

Final Project Report

Team 704

members: Joseph Gu, Ise (Britney) Okiria, Aminah Statham-Adams, Daniel Rosenman

Feature Overview and User Manual

Build Steps

To begin building the application, it is recommended that users clone the repository onto their local machine by using the command 'git clone,' along with the appropriate URL, which can be obtained directly from the repository located on Github at the following link:

<https://github.com/neu-cs4530/fall23-team-project-group-704/tree/main>

From the 'main' branch of the repository, click on the green 'code' button, and copy the URL located under the 'SSH' tab. Then, open a new project folder in your IDE of choice and enter the command 'git clone [URL here]' into the terminal. This should clone the folder and file structure of the repository into the IDE. (Note that for this approach to work, it is necessary to have an SSH key set up on your github account).

Setup the .env files in the backend and frontend using the instructions in the README.md.

Setting up the backend

To run the backend, you will need a Twilio account. Twilio provides new accounts with \$15 of credit, which is more than enough to get started. To create an account and configure your local environment:

1. Go to [Twilio](https://www.twilio.com/) and create an account. You do not need to provide a credit card to create a trial account.
2. Create an API key and secret (select "API Keys" on the left under "Settings")
3. Create a '.env' file in the 'townService' directory, setting the values as follows:

Config Value	Description
TWILIO_ACCOUNT_SID	Visible on your twilio account dashboard.
TWILIO_API_KEY_SID	The SID of the new API key you created.
TWILIO_API_KEY_SECRET	The secret for the API key you created.
TWILIO_API_AUTH_TOKEN	Visible on your twilio account dashboard.

Configuring the frontend

Create a '.env' file in the 'frontend' directory, with the line: 'NEXT_PUBLIC_TOWNS_SERVICE_URL=http://localhost:8081' (if you deploy the towns service to another location, put that location here instead)

For ease of debugging, you might also set the environmental variable 'NEXT_PUBLIC_TOWN_DEV_MODE=true'. When set to 'true', the frontend will automatically connect to the town with the friendly name "DEBUG_TOWN" (creating one if needed), and will *not* try to connect to the Twilio API. This is useful if you want to quickly test changes to the frontend (reloading the page and re-acquiring video devices can be much slower than re-loading without Twilio).

The next step is to install necessary packages in order for the application to run properly.

Navigate to the 'townservice' directory in the command terminal and run the command 'npm install.' Then, run 'npm start.' Next, in the 'frontend' directory, run the commands 'npm install' and 'npm run dev' consecutively. In this directory should appear the message 'open on

local host?'; click this link to open the application in your browser. You are now ready to use the app!

Interacting with the App

Known Bug: documents shared with you under view-only permission may be initially editable. This is easily fixed by refreshing the UI by creating a new document of your own. Then, the view-only document should become truly view-only.

To begin interacting with the app, use the arrow keys on your keyboard to navigate to the CoveyBoards Area of the game, located at the bottom of the default room between Basement Dining Tables 1 and 2. Once close enough to the CoveyBoards Area, a pop-up will appear with the message, 'here is a doc area,' as shown below.



Once this message appears, click on the spacebar to open the CoveyDocs interface. The initial screen is a sign-in page, which gives the option to either create a new username and password, or to enter an existing username and password to access any documents associated with an existing account.

Welcome to CoveyDocs! :)

Username *

Enter the username you would like to go by.

Password *

Enter the a secure password

Sign up

Sign in

Cancel









After signing in, the documents directory will appear (signing up with a new username and password will show an empty directory, while signing in with an existing account will show a list of documents you have access to, if any).

Document Directory


NAME	OWNER	CREATED AT	PERMISSIONS
New Doc			

Cancel

Click on the 'New Doc' button to create a new document. This opens a blank document. Documents can be edited by inserting and formatting text, inserting images, or reordering or deleting text and images. To insert text, click within the white document screen and type. Text can be formatted by highlighting with your cursor and then selecting either the bold, italic, or highlight formatting options shown below.

