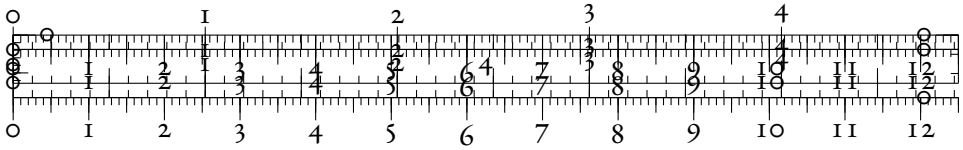
Figure 1.2: Example of a common problem in real-time collaboration

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 [15]. Google Docs has been using OT since at least 2009 [18].

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 1.3 and 1.4 explain the algorithm followed by OT for resolving conflicts. Figure 1.5 illustrates the solution proposed by OT for the problem proposed in 1.2.



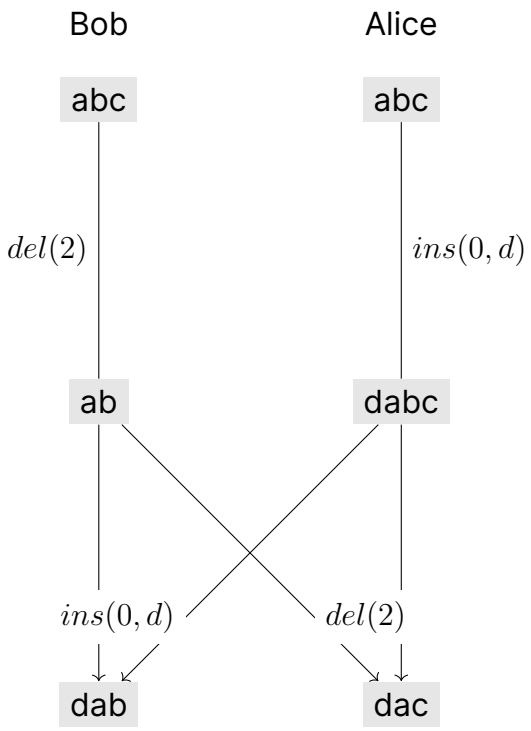


Figure 1.3: Without OT

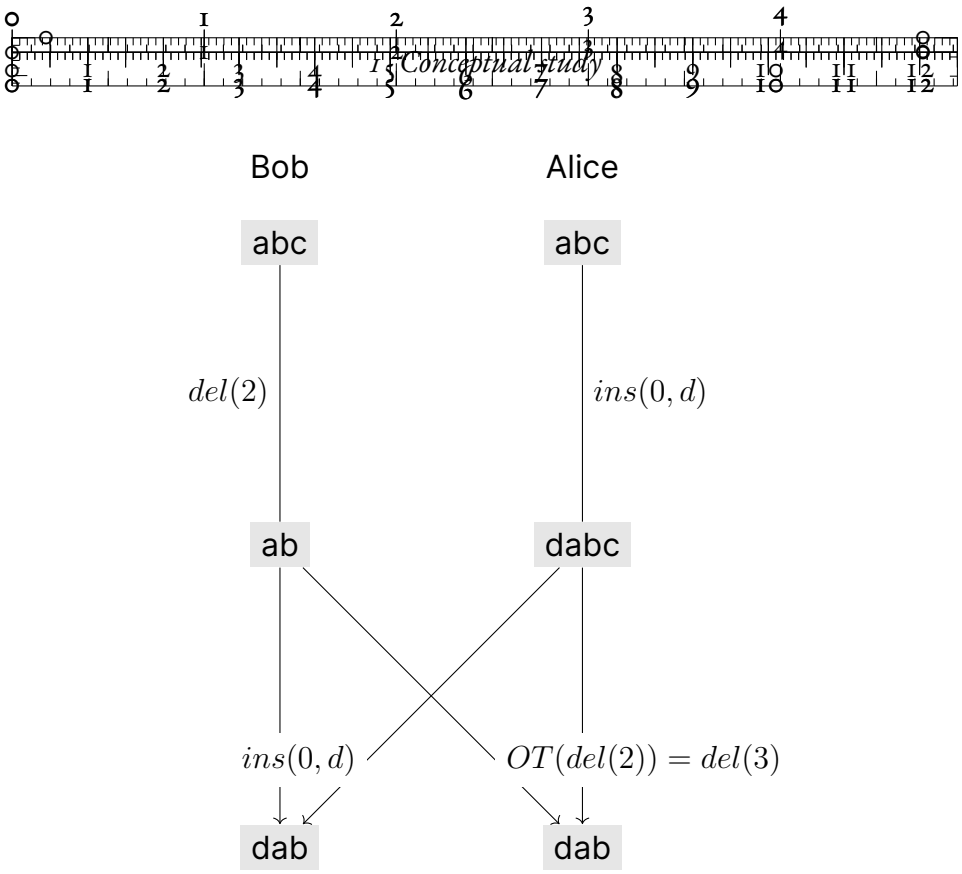


Figure 1.4: With OT

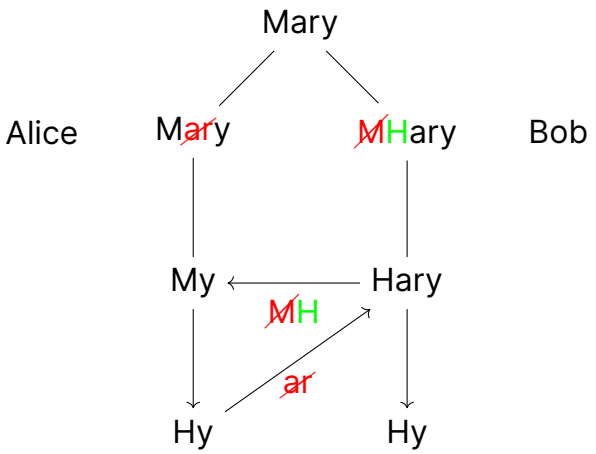


Figure 1.5: OT's solution for the problem in figure 1.2

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) [15]. 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 [13].

In 2011, CRDT was formally defined [13], 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.

1.1.3 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

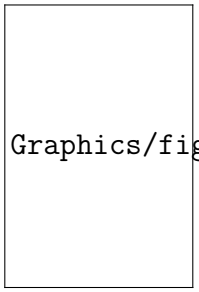


Figure 1.6: Figma logo

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 [16].

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 [17]. 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 [17]. 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.

