Warsaw University of Technology

Faculty of Mathematics
and Information Science

Bachelor's diploma thesis

in the field of study Computer Science and Information Systems

Design and Implementation of a Collaborative Board Using the Local-First Approach

Piotr Jankiewicz

student record book number 288767

thesis supervisor

dr inż. Paweł Kotowski

WARSAW 2026

Abstract

Design and Implementation of a Collaborative Board Using the Local-First Approach

The purpose of this thesis is to design and implement a collaborative board using the local-first approach. The solution uses distribuited conflict free replicated data types (CRDTs) and real-time communication via WebRTC protocol to bring effortless collaboration with other users.

Final result of the effort done is an desktop application that enables drawing and erasing on the whiteboard with mutiple users simultaneously. This paper covers description of solution, but also thouches broader topic of local-first application design architectures.

Key words: rust language, local-first, CRDT, WebRTC, distributed system, star topology, multithread real time communication

Streszczenie

Projekt i wdrożenie kolaboratywnej tablicy z wykorzystaniem podejścia „local-first"

Celem pracy inynierskiej jest zaprojektowanie i implementacja kolaboratywnej tablicy z wykorzystaniem podejścia "local-first". Wypracowane rozwiązanie uzywa rozproszonego bezkonfliktowego typy danych (CRDTs) oraz komunikacji w czasie rzeczywistym przez protokół WebrRTC by umozliwic bezproblemową kolaborację z innymi uzytkownikami. Ostatecznym wynikiem jest aplikacja desktop-owa umozliwiająca rysowanie oraz zmazywanie poprzednich pociągnięć na białej tablicy z wieloma uzytkownikami jednocześnie. Praca pokrywa opis rozwiązania ale równiez porusza bardziej obszerny temat architektur typu local-first w aplikacjach kolaboratywnych.
Słowa kluczowe: język rust, local-first, CRDT, WebRTC, system rozproszony, topologia gwiazdy, wielowątkowa komunikacja w czasie rzeczywistym

Contents

1. Introduction

1.1. Analysis and description of collaborative local-first applications

Collaborative applications have not always been so popular as they are these days. You might not even know that some of applications that we use every day are collaborative. In everyday life of a software developer almost all of modern IDEs have collaborative work options, almost all project management software are also collaborative. These are examples close to software engineers, but these applications do not limit themselves to software development. Google Docs, Microsoft 365, Figma, Miro and many more are examples of collaborative applications that are used by millions of people every day. These applications are not only used for work, but also for education and entertainment.

It hasn't always been like this. The steep rise of collaborative applications began as the remote work became more popular but also it owes its rise of popularity to universality of fast global network and increase of computing power of everyday devices.

Current collaborative apps couldn't exist 20 years ago. They haven't been simply invented yet. Typical application of early days of online applications had a simple architecture. It was a client-server application. The server was responsible for storing data and processing it, while the client was responsible for presenting the data to the user and sending user input to the server. This architecture had its advantages and disadvantages. It was simple and easy to implement, but it had a single point of failure. If the server went down, the entire application went down. It also had scalability issues, as the server had to handle all the requests from the clients.

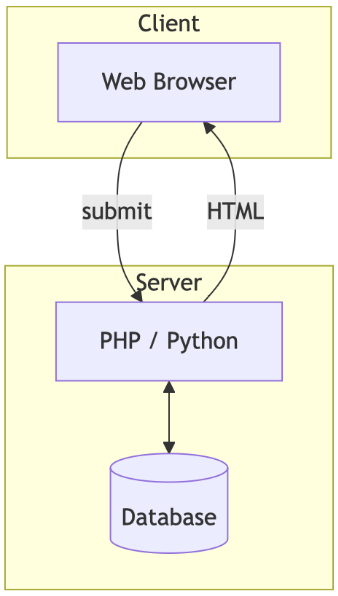

Figure 1.1: Simple client-server architecture of early internet era applications

These kind of application belong to "cloud-first" family. There was no need for "local-first" architectures yet and this term has not been popularized. Before even the discussion on topic of "local-first architectures" the "cloud-first" approach had its own more than one decade of fast development. From simple client server architectures the system rose to more difficult forms of large size client-side application with lots of layers and microservices. Picture below shows architectures that emerged on the market with rise of popularity of mobile and web apps from 2010.