← → B I        

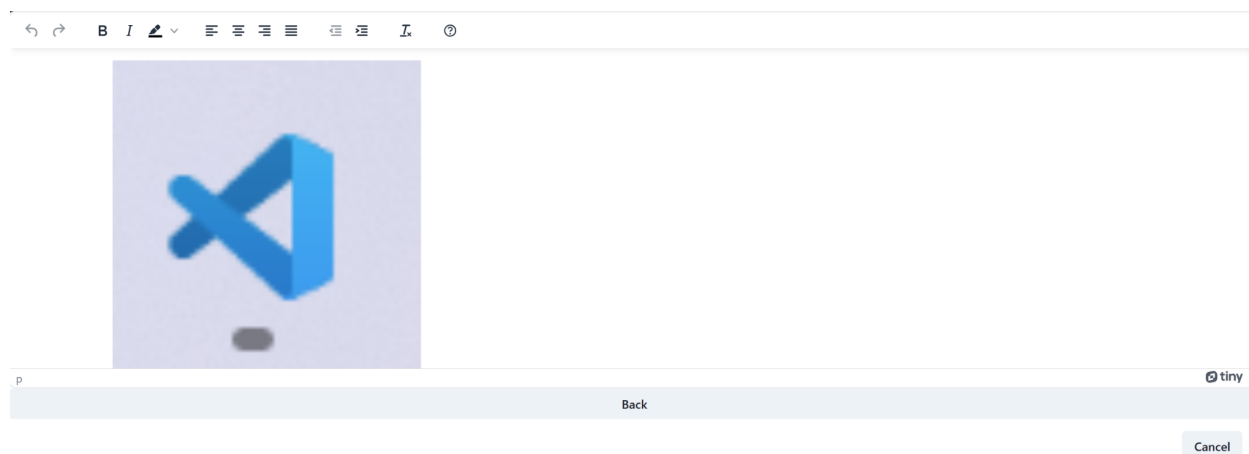
this is a default doc

 tiny

Back

Cancel

Images can be inserted by simply copying an image to your clipboard and using the ‘Ctrl + V’ shortcut within the document to paste the image. Images can also be resized by dragging the blue border that appears upon clicking the image. An example of a document that contains formatted text and an image is shown below.



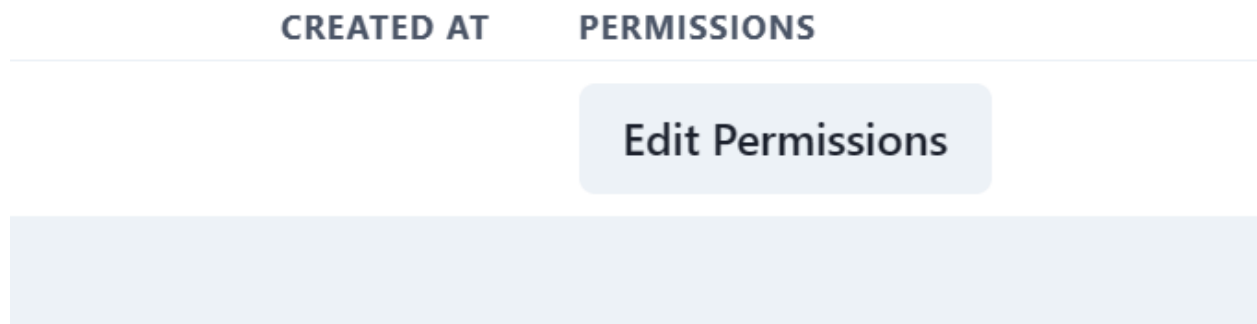
To undo or redo a change, simply click on the ‘back arrow’ and ‘forward arrow,’ respectively. Only a certain amount of undo and/or redo commands can be done consecutively, which is determined by the logic of the tinyMCE library which was used to create the document component.

To return to the document directory from a document, click on the back button (note that clicking the ‘cancel’ button at any time will exit the CoveyDocs Area interface entirely). After a document has been created, it will appear in the document directory with its name, owner, created date, and an edit permissions button displayed, as shown below.

Document Directory

NAME	OWNER	CREATED AT	PERMISSIONS
New Document	Paula	2023-11-29T03:48:32.386Z	Edit Permissions
New Doc			
Cancel			

To edit the permissions of a document, scroll to the edit permissions button and click on it.



This opens the permissions editing interface, which provides the user with the ability to add a new viewer/editor to a document or remove them from the doc. If the user is already a viewer/editor, the permission level can be increased or decreased by simply selecting the new permission level and pressing submit.

Edit the Permissions of this Document

People who can edit:

Bill

People who can view:

Add a New User

User ID:

☐ Add as Editor

☒ Add as Viewer

☐ Remove

Submit

Exit

Cancel

You can test out the multi user functionality of our app by signing in another user B in another tab on our browser. Notice how the documents user A shares with user B are visible to user B in their document directory. If the document shared with B is viewer-level permissioned, B cannot edit the documents. When a person has a document open, edits are pushed to the backend at 5 second intervals. Therefore, edits made by A will be visible to B in 5 second intervals, where the frontend pushes updates to the backend.