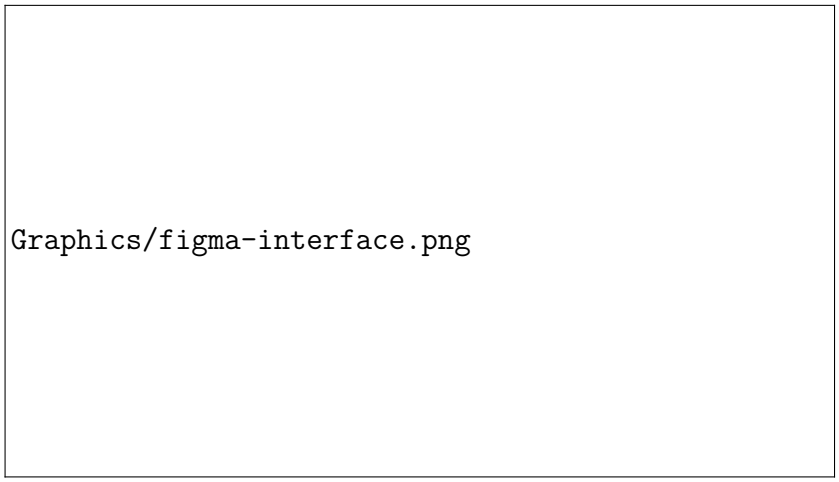


Figure 1.7: Figma collaborative interface

Excalidraw

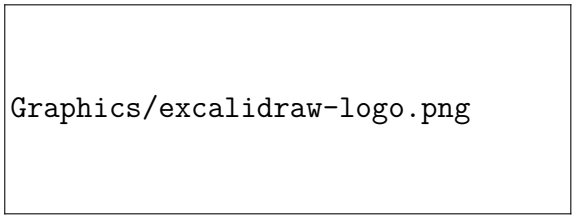


Figure 1.8: Excalidraw logo

Excalidraw is an open-source virtual collaborative whiteboard tool that lets you easily sketch diagrams that have a hand-drawn feel to them [12]. 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 [1]. 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.

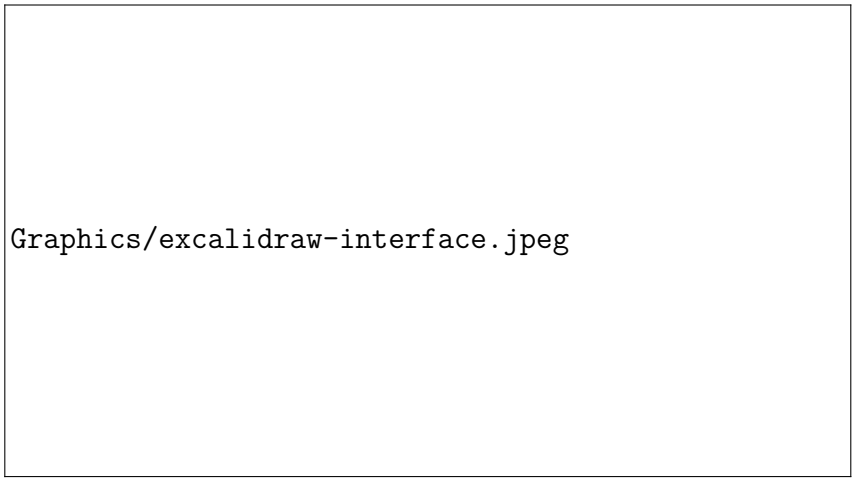


Figure 1.9: Excalidraw collaborative interface

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 [1]. Figure 1.10 illustrates the algorithm implemented by Excalidraw for merging conflicting states.

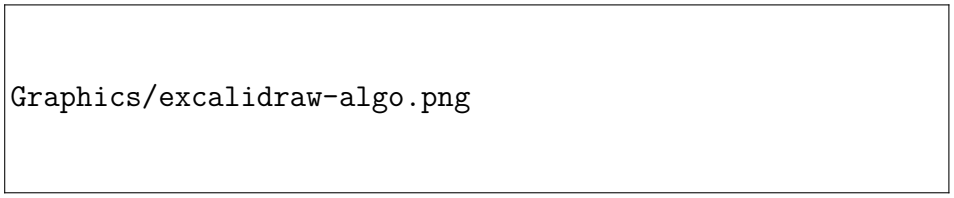


Figure 1.10: Excalidraw collaboration algorithm as explained on their blog [1]

Automerge

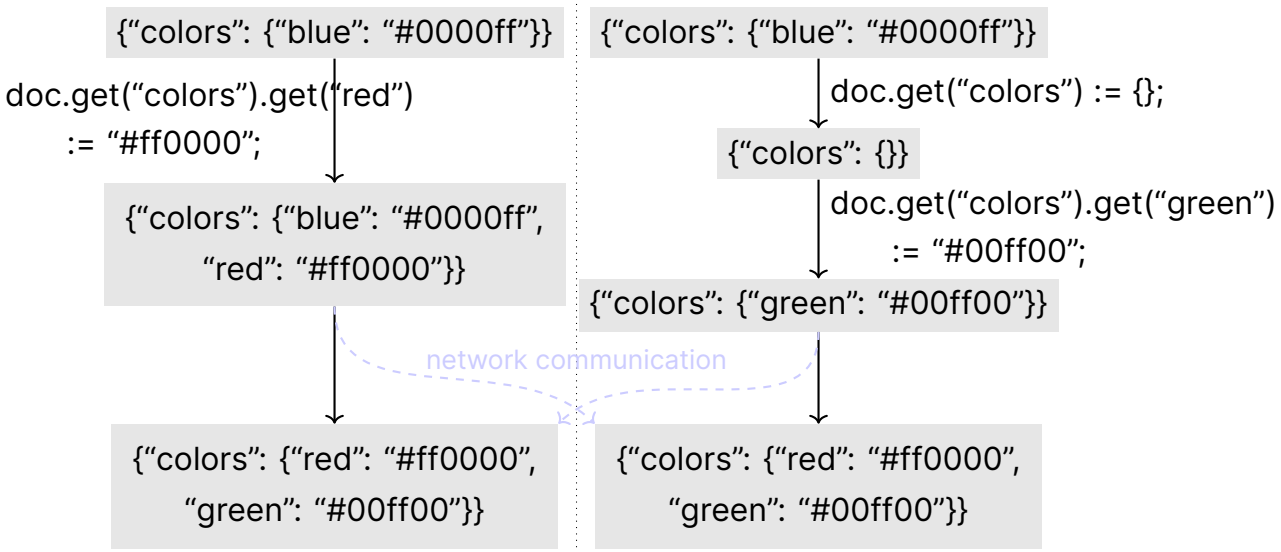


Figure 1.11: Example of Automerge algorithms

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 [9]. Automerge also comes with a ready-to-use JavaScript library. Figure 1.11 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.

1.1.4 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.

1.2 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.

1.2.1 Common patterns

As architectural patterns aim to solve various problems in different contexts, the term can be vaguely used to designate differing concepts.

The MVC pattern, illustrated in figure 1.12, tends to get paired with a monolithic architecture, illustrated in figure 1.13, 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.