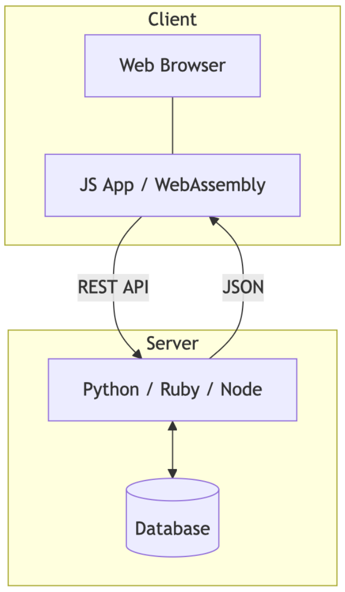

Figure 1.2: More complicated architecture of modern mobile/web applications

How far are these architectures from beeing good fit for collaboarative appliactions? Lets have a look on typical modules that take part in standrd user interface interaction. Picure below shows typical modules of morden web/mobile/desktop cloud-application.

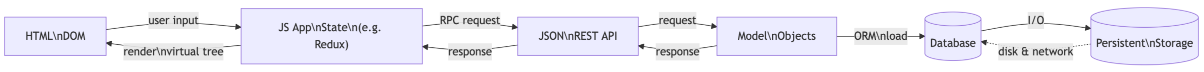

Figure 1.3: Typical modules of modern cloud-based applications

As we can see the level of complication if very high. Simple user intercation with a system triggers multiple functions from multiple modules from multiulpe services. This level of sophistication simply cannot be used for collaborative applications, because orchestration of synchoronization of state shown in simultaneus users would be extremely difficult.

For one, in would require to many asynchronous time-consuming remote calls to different services. Total synchronization nightmare. For some time tough there has been efforts to implement collaboration this way. This approach was known as OT - operational transformation (still used in Google Docs). Although Google Docs is a good example of usage of cloud-first approach, in this thesis I decided to focus on local-first architecutres as they have proved to be more broadly used in the industry and it offers way more possibilities of different kind of applications.

There are also my other reasons to avoid cloud-first architectures when designing collaborative applications. Lets take a look at simple sequance of user interaction with the system. Below we can see general asynchronous interaction of a user with normal cloud-basen application.