Document Directory

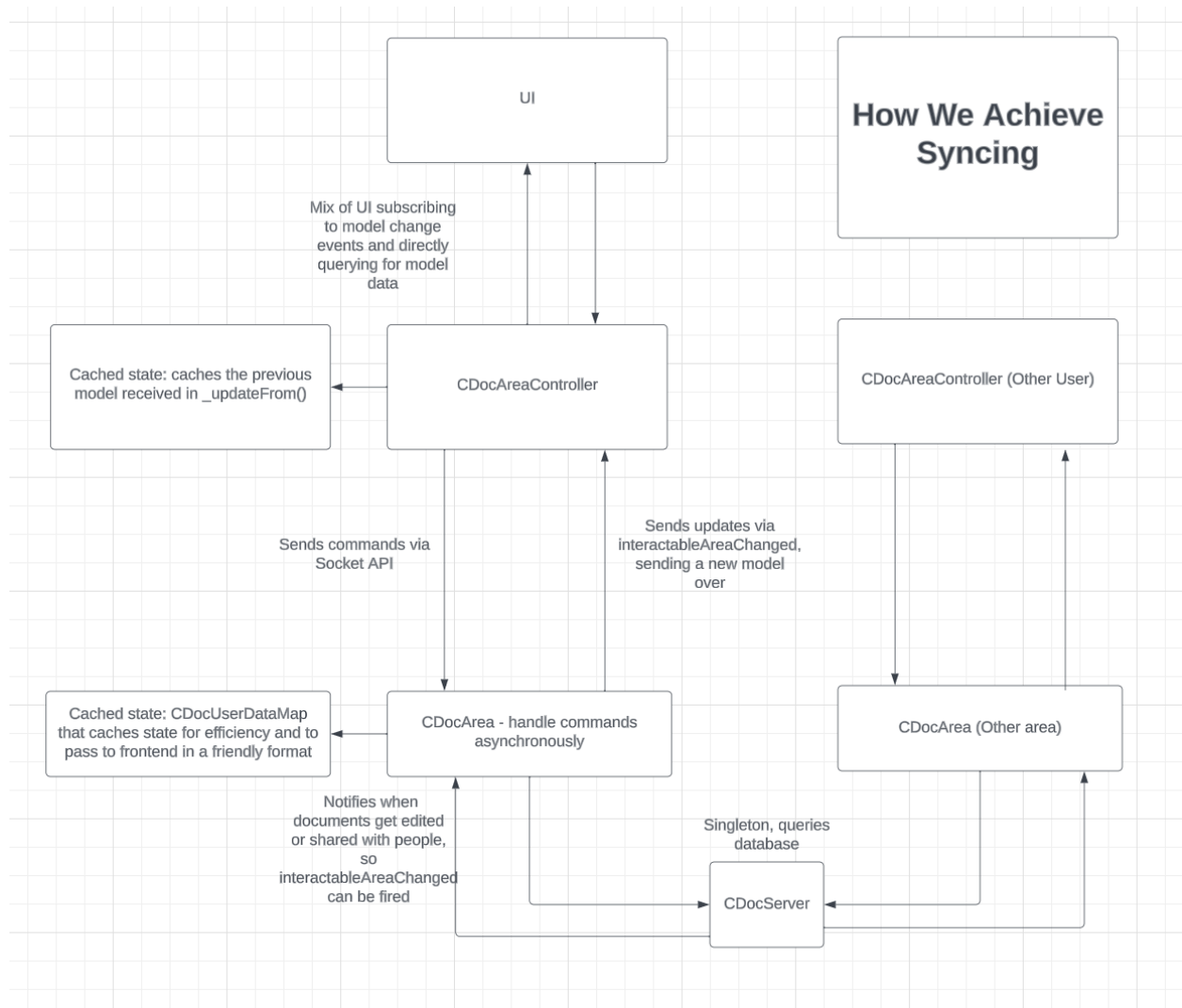
NAME	OWNER	CREATED AT	PERMISSIONS
New Document	Alice	2023-11-28T19:00:21.515Z	<div>Edit Permissions</div>
New Document	Alice	2023-11-28T19:59:34.925Z	<div>Edit Permissions</div>
New Document	Alice	2023-11-29T03:19:13.943Z	<div>Edit Permissions</div>
New Document	Joseph	2023-11-28T21:59:21.341Z	
New Doc			

Cancel

Known Bug: documents shared with you under view-only permission may be initially editable. This is easily fixed by refreshing the UI by creating a new document of your own. Then, the view-only document should become truly view-only.

For proof of data persistence, close the browser tab and sign in again as user A. You will see that the documents owned are the same as before. Even if the server were restarted, the data will persist because it lives in a PostgreSQL database.

Technical Overview



The above diagram is a summary of how we achieve syncing and our architecture as a whole.

We utilize the socket API to send commands to the backend for operations such as getting documents and creating new documents. These commands sometimes return values such as documents or booleans, because it is useful to allow our UI to actively query for data, especially for things like loading a document or signing in.

We modified `handleCommand` in the backend to be async since our database operations are all async.

There is also an observer pattern for getting other state updates, because the frontend sometimes has to get updated without actively querying for new state from the backend via socket commands. For example, if another user edits the same document, the only way to get this state to reflect in the frontend is via observer pattern - active querying won't work unless done every fixed time step. So the backend intermediately fires interactable area change events, passing the relevant state to the frontend.

