Faculty of Engineering

Ain Shams University

Project Report

Distributed Computing

CSE 432 – Spring 2022

Computer Engineering & Software Systems Program

Supervised by

Prof. Ayman Bahaa

Eng. Mostafa Ashraf

Team members

Ali Sayed Attia Elganzory	18P2073
Mohamed Ahmed Hassan	18P3206
Youssef Ahmed Ibrahim	18P7042

Table of Contents

Table of Figures

Introduction

This project presents a reliable and robust real-time general document collaboration between remote clients on the internet. The system supports various usability considerations (i.e., view of active peers, cursor preservation, and consistency across all clients).

Project Description

The system consists of two logical (however, replicated) parts. Client nodes where the editor interface lives. And, central points of synchronization that connect the client nodes.

Each client node can create new documents. Each document can be opened in a dedicated editor page. The editor page shows the active collaborating peers on the same document along with simulated cursors for those peers; this can be helpful for interpreting intentions of collaborators.

Each server node responds to client connection and disconnection events, and, in response, creates and closes client session, respectively, on the subject document. During a session, the client sends edits, the server synchronizes the edits to the shared document, and replies back with other peers' edits to the client.

The server can be replicated (i.e., horizontally scaled) as needed with minimal incur on performance, for all server instances are mutually exclusive and are backed with a real-time and shared database.

Beneficiaries

Generally speaking, any end-users with a need to collaborate on textual content in real-time are beneficiaries of the project.

Example Personas

- Software engineers working on code.
- Business analysts preparing plan documents.
- Students writing reports or group assignments.

Analysis

The analysis started by reviewing the literature. Next, integrating the needed features into a single system has been thoroughly planned. Lastly, deployment options and methodologies were researched.

Literature Review

The most widely known approaches for textual synchronization are

- Three-way merge.
- Operational transformation.
- Differential synchronization.

Differential synchronization is the approach implemented in this project, and here is why.

There-way Merge

Three-way merge is a blocking algorithm where the subject files are freezed before operation. This is the method used in **Git** source control systems.

Operational Transformation

Operational transformation is an event-based method where each edit is classified into an operation type. Each operation has a transformation that applies it to the document. This method has a couple of disadvantages: if at some point one client diverges from the rest, the upcoming edits accumulate errors and increase divergence, which requires a more sophisticated divergence prevention protocol. Also, implementing the right transformations for each operation is a tremendous effort compared to the next method.

Differential Synchronization

This approach takes advantage of the diff-match-patch operations already known, and employees a cycle of edits with reference to copies known as shadows at each node. This method has the advantage of inevitable convergence, for if a client happens to diverge, an upcoming cycle will include this difference in the sent edits: the edits are the current difference between shadow and content at any point in time not just the last applied operation.

Text Synchronization

A symmetrical algorithm employing an unending cycle of background *diff* and *patch* operations as depicted in Figure 1.

A walkthrough of a cycle:

1. Client Text is diffed against the Common Shadow.
2. This returns a list of edits which have been performed on Client Text.
3. Client Text is copied over to Common Shadow. This copy must be identical to the value of Client Text in step 1, so in a multi-threaded environment a snapshot of the text should have been taken.
4. The edits are applied to Server Text on a best-effort basis.
5. Server Text is updated with the result of the patch. Steps 4 and 5 must be atomic, but they do not have to be blocking; they may be repeated until Server Text stays still long enough.