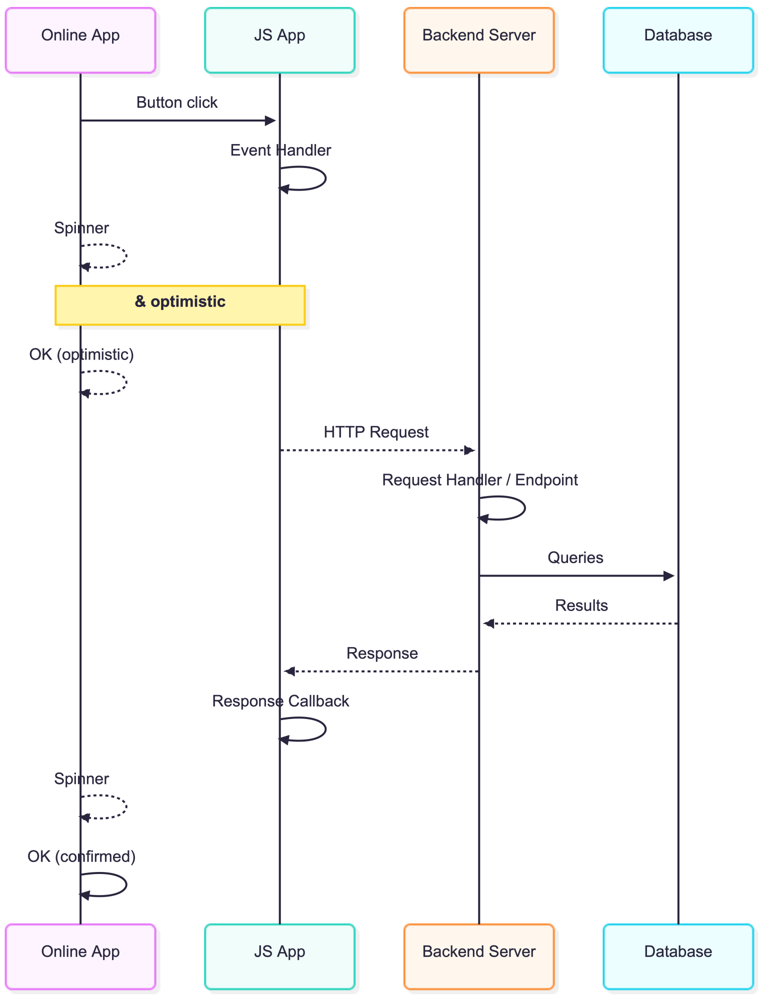

Figure 1.4: Typical sequance of interaction with a user in a modern cloud-based applications

As we can see on on this diagram, the system relies heavily on connectivity with the database. The big question that comes comes to mind is - what happens to the system if there is no internet ? Answer is simple. It would not work. Therefore, one could even say, that whole application is just a asynchronous graphical database wrapper, and it wouldn't be far from truth. This brings us the quastion - What is the definition of cloud-first and local first applications?

In simplest words the definition of cloud-approach is "If the interaction of the user with the system haven't been saved to database, it never happened". On other hand local-first is "What's important the most is local persistance - cloud brings only connectivity".

Therefore let's have a look on how would typical sequance of user interaction with local-first system look. The diagram below shows typical sequance in such system.

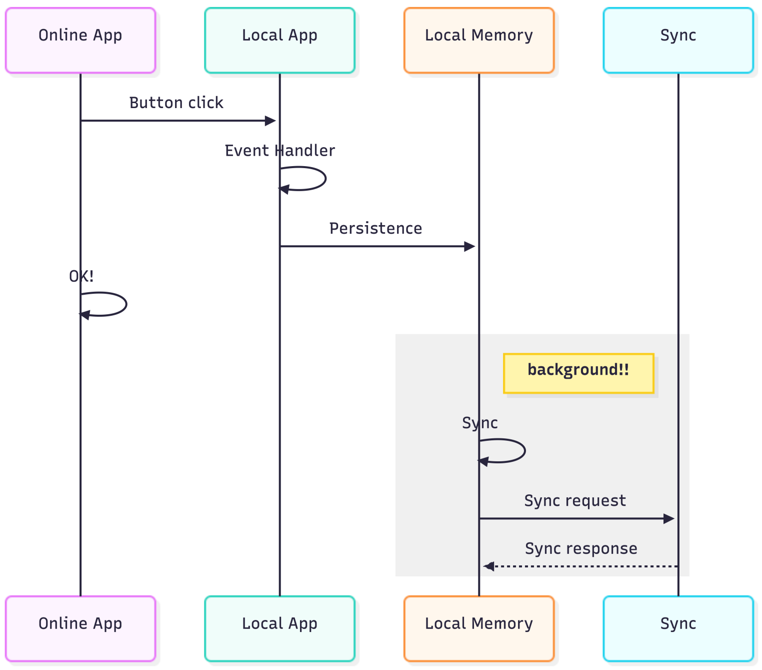

Figure 1.5: Typical sequance of interaction with a user in a local-first applications

The distinction is clear. In local-first approach the user interaction with the system is not dependent on connectivity with the database. Connectivity brings only messaging with other peers. Whole synchronization of replicas of data between users happens on local machine. This is the reaseon why we can call this system distributed. We can also notice that this approach requires much less modules across different serivces. Later, in the thesis, we will clearly how this concept allows us to bring true real time collaboration to the users.

1.2. Local-first vs Cloud-first

Lets have a look on the main differences between local-first and cloud-first approaches. The table below shows the main differences between these two approaches when it comes to designing collaborative application.