The database querying object, `CDocServer`, is a singleton. That way, all requests go through one object which can notify all listeners of important events such as a particular document getting updated. These listeners are functions in `CDocArea` and can fire their own events to notify the frontend.

There are two caching mechanisms, one in the `CDocAreaController` and `CDocServer`. They mostly function to reduce unnecessary event firing if state hasn't changed, or to reduce the frequency of queries.

Important outside libraries we used are TinyMCE and PostgreSQL. TinyMCE is a rich text editor UI that allows exporting and importing content as raw HTML. Using TinyMCE, we didn't need to worry about representing formatted text or images as it was all raw HTML. PostgreSQL allowed us to have long term data persistence. We used three tables, Documents, Users, and Permissions.

Process Overview

Sprint 0: Planning and Learning

During Sprint 0, the team encountered an unexpected hurdle in testing, so focused our efforts towards research and learning. Ise took the initiative to delve into the selection of a suitable database API/library, recognizing its impact on our code structure. Joseph, foreseeing the importance of regular team check-ins, proposed the integration of weekly meetings, scheduled for Mondays in the morning, with flexibility for additional sessions based on team needs and availability. Meanwhile, Aminah and Daniel identified a need to revisit the application design, acknowledging the challenges in determining the optimal distribution of elements between the model, view, and controller. The team collectively decided to use this sprint as a foundation for a robust understanding of tools and frameworks.

Sprint 1: Planning and Setup

In Sprint 1, meticulous planning was executed through diagrams and detailed descriptions of the envisioned product. Joseph, as part of his commitment to testing, informed the team that his testing task had been branched on GitHub. Daniel shared that all his tasks would be visible on GitHub upon completion. Ise, contributing to testing as well, communicated that her tasks were also available in the designated branch. Aminah also added a branch for her tests. The team collectively acknowledged that certain aspects of the project remained mysterious until the implementation phase, emphasizing the importance of expediting the implementation process for enhanced project comprehension.

Sprint 2: Implementation Challenges

In Sprint 2, the team shifted its focus towards implementation, encountering challenges in communication and integration between the frontend and backend.

Joseph took the lead in backend development, concentrating on setting up the backend infrastructure. He dedicated efforts to creating queries for the database and developing methods to connect to the town controller, laying the foundation for a robust backend structure.

Daniel, recognizing the importance of regular communication, emphasized the significance of stand-up meetings. He actively participated in these check-ins, providing valuable insights that contributed to overcoming difficulties related to design and integration. In addition, Daniel focused on designing the document page using TinyMCE, contributing to the overall user interface.

Ise, leveraging research conducted in Sprint 0, assumed a prominent role in frontend development. She spearheaded the creation of the sign-in page and document directory page, focusing on delivering an optimal user experience. Ise also took charge of setting up the database and servers for the site using render.com. When challenges arose with render.com, Ise promptly facilitated the transition to Heroku, showcasing adaptability in response to unforeseen issues.

Aminah, contributing to the design aspects, played a crucial role in shaping the permissions page and designing the CoveyDocs controller.. Her efforts were instrumental in ensuring a cohesive and user-friendly design across different sections of the application.

Collectively, the team navigated through the hurdles, recognizing the iterative nature of development. The collaborative efforts of Joseph, Aminah, Daniel, Ise, and the entire team underscored the importance of continuous collaboration in addressing challenges and driving the project forward. This sprint highlighted the diverse skill sets and contributions of each team member in achieving a successful implementation phase.

Sprint 3: Rush to Completion

Sprint 3 marked a sprint to the finish line, with a primary goal of completing testing and addressing smaller bugs. Everyone worked in tandem to resolve unforeseen issues that emerged, including the unexpected malfunction of Render. In response, the team made a decisive move to switch to Heroku for backend hosting, maintaining Render exclusively for the frontend. Despite being behind on some implementation tasks, the team displayed resilience and adaptability, prioritizing problem resolution and project completion.

****Agile Techniques Utilized****

- ****Sprints:**** Joseph, Aminah, Daniel, and Ise actively participated in the planning and execution of sprints, ensuring tasks were well-defined and progress was tracked.
- ****Sprint Reviews:**** The team collectively engaged in reviews at the end of each sprint, allowing for a comprehensive analysis of completed tasks and adjustments for subsequent sprints.
- ****Daily Stand-ups:**** During challenging periods, the team as a whole adopted text-based daily stand-ups to enhance productivity and maintain effective communication.
- ****Blameless Reviews/Pull Requests:**** All team members actively participated in code reviews, providing valuable feedback and ensuring a smooth merging process on major branches. This practice contributed to a shared understanding of the codebase and facilitated continuous improvement.