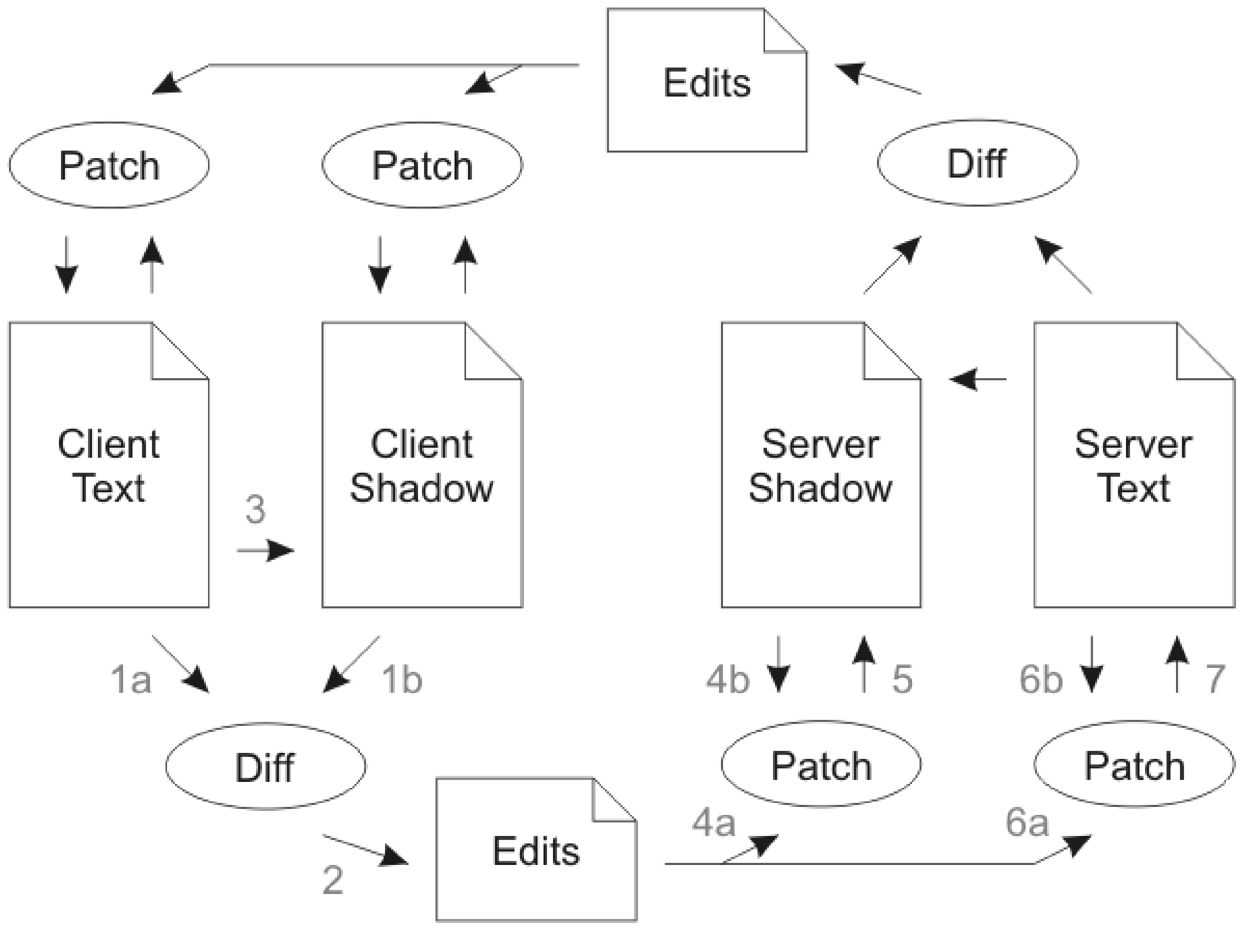


Figure 1 - Differential synchronization cycle.

Cursor Preservation

Absolute Referencing

The simplest preservation method. The offset (i.e., position) of the cursor in text before updates is recorded. After the *diff*, insertions before the cursor move it forward and deletions move it backward.

Here is an example of absolute referencing. The cursor is currently at offset 24, just before the word "slithy":

```
`Twas brillig, and the ^slithy toves
```

The following edits arrive from a remote user (strike-through represents a deletion, underline represents an insertion):

```
`Twas brillig, ~~and~~& the slithy toves
```

Three characters were deleted and one was added. If there were no deltas made to the cursor offset, the cursor would shift by two characters:

```
`Twas brillig, & the sl^ithy toves
```

By subtracting three and adding one to the cursor location, the cursor is moved to the expected location:

```
`Twas brillig, & the ^slithy toves
```

While good (and depending on the application, potentially "good enough"), absolute referencing is not perfect. Consider the following case:

```
`Twas brillig, and the slithy toves
  Did gyre and ^gimble in the wabe:
All mimsy were the borogoves,
  And the mome raths outgrabe.
```

If the middle two lines were swapped, the difference might look like this:

```
`Twas brillig, and the slithy toves
~~Did gyre and gimble in the wabe:~~
All mimsy were the borogoves,
Did gyre and gimble in the wabe:
  And the mome raths outgrabe.
```

Given that the original position of the cursor was in the middle of line 2, and that line 2 was completely deleted, then absolute referencing would set the new location of the cursor at the beginning of line 2:

```
`Twas brillig, and the slithy toves
^All mimsy were the borogoves,
  Did gyre and gimble in the wabe:
  And the mome raths outgrabe.
```

However, the user would expect their cursor to appear in the middle of line 3.