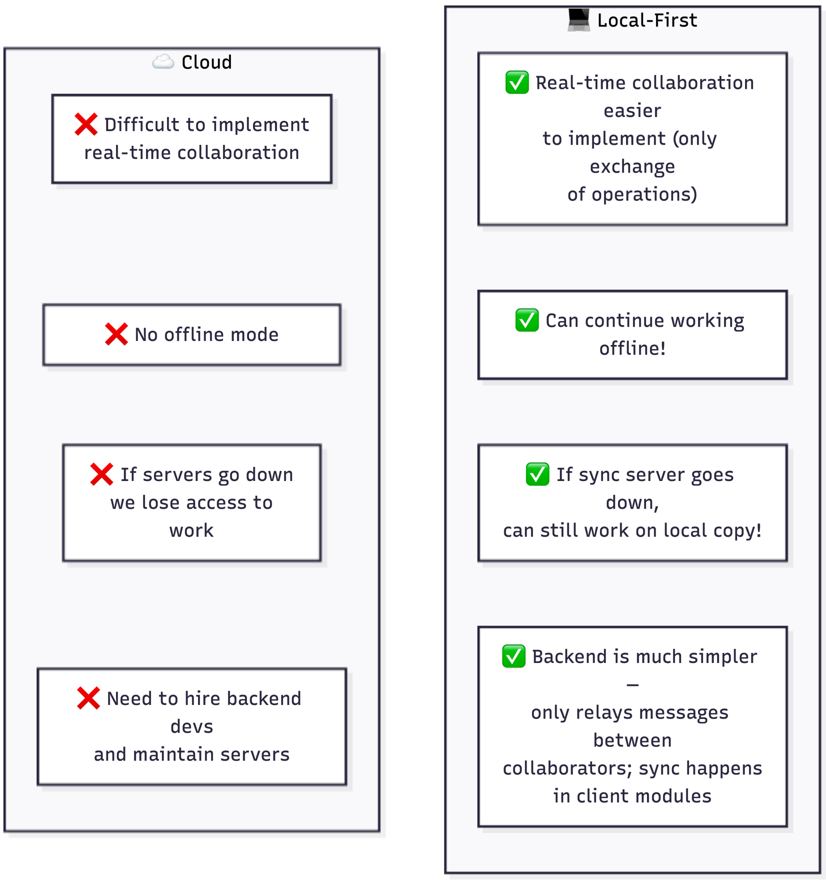
Cloud
❌ Difficult to implement real-time collaboration
❌ No offline mode
❌ If servers go down we lose access to work
❌ Need to hire backend devs and maintain servers
Local-First
✅ Real-time collaboration easier to implement (only exchange of operations)
✅ Can continue working offline!
✅ If sync server goes down, can still work on local copy!
✅ Backend is much simpler – only relays messages between collaborators; sync happens in client modules

Figure 1.6: Cloud vs Local comparison

Clearly, the obvious choice for apprach when designing collaborative application would be Local-first. Let's now try to distinguish, what are boundaries of such system - what use cases are not applicable with local-first apprach. The table below shows use cases that are not good fit for local-first approach.

Table 1.1: Use cases: Suitable and Unsuitable for Local-First Approach

Suitable for	**Unsuitable for**
Real-time collaborative editing (documents, whiteboards)	Applications requiring strong centralized control
Offline-first mobile/desktop apps	Systems with strict regulatory or audit requirements
Peer-to-peer file sharing	Applications with massive global data aggregation
Personal productivity tools with sync	Centralized transactional systems (e.g., banking)
Small/medium team collaboration	Apps needing immediate global consistency

1.3. Why would one implement own collaborative system ?

Creating collaborative whiterboard is just an intriduction to the general topic of distrubuted conflict free real time data storage with editable operations. Designign whiteboard is easy exaple for system that can be used to many use cases (distribited conflict free data) - it serves as a great introduction to this topic.

There are multiple use cases/possible usage of distrubuted systems. For example drone swarms - distributed replicas of formation among drones broadcasted between each other. Peer-to-peer file sharing - distributed replicas of files among users. Real time collaborative editing - distributed replicas of documents among users.

1.4. Examples from industry

Known examples of local-first applications are: Figma, Miro, Notion, Obsidian, Roam Research, Coda, Google Docs (although it is not pure local-first application). These applications are used by millions of people every day. They are used for work, education and entertainment.