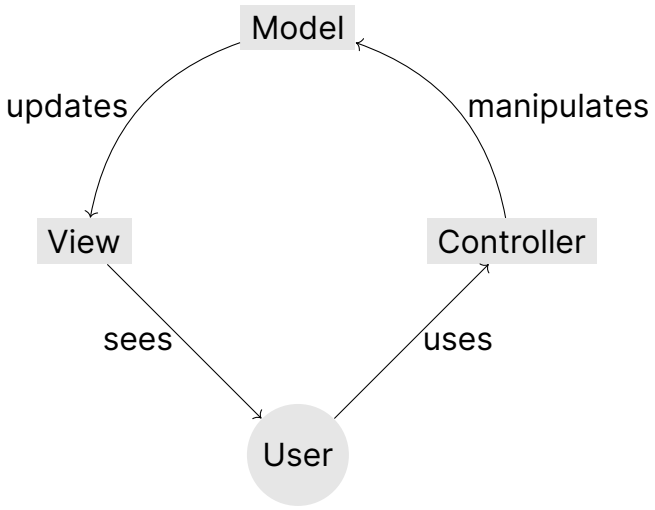


Figure 1.12: The MVC pattern

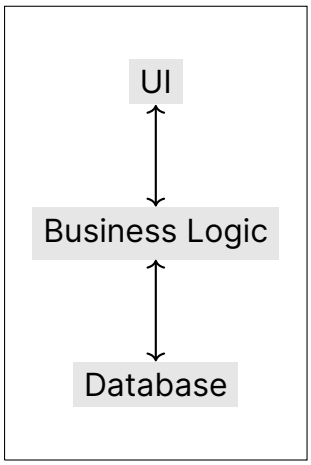


Figure 1.13: The monolith architecture

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

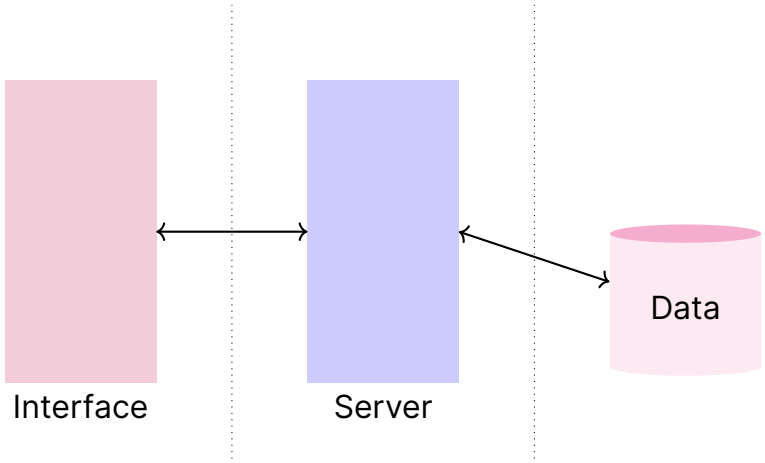


Figure 1.14: The three-tier architecture

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 1.14. 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 [10]. Therefore, each application has to find a sweet spot between scalability and flexibility, and maintainability.

MVVM pattern

Figure 1.15: 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 [14].

Most recent UI frameworks, such as React, Angular, Vue.js, and Svelte, implement some variation or another of MVVM [8].

1.2.2 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 1.16 illustrates our choice of architecture.

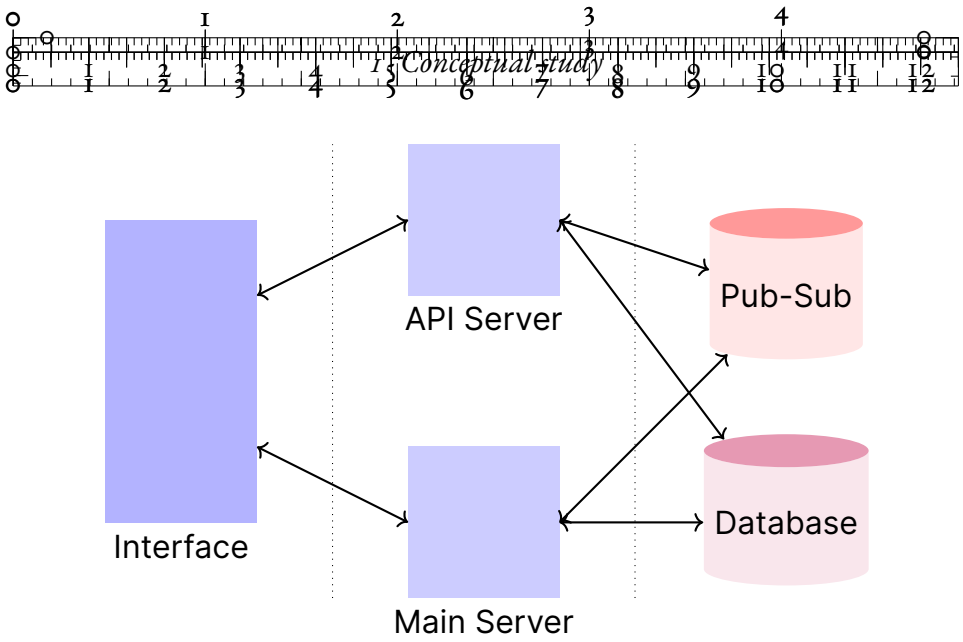


Figure 1.16: Our *n*-tier architecture

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.

1.3 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.

1.3.1 Class diagrams

Figure 1.17 represents our application's general class diagram.

1.3.2 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 1.18 represents the sequence diagram for "sign up".

Log in

Figure 1.19 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 1.20 represents the sequence diagram for "confirm email".