Context Matching

Another approach is to use fuzzy matching to find the location of the cursor after patching updates. First, you specify a context around the cursor; let’s say of size 8.

This is an example where the context is highlighted.

`It was such a nice day and we ate^ a lot.`

After edits, fuzzy matching will have enough context to get it back, considering the offset isn’t far, which introduces the problem in this approach.

It was a ~~nice~~brilliant day and we ~~ate~~8^ a lot.

If duplications in edits are introduced the fuzzy match cannot decide the right place. For example,

`It was such a nice day and we ate a lot.`

`It was such a nice day and we ate^ a lot.`

If a statement is inserted at the beginning, the matched location is missed.

`We decided to go out around the city.`

`It was such a nice day and we ate^ a lot.`

`It was such a nice day and we ate a lot.`

The user expects the cursor to be at the second line, but this is where the context matching fails.

Context Matching and Absolute Referenced Offsets

Combining the two approaches rendered better results than any of them. They solve each other’s problem.

The context matching approach seemed to match correctly but the offset confuses it. Calculating the absolute reference offset will help it search near the probable location.

So, the steps are

1. Capture the context and absolute offset of each point (the first step of context matching).

2. Apply the insertions and deletions while updating the absolute offsets of each point (the absolute referencing algorithm).
3. Restore the points on the new text based on the context and delta adjusted offsets (second step of context matching).

Let’s tackle the previous example again.

```
It was such a nice day and we ate a lot.

It was such a nice day and we ate^ a lot.
```

Initially, the cursor is at 74. After the edits the offset-corrected cursor is at 78 + 38 (length of inserted text before cursor) = 116

```
We decided to go out around the city.

It was such a nice day and we ate a lot.

It was such a nice day and we ate(^ with offset) a lot.
```

Now this is a trivial task for the fuzzy match.

Cursor and Active Clients Synchronization

The task was relatively straight forward. Each client heartbeats his cursor location with the edits, and the sever sends back last peers’ cursors with the edits back to the client.

The frontend application uses this information as fit for the interface.

Deployment

The system is deployed on an AWS EC2 instance for demo purposes. The website application can be accessed using this link.

Developing Team Members

- Ali developed the backend (server node).
- Mohamed deployed the system on AWS.
- Youssef developed the front end (client node)

System Architecture and Design

The system follows a 3-tier architecture: shared database, server, and client. The main 2 tiers are the server node, and client node. The nodes can be replicated and scaled as needed: explained in the description. However, one node in each layer is sufficient for demonstrating the full and proper functionality of the system. All these layers are interfaced with reliable and reasonably transparent middleware as depicted in Figure 2.

The server is implemented using the NodeJS runtime, and the client is implemented using ReactJS framework. Using the same language was a factor in our decision, for using the same language made the integration smoother and narrowed the needed expertise from the team.