1.5. Known architectures for collaborative applications

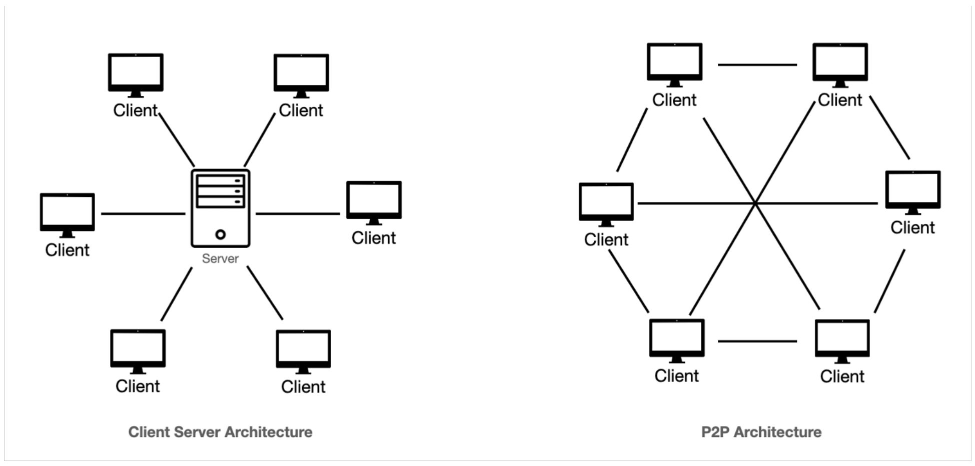

Figure 1.7: Comparison of Client-Server and Peer-to-Peer (P2P) Architectures

Connectivity among multiple users is always solved eigher by peer-to-peer or more "centered" relay architecure for example start-topology. When designig a collaborative system each solution has its benefits and setbacks. In a table belowe there is simple analysis of pros and cons of choosing one or the ohter when designing local-first collaborative whiteboard.

Table 1.2: Comparison of Peer-to-Peer and Star Topology Architectures

Peer-to-Peer (P2P)	**Star Topology**
No need to create a server for communication. Communication is direct - good. Clients manage their connections themselves - might be problematic.	Central node relays communication. It requires support of server. Keeping servers running is expansive.
Communication might be faster because it is direct. Users might be close to each other.	Clients sometimes might unnesesarry communicate via very distant server to both of them, which would cause delays.
High resilience to single node failure	Central node is a single point of failure
Complex synchronization and conflict resolution - corrdination of connections has to be implemented on client side.	Easier management and coordination - server keeps all informations on connectivity. Clients connect only to one server.
Scalability can be challenging - many users might have very low maximum of connections it could sustain.	Scalability depends on central node capacity, but server can easily be scaled hirizontlay or verticaly
Suitable for decentralized systems with limited numer of peers at the same time (for example 10)	Suitable for medium (up to 50) teams with central coordination - for example video conferance rooms

As we can see - each solution suits different applications.

1.6. OT vs CRDTs

One very important topic in the matter of conflict resolution and collaboration is data structures algorightms that can be used. Local / remote inserts into data structures need to be **commutative**, **associative** and **idempotent**. These three properties are required to ensure that all replicas of data will be eventually consistent - assuming that all operations (inserts/deletes) propagate properly. There are two main approaches to achieve this - OT (operational transformation) and CRDTs (conflict-free replicated data types).

Historically well known solution is OT - operational transformations. The basic idea behind this approach is to transform operations in such a way that they can be applied in any order and still produce the same result. This approach was used in Google Docs and it is still used in some applications. However, it has some drawbacks. It is complex to implement and it can lead to conflicts that are difficult to resolve.

CRDTs - conflict-free replicated data types - are a more recent approach to achieve eventual consistency in distributed systems. The basic idea behind CRDTs is to design data structures in such a way that they can be updated independently and still converge to the same state. The algorithms are implemented directly into data structures. This approach is simpler to implement and it can handle conflicts more gracefully. It is used in many modern applications, including Figma, Miro, Notion, Obsidian, Roam Research, Coda and many more.