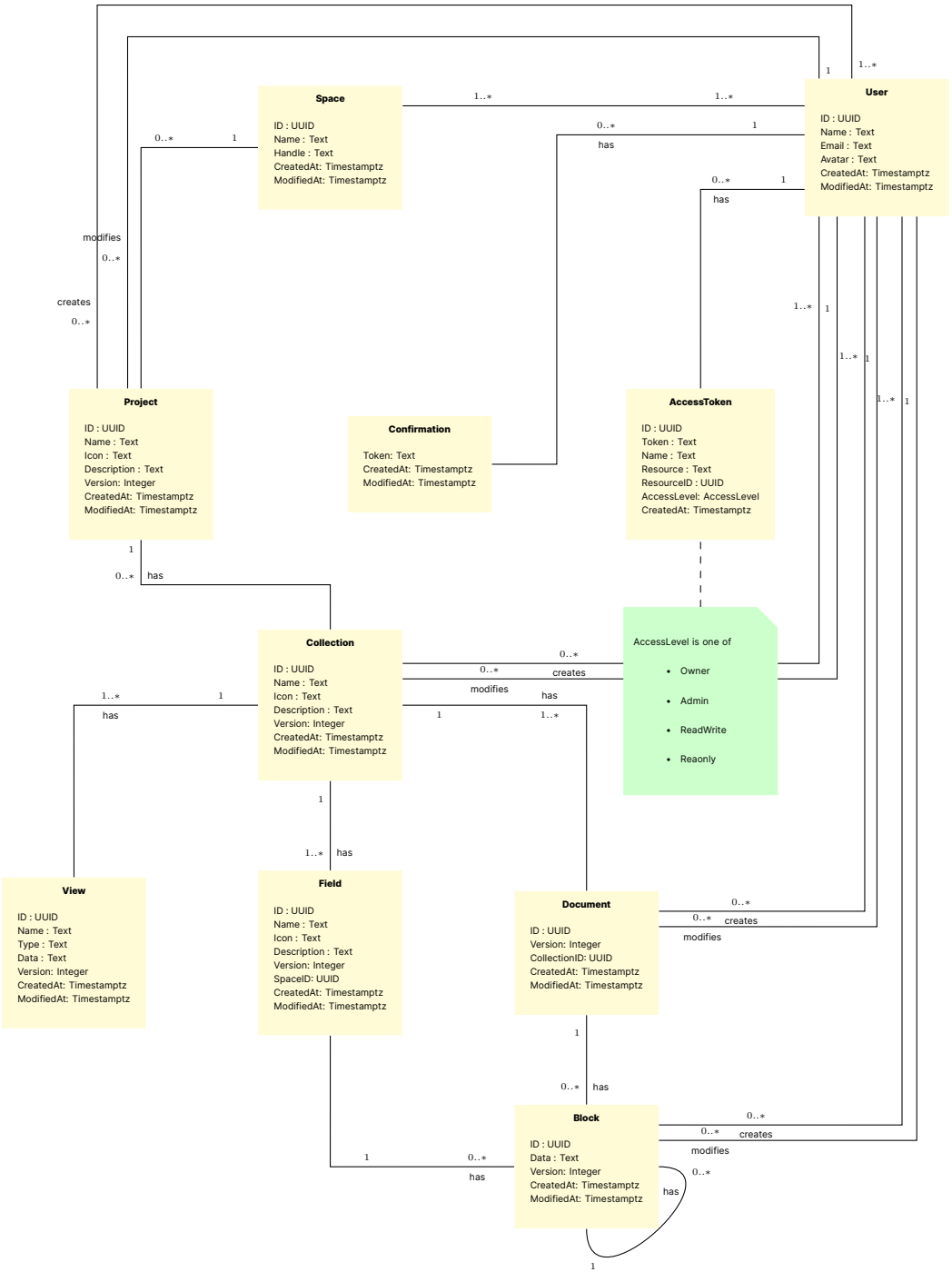


Figure 1.17: General class diagram

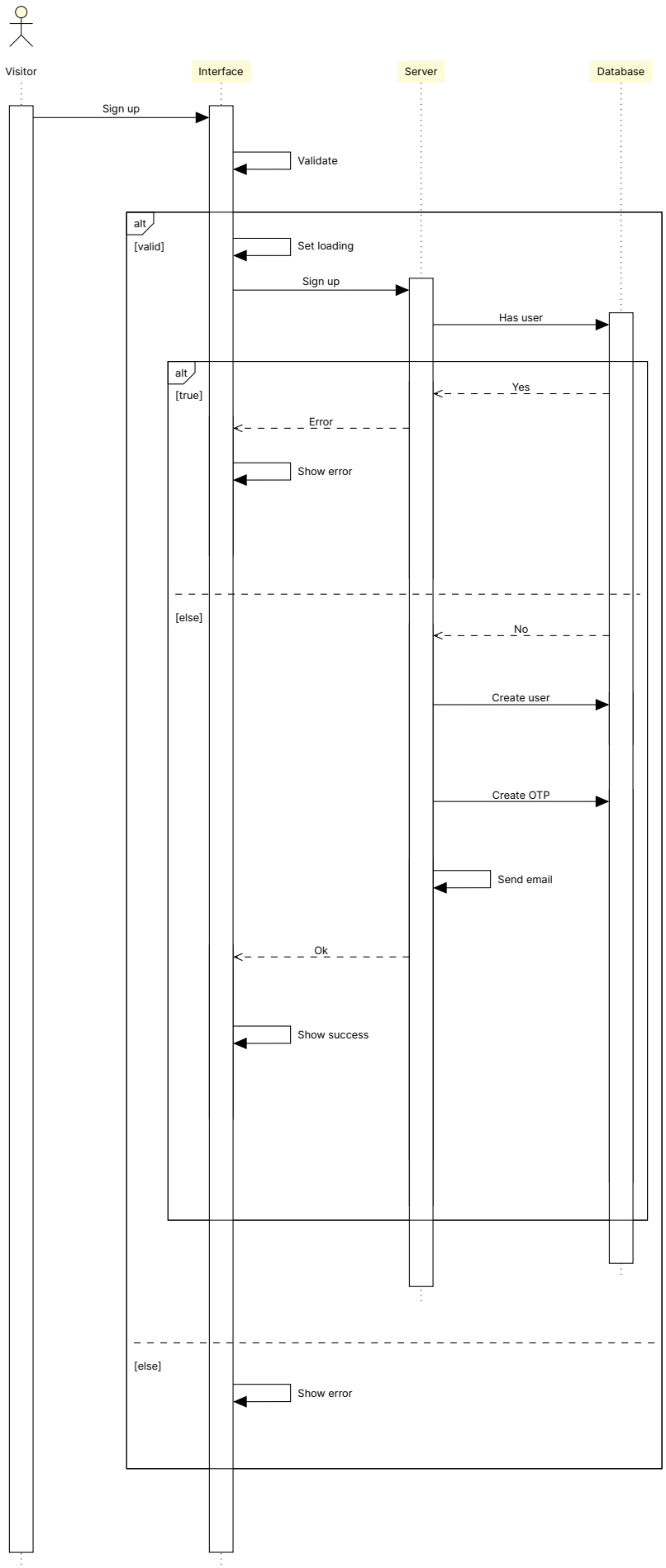


Figure 1.18: “Sign up” sequence diagram

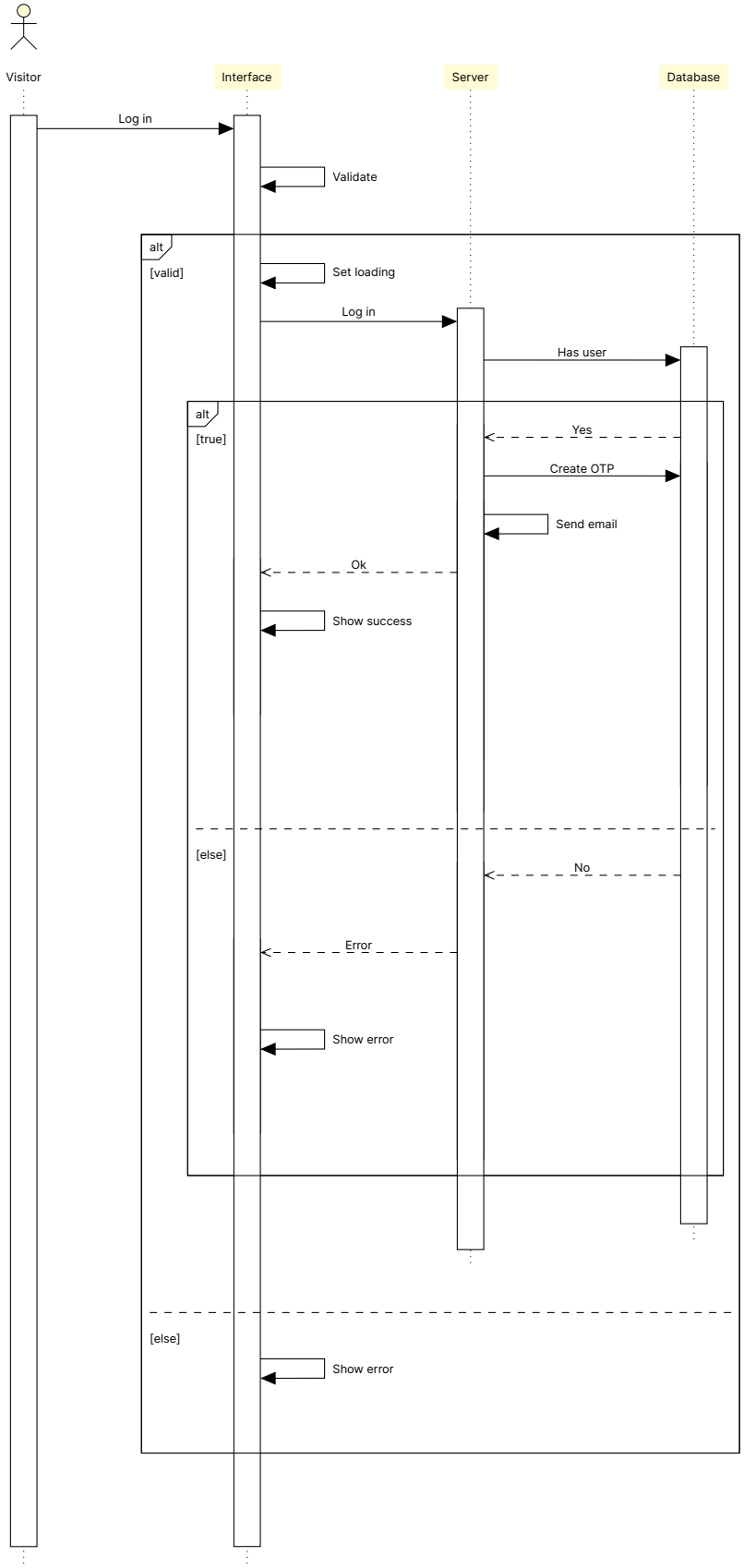