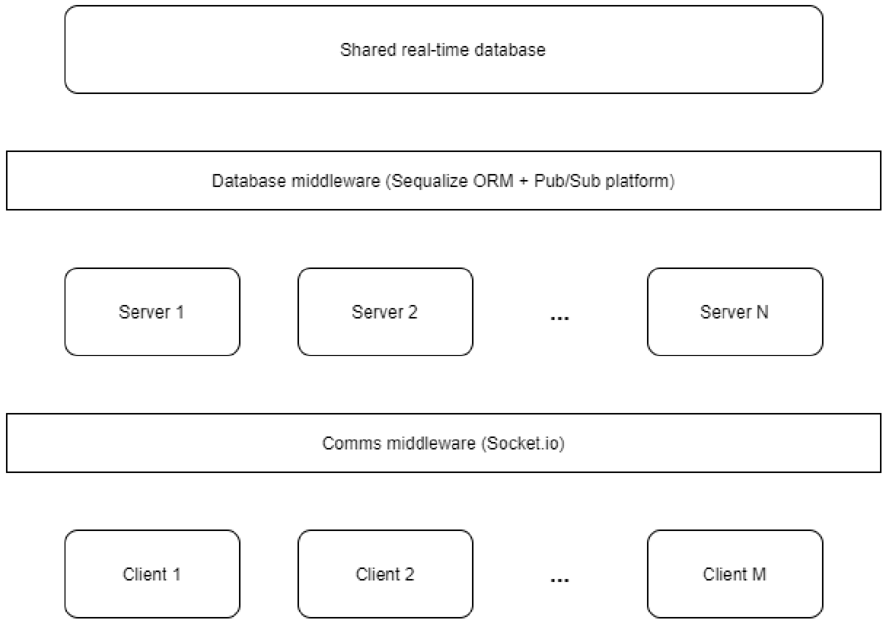

Shared real-time database
Database middleware (Sequalize ORM + Pub/Sub platform)
Server 1
Server 2
...
Server N
Comms middleware (Socket.io)
Client 1
Client 2
...
Client M

Figure 2 - Proposed System Architecture.

Server Node

The server serves three main functions:

1. The client node files (frontend).
   The needed HTML, JavaScript, and asset files for client (browser).
2. API for manipulating the documents (i.e., retrieving, creating).
   RESTful API. The documents are persisted in a shared database.
3. Document content synchronization.
   Socket.io is a reliable streaming communication implementation for the internet. It uses long-polling, and upgrades to WebSocket if supported by the client's browser.

For the synchronization, 3 actors are responsible for the process (association shown in Figure 3):

1. Comms
   Deals with communication channel (i.e., Socket.io) and interfaces with the Synchronizer.
2. Synchronizer
   Implements the differential synchronization cycle by interfacing with comms (clients' channel), and commands in return the Documents.
3. Document
   Represents the document entity and provides all the needed operations and utilities for manipulating the documents. Some of the operations are *open*, *close*, *diff*, and *patch.*

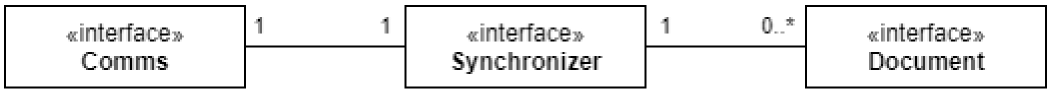


Figure 3 - Association of server synchronization actors.

Client Node

The client node's main functions:

1. User interface.
   The home screen of available documents, and the dedicated document editor page. ReactJS made the whole UI fairly modularized.
2. Cursor preservation algorithm.
   Thoroughly explained in analysis.
3. Document content synchronization.
   Symmetric to the server.

Testing Scenarios and Results

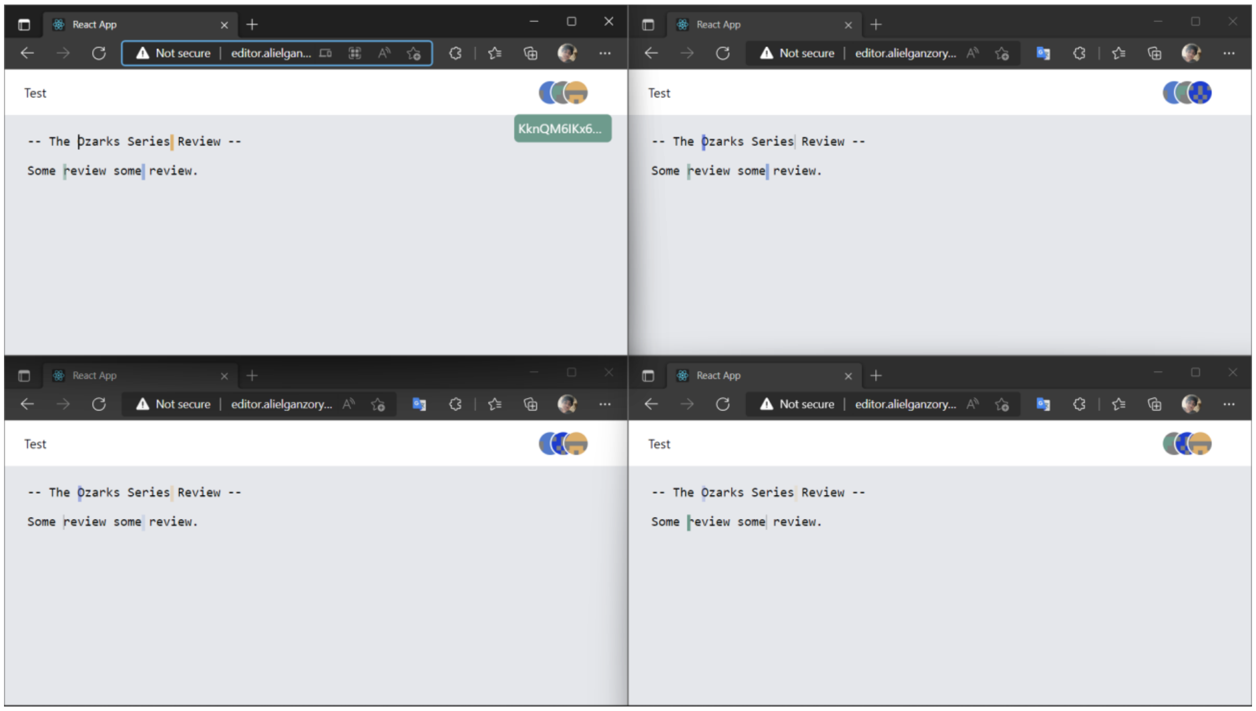
The application has undergone two holistic test scenarios.