1.7. Vision of the System

The envisioned system is a desktop application designed for real-time collaborative drawing on a whiteboard. It leverages the Rust programming language for performance and reliability, and utilizes the egui library to provide a modern, responsive user interface. Communication between users is achieved through WebRTC, with the Livekit open-source server facilitating room-based connections. Each room acts as a channel or document, allowing users to join by simply entering a room code or name, without the need for user accounts, authentication, or authorization.

The application employs CRDTs (Conflict-Free Replicated Data Types) from the Automerge library to ensure seamless synchronization of whiteboard state across all participants. This approach enables users to draw and erase simultaneously, with changes instantly reflected for everyone in the room. The cursors of other users are visible, enhancing the collaborative experience. The whiteboard will be represented as a canvas with raster graphics of strokes from users as vector of points with certain width and color.

For real-time, multithreaded communication, the system uses the Tokio library, ensuring efficient handling of concurrent operations. Notably, the application does not persist data by default; once the app is closed, all data is lost unless explicitly exported to a file by the user. The focus is on desktop environments, with no mobile version planned.

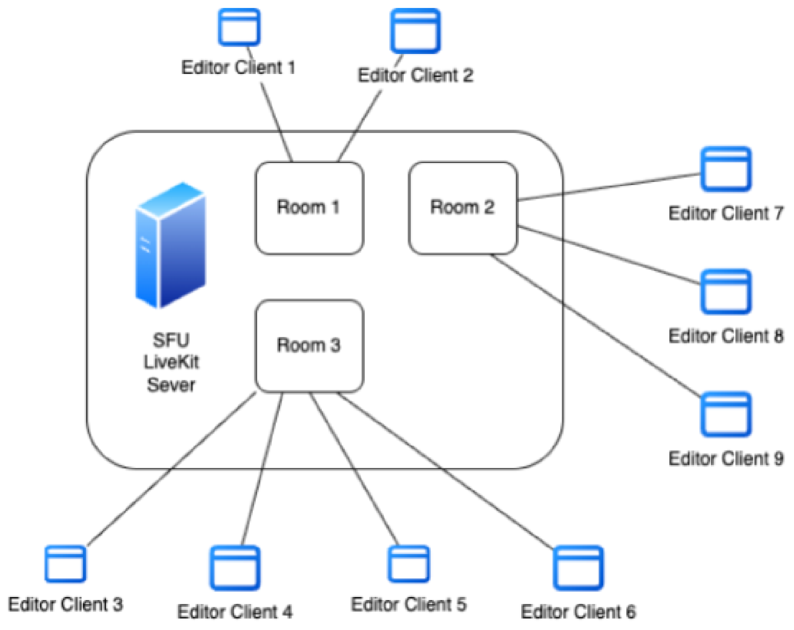

Figure 1.8: Architecture of the envisioned system

1.8. Requirements Specification

1.8.1. Functional Requirements

Below you can see tables with user stories for each actor in the system: user, application, and server.

Table 1.3: Functional Requirements – User Stories (Actor: User) Part 1

ID	**As a...**	**I want to...**	**In order to...**
US-01	User	launch the desktop app and see a blank whiteboard	start drawing immediately without any setup
US-02	User	draw freehand strokes on the white-board using a pen tool	express my ideas visually
US-03	User	erase previously drawn strokes	correct mistakes or modify drawings

Table 1.4: Functional Requirements – User Stories (Actor: User) Part 2

ID	As a...	I want to...	In order to...
US-04	User	change the color of the pen	differentiate between elements and add visual clarity
US-05	User	change the thickness (width) of the pen	draw both fine details and bold strokes
US-06	User	clear the entire whiteboard	start drawing from an empty board again
US-07	User	save the current whiteboard state to a file (.crdt/.png)	persist my work for later use
US-08	User	load a previously saved whiteboard document (.crdt/.png)	continue working on a previous drawing
US-09	User	create a new empty document	begin a fresh collaborative session or personal workspace
US-10	User	enter a room name and connect to a collaborative session	work together with other users in real time
US-11	User	disconnect from a collaborative session at any time	leave the room without closing the application
US-12	User	see the cursors of other connected users on the whiteboard	know where collaborators are drawing and coordinate visually
US-13	User	see strokes drawn by other users appear in real time	have a seamless collaborative experience
US-14	User	send chat messages to other participants in the session	communicate textually alongside visual collaboration
US-15	User	be notified when another participant joins or leaves the room	stay aware of who is currently collaborating
US-16	User	join a session without creating an account (only room code needed)	minimize friction and start collaborating instantly