Figure 1.19: “Log in” sequence diagram

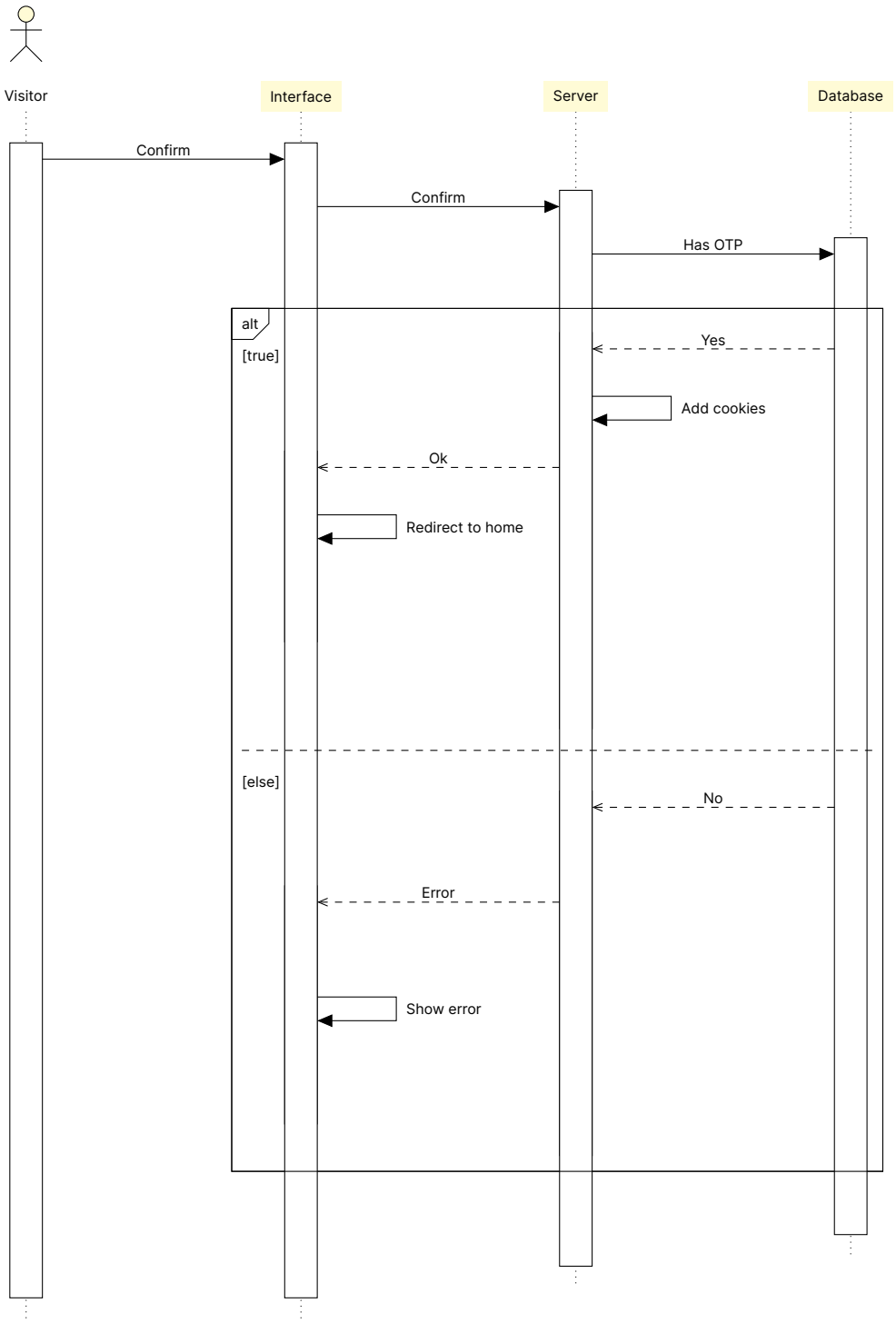


Figure 1.20: “Confirm email” sequence diagram

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 1.21 represents the sequence diagram for "create workspace".