Scenario 1 – Local instances

Four app instances are launched in different browser windows (i.e., four different client sessions) as seen in Figure 4.

Edits are made from all four of them. The system successfully synched all updates and converged to the same document content. During all edits, the cursor is preserved.

Also, the active users' avatars showed correctly at the top app bar.



```
-- The Ozarks Series Review --
Some review some review.
```

```
-- The Ozarks Series Review --
Some review some review.
```

```
-- The Ozarks Series Review --
Some review some review.
```

```
-- The Ozarks Series Review --
Some review some review.
```

Figure 4 - Four simultaneous client sessions.

Scenario 2 – Remote instances

Three app instances are launched in different browsers on different machines that geographically remote from other (i.e., four remote client sessions.)

The system behaved in the same successful manner shown in scenario 1 and this demo video.

End-User Guide

Access

Method 1

Deployed version at http://editor.alielganzory.me

Method 2

1. Download source code from Git repository.
2. Run `yarn install` in project folder.
3. Run `yarn build` to generate production files.
4. Run `yarn start` to start the app on your local machine.
5. Access the app on `http://localhost:8000`.

Usage

Home page
All created and available documents are listed.

You can select any document to open its dedicated editor page.

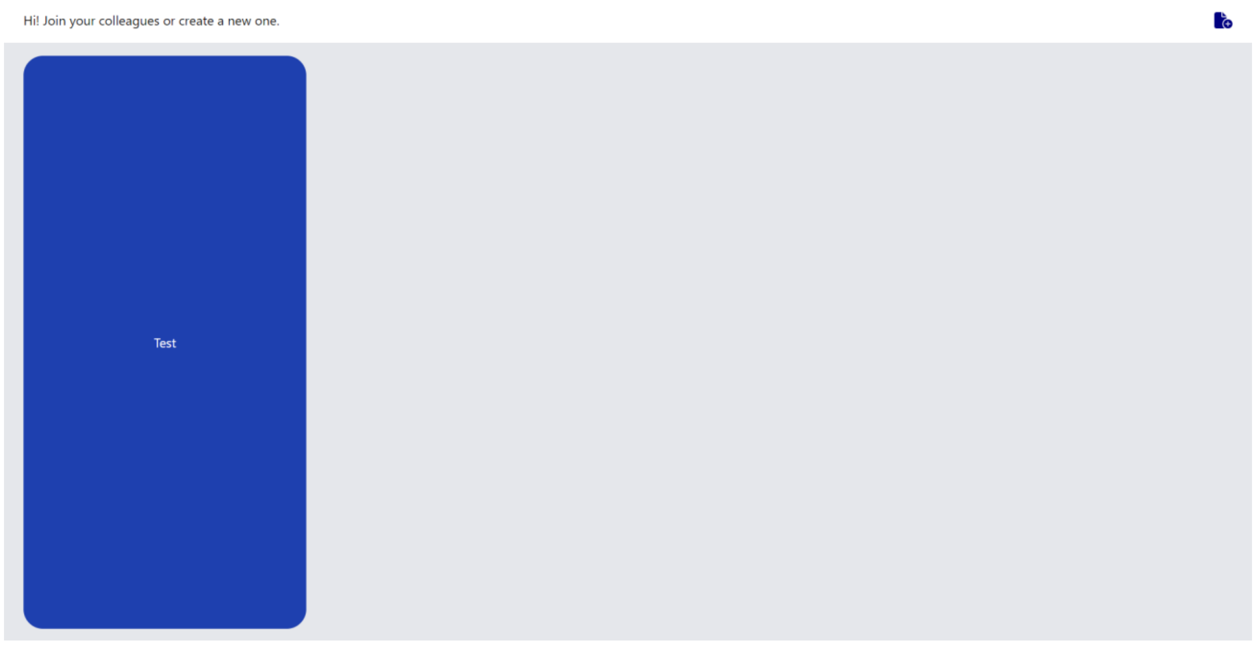


Figure 5 - Client Home Page.

Creating a new document

Click on the button in the upper right corner, and type the desired document title, and then click create (visible in Figure 6).

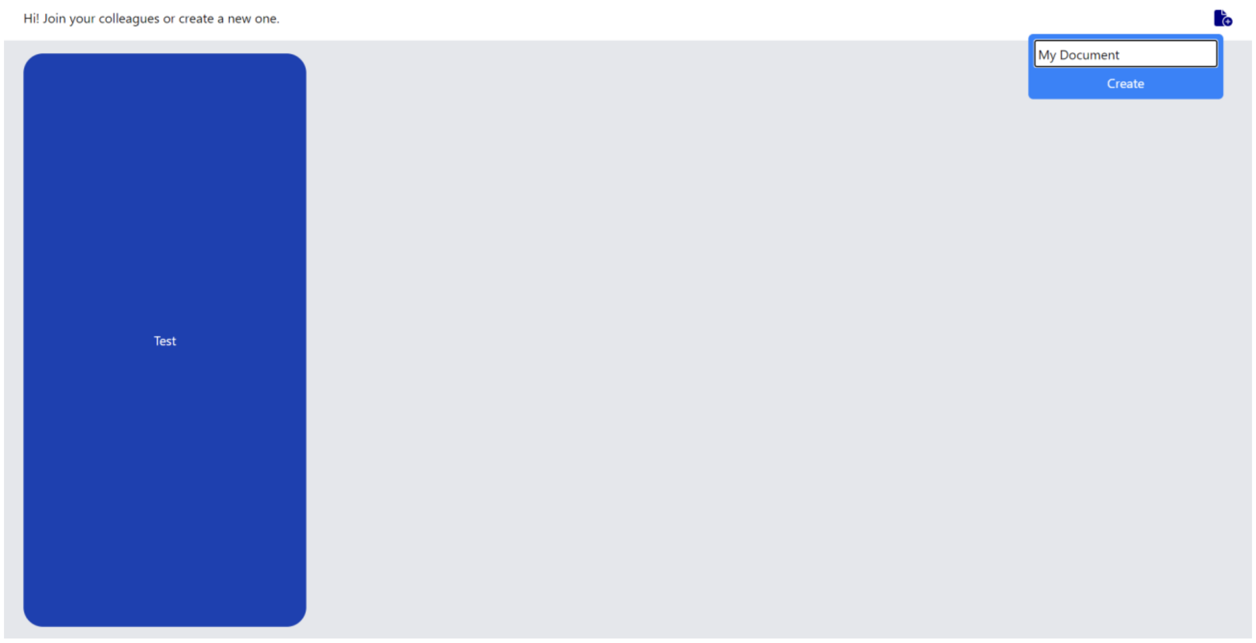
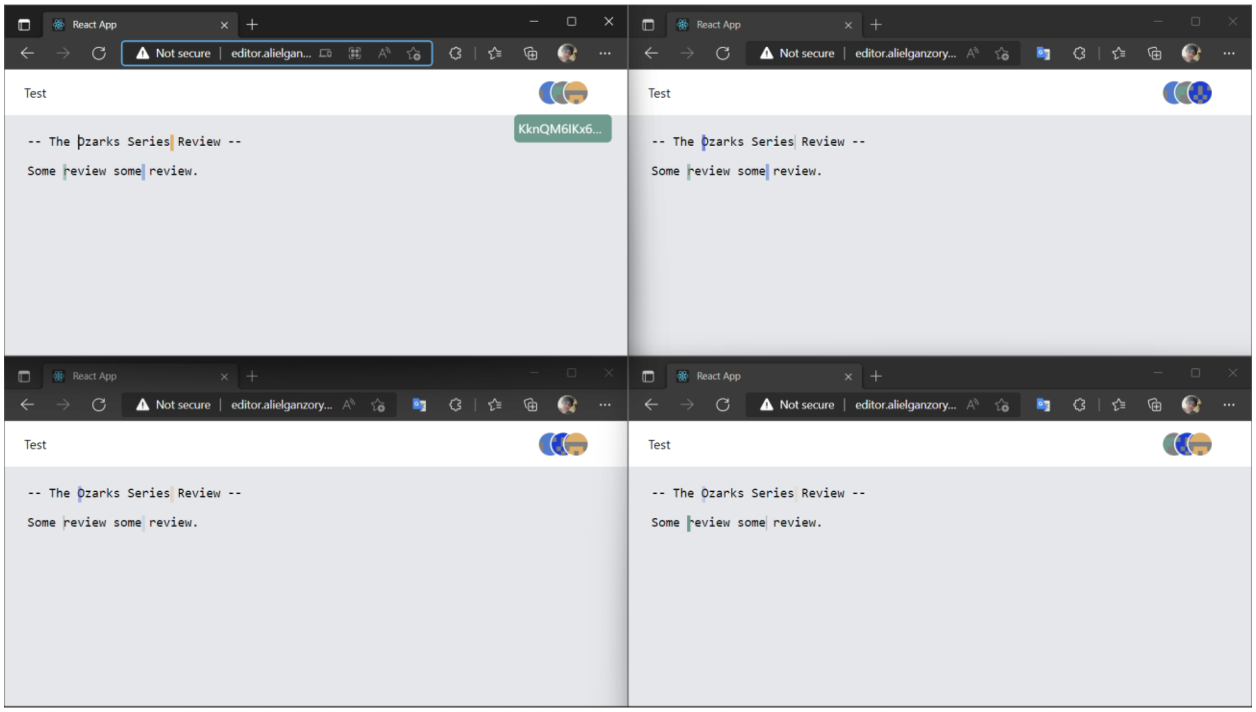