Table 1.5: Functional Requirements – User Stories (Actor: Application, Part 1)

ID	**As a...**	**I want to...**	**In order to...**
AP-01	Application	maintain a local CRDT document (Automerge) representing the whiteboard state	enable offline-capable and conflict-free editing
AP-02	Application	render all strokes stored in the CRDT as raster graphics on a canvas	display the current whiteboard state to the user
AP-03	Application	generate CRDT sync messages when local changes occur	propagate the user's drawing operations to all connected peers
AP-04	Application	receive and apply incoming CRDT sync messages from remote peers	update the local whiteboard with changes made by others
AP-05	Application	establish a WebRTC data channel connection through the LiveKit server	enable real-time, low-latency peer-to-peer communication
AP-06	Application	broadcast the local user's cursor position to all connected peers	allow other users to see where the local user is pointing
AP-07	Application	receive and render remote cursor positions on the whiteboard	display collaborators' cursors with distinguishing color and name
AP-08	Application	run network I/O and CRDT synchronization on separate threads (Tokio)	keep the UI responsive and non-blocking during communication
AP-09	Application	serialize the CRDT document to a byte vector for file export	allow the user to save the full document state
AP-10	Application	deserialize a byte vector from a file into a CRDT document	restore a previously saved whiteboard with full edit history
AP-11	Application	read LiveKit connection credentials from an .env file or environment variables	configure server connectivity without hardcoding secrets

Table 1.6: Functional Requirements – User Stories (Actor: Application, Part 2)

ID	**As a...**	**I want to...**	**In order to...**
AP-12	Application	resolve conflicts between concurrent edits automatically via CRDT merge	guarantee eventual consistency without manual conflict resolution
AP-13	Application	represent each stroke as a vector of points with associated width and color in the CRDT	faithfully reproduce strokes across all replicas
AP-14	Application	provide a modern responsive GUI via the egui library	deliver a native desktop experience on macOS and Windows

Table 1.7: Functional Requirements – User Stories (Actor: Server)

ID	As a...	I want to...	In order to...
SV-01	Server	manage rooms (channels) that users can join by name	group collaborating users into logical sessions
SV-02	Server	relay WebRTC data channel messages between all participants in a room (star topology)	ensure every client receives every peer's sync messages
SV-03	Server	authenticate incoming client connections using API key/secret token validation	prevent unauthorized access to collaboration rooms
SV-04	Server	notify all participants when a new user joins a room	allow clients to initiate CRDT sync with the newcomer
SV-05	Server	notify all participants when a user disconnects from a room	allow clients to remove the disconnected user's cursor and update UI
SV-06	Server	automatically create a room when the first participant connects to a new room name	eliminate the need for manual room provisioning
SV-07	Server	support multiple concurrent rooms with independent participant lists	allow several collaborative sessions to run simultaneously
SV-08	Server	forward data channel packets with minimal latency	support the real-time requirements of collaborative whiteboard synchronization

1.8.2. Non-Functional Requirements

1.9. Business Cases

2. Example chapter

This TEX file is to be compiled with pdfLaTeX (it's just quick build in TeXMaker).

2.1. Example section

Definition 2.1 (Definition). A *definition* is a statement of the meaning of a term (a word, phrase, or other set of symbols).

2.1.1. Example subsection

It's the deepest deph of sectioning allowed by rector.

Definition 2.2 (Equation). In mathematics, an *equation* is a statement of an equality containing one or more variables.

Example 2.3. This is an example of an equation:

$$2 + 2 = 4. \tag{2.1}$$

Equation without a number:

$$2 + 2 = 4,$$

or:

$$2 + 2 = 4.$$

It is worthwhile to peruse other mathematical environments like *multline*, *align* and their versions with a star (, i.e. without numeration). The description of their use can be found at `https://texdoc.org/serve/amsldoc.pdf/0` starting from the end of the third page.