Create collection

Figure 1.22 represents the sequence diagram for "create collection".

Create field

Figure 1.23 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.

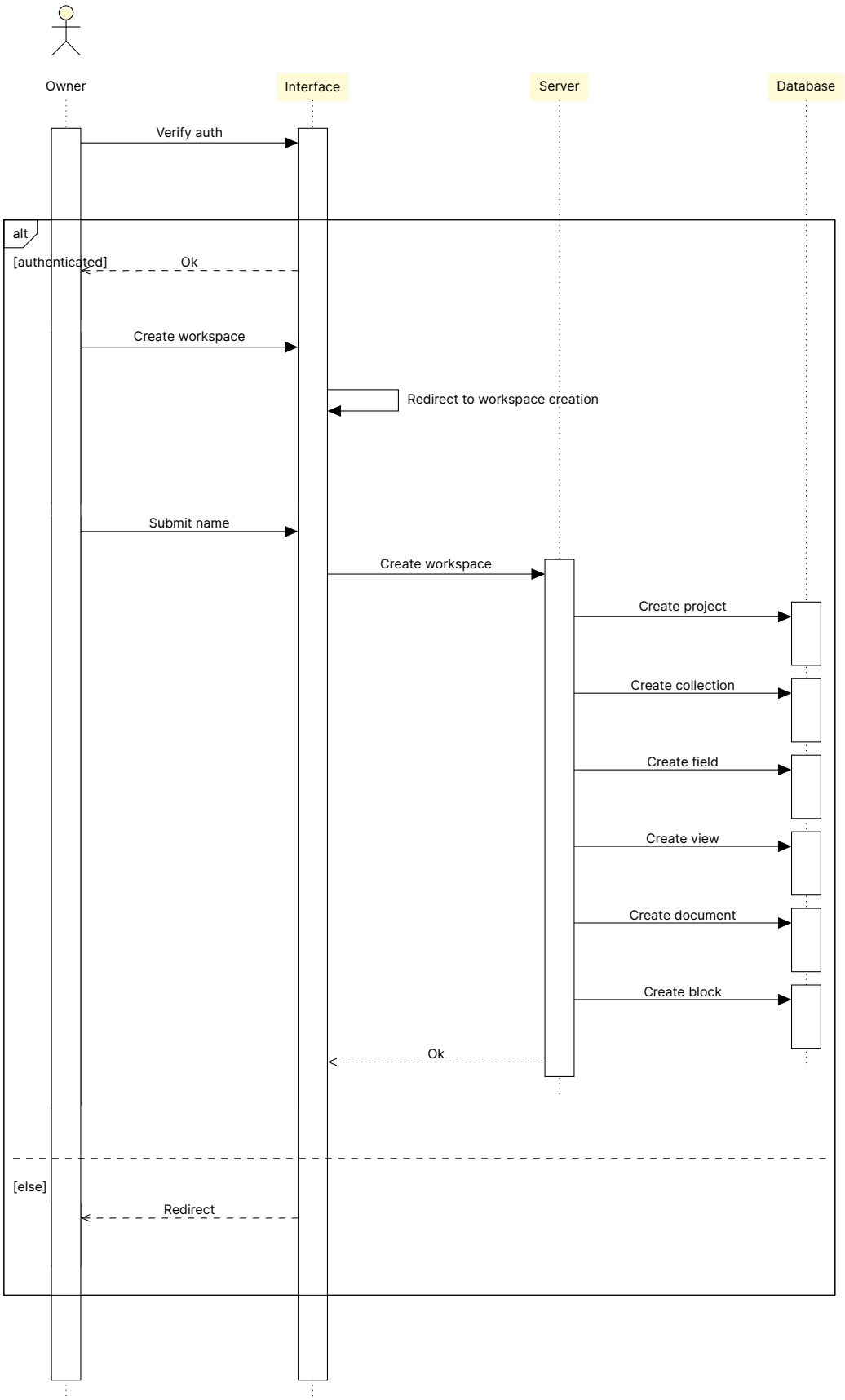


Figure 1.21: “Create workspace” sequence diagram