Figure 6 - New Document Creation.

Editor page

Voilà! Start editing. You'll see other active peers' unique avatars on the top right. Their cursors are simulated also at the last place they are at.



```
-- The Ozarks Series Review --
Some review some review.
```

Figure 7 - Simultaneous client editors editing the same document.

Conclusion

After all, building a reliable real-time collaborative editor proved to be a humongous task, for all the interdisciplinary parts that go into it — as discussed throughout this document. Nonetheless, we tackled all the theoretical and technical difficulties by being focused and pragmatic: we reviewed the literature, analyzed the different techniques, and then designed, implemented, and deployed the system.

We hope this project helps others seeking an alike endeavor or a related one in the future. We did try our best to organize the architecture and provide in-code comments wherever we could.

References

[1] *Differential synchronization*. Neil Fraser: Writing: Differential Synchronization. (n.d.). Retrieved May 18, 2022, from https://neil.fraser.name/writing/sync/

[2] *Cursor preservation*. Neil Fraser: Writing: Cursor Preservation. (n.d.). Retrieved May 18, 2022, from https://neil.fraser.name/writing/cursor/

[3] *Fuzzy patch*. Neil Fraser: Writing: Fuzzy Patch. (n.d.). Retrieved May 18, 2022, from https://neil.fraser.name/writing/patch/

[4] *Socket.io Official Documentation*. SocketIO RSS. (2022, May 6). Retrieved May 18, 2022, from https://socket.io/docs/v4/

[5] Node.js. (n.d.). *NodeJS Official Documentation*. Node.js. Retrieved May 18, 2022, from https://nodejs.org/en/docs/

[6] *React – a JavaScript library for building user interfaces*. – A JavaScript library for building user interfaces. (n.d.). Retrieved May 18, 2022, from https://reactjs.org/