Equation (2.2) is false. References (and some other things) work properly after compliling TEX file twice.

$$\int\limits_0^1 x\,dx = \frac{3}{2}. \tag{2.2}$$

Theorem 2.4 is a very interensting result.

Theorem 2.4 (Pythagoras' Theorem). Let c represent the length of the hypotenuse and a and b the lengths of the triangle's other two sides. Then:

$$a^2 + b^2 = c^2.$$

Proof. The proof has been presented in [1] and [2]. We can write then [1, 2]. □

Corollary 2.5. The use of the term *corollary*, rather than *proposition* or *theorem*, is intrinsically subjective.

Remark 2.6. You can find a rather comprehensive list of available symbols at `https://www3.nd.edu/~nmark/UsefulFacts/LaTeX_symbols.pdf`.

If you want to find a symbol by its shape, you can use the following site: `https://detexify.kirelabs.org/classify.html`.

Lemma 2.7 (Someone's Lemma). Ten lemat jest nie na temat.

Proof. Dowód przez indukcję. □

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

2.2. Floats – tables and figures

Place labels after captions or you get the wrong labelling.

In Table 2.1 there are additional options for `table` and `figure` environments.

Lorem ipsum dolor sit amet, consetetur sadipscing elit, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam non-

Table 2.1: Additional options

symbol	effect
`h`	Place the float here, i.e., approximately at the same point it occurs in the source text (however, not exactly at the spot)
`t`	Position at the top of the page
`b`	Position at the bottom of the page
`p`	Put on a special page for floats only
`!`	Override internal parameters LaTeX uses for determining "good" float positions
`H`	Places the float at precisely the location in the LaTeX code. Requires the float package,[1] i.e., `\usepackage{float}`. This is somewhat equivalent to !ht.

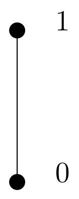

Figure 2.1: Example figure – it has been drawn by LaTeX default tools

umyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

3. The next chapter

Lorem ipsum dolor sit amet, consetetur sadipscing elit, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumyeirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diamvoluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

3.1. Matrices

Simple matrix:

$$\begin{matrix} a & b & c & d \\ d & e & f & g \\ 1 & 1 & 1 & 1 \end{matrix}$$

Matrix with parentheses:

$$A = \begin{pmatrix} a & b & c & d \\ d & e & f & g \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Matrix with brackets:

$$\begin{bmatrix} a & b & c & d \\ d & e & f & g \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

You can also use more general environment:

$$\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}$$

Matrix with braces:

$$\begin{Bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix}$$

Definition 3.1. Let $A \neq \emptyset$, $n \in \mathbb{N}$. Every function $f\colon A^n \to A$ is called an *n-ary operation* or *działaniem* określonym na A. 0-ary operations are constant functions.

Definition 3.2 (Algebra). The ordered pair (A, F), where $A \neq \emptyset$ is a set and F is a family of operations defined on A, shall be called an *algebra* (or *F-algebra*). The set A is called *the set of elements*, *support* or *universe* of an algebra (A, F) and F is called *the set of elementary operations.*

Proposition 3.3. I state that, having passed to the limit, the only thing left me me is to camp at said limit or return, or, maybe, search for a pass or an exit to other areas.

Bibliography

[1] A. Author, *Title of a book*, Publisher, year, page–page.

[2] J. Bobkowski, S. Dobkowski, Title of an article, *Magazine X, No. 7*, year, PAGE–PAGE.

[3] C. Brink, Power structures, *Algebra Universalis 30(2)*, 1993, 177–216.

[4] F. Burris, H. P. Sankappanavar, *A Course of Universal Algebra*, Springer-Verlag, New York, 1981.

List of symbols and abbreviations

nzw.	nadzwyczajny
*	star operator
~	tilde

If you don't need it, delete it.

List of Figures

If you don't need it, delete it.

Spis tabel

If you don't need it, delete it.

List of appendices

1. Appendix 1
2. Appendix 2
3. In case of no appendices, delete this part.