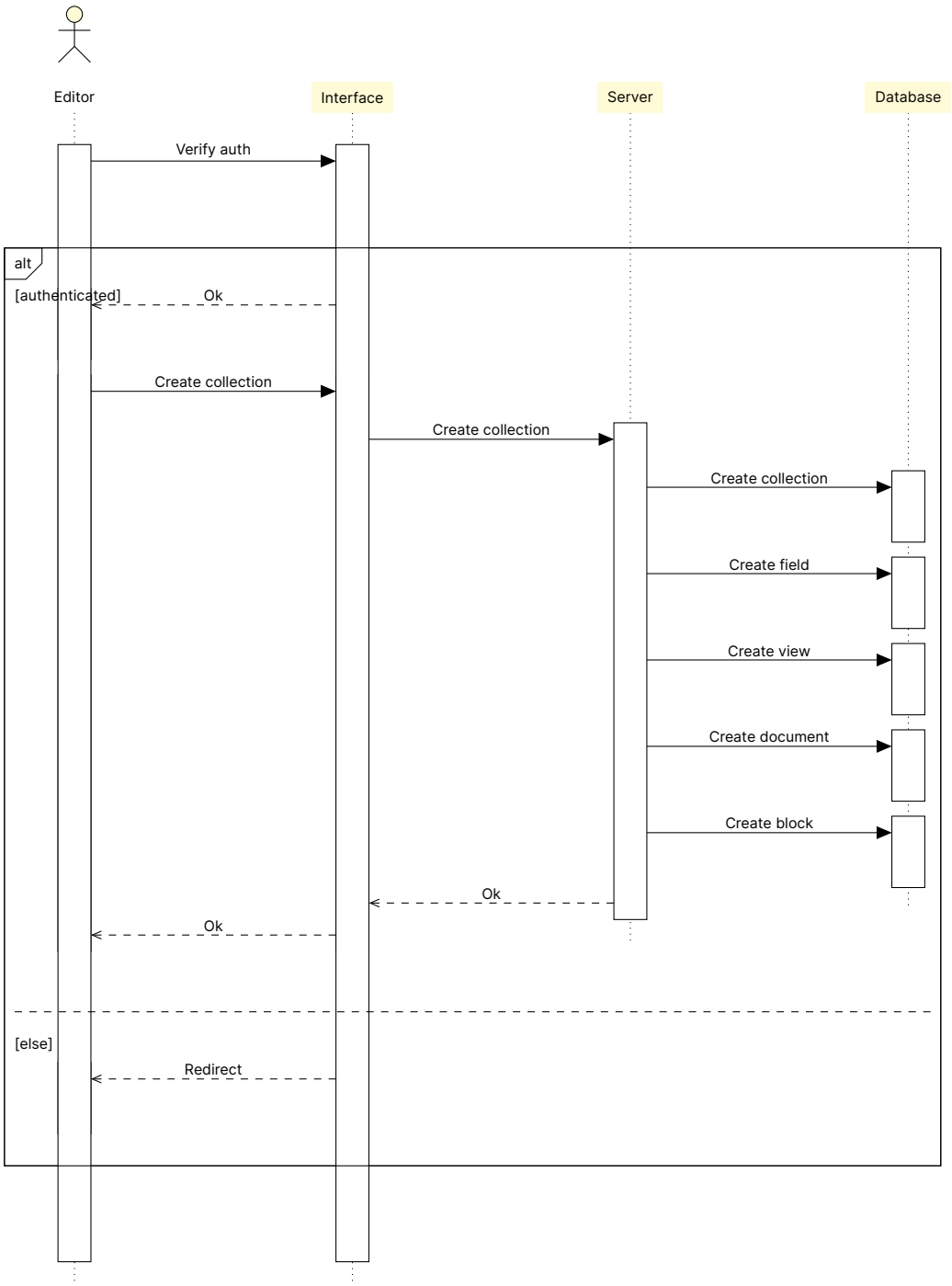


Figure 1.22: “Create collection” sequence diagram

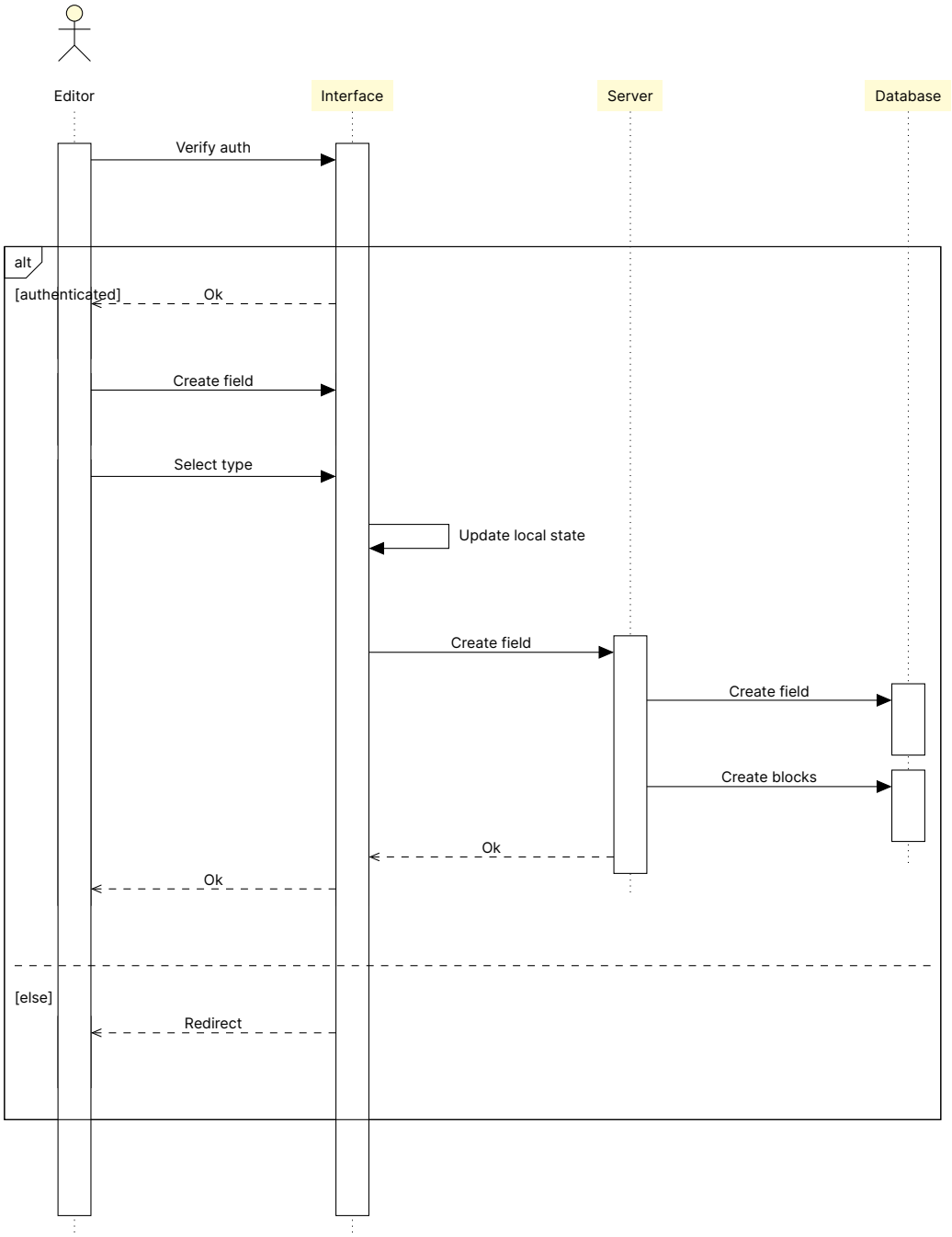


Figure 1.23: “Create field” sequence diagram

Figure 1.24 represents the sequence diagram for “update block”.

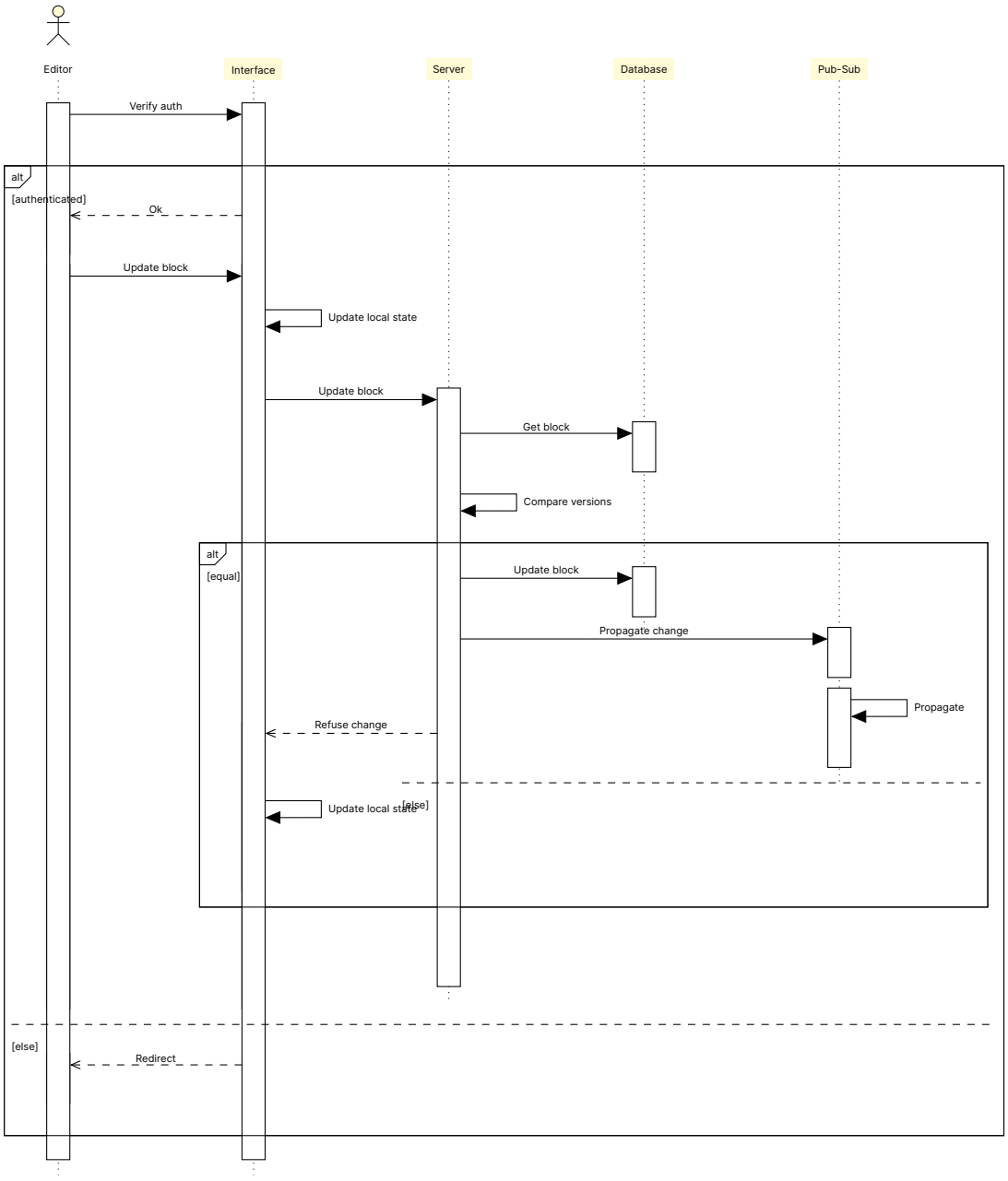
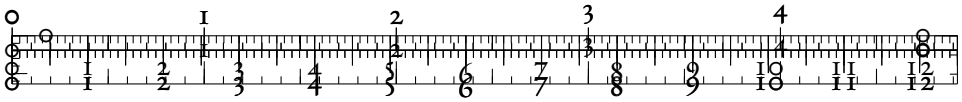


Figure 1.24: “Update block” sequence diagram

1.4 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.



Acronyms

CmRDT Commutative Replicated Data Type

CMS Content Management Software

CRDT Conflict-free Replicated Data Type

CvRDT Convergent Replicated Data Type

DOM Document Object Model

FDD Feature-Driven Development

GPU Graphics Processing Unit

JSON JavaScript Object Notation

MVC Model-View-Controller

MVVM Model-View-ViewModel

OT Operational Transformation

OTP One-Time Password

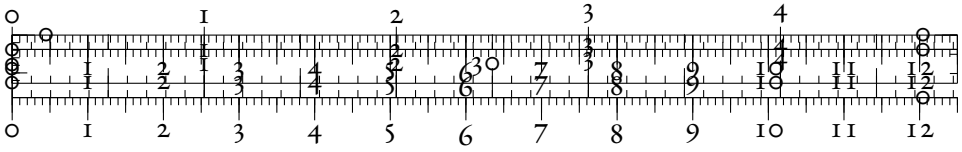
P2P Peer-To-Peer

RBAC Role-Based Access Control

RDMBS Relational Database Management System

SaaS Software as a Service

SDK Software Development Kit



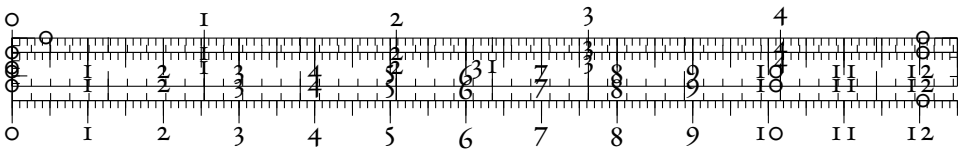
SSPL Server Side Public License

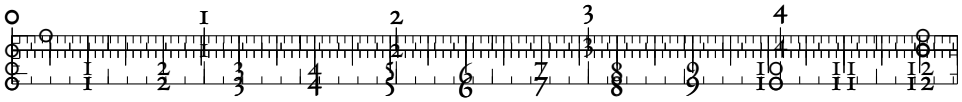
UI User Interface

UX User Experience

WAI-ARIA Web Accessibility Initiative – Accessible Rich Internet Applications

WOOT WithOut Operational Transformation

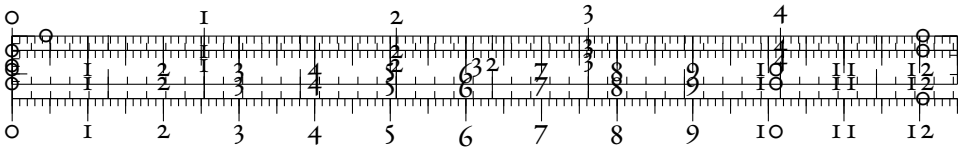


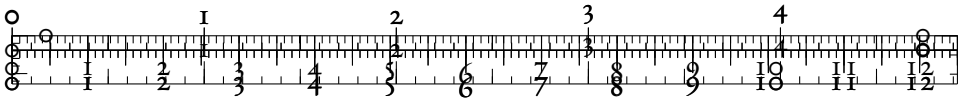


Glossary

cacheable A cacheable response is an HTTP response that is stored to be retrieved and used later, saving a new request to the server [2].

polyfill A piece of code used to provide modern functionality on older browsers that do not natively support it [11].





Bibliography

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

[2] *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).

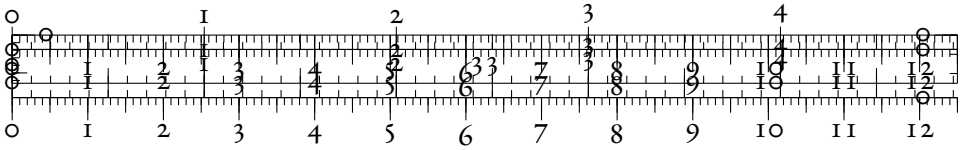
[3] E. Chang. *eHub Interviews Writely*. English. Oct. 2005. URL: https://web.archive.org/web/20110722190058/http://emilychang.com/ehub/app/ehub-interviews-writely/ (visited on 06/04/2021).

[4] *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).

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

[6] 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).

[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] *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).

[9] 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.

[10] 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).

[11] *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).

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

[13] 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.

[14] 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).

[15] 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).

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

[17] 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).

[18] *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).