Computer Science 343
Project 2 Fire Docs

Iliya Meyer, 24802824
Ryan Shearer, 24695394
Leonard Brugman, 24791040
Dietrich van Eeden, 23844701
Stanley Shikwambi, 24735728
Danielle Kleyn, 24696781

October 17, 2023

1 Introduction

Fire Docs is a collaborative markdown note-taking Website application, that allows simultaneous editing of notes by multiple users. In almost every single work environment found today, the writing down of notes has proved to be useful and almost mandatory. Fire Docs allows multiple parties to create, edit, and delete notes in their respective domains.

The ability to collaborate on a set of notes is a game-changer, in the sense that changes are realised immediately. This allows instantaneous feedback from other contributors, which improves overall productivity.

2 Description

The use case diagram below illustrates the different types of functionality that Fire Docs provides. A user can register, log in and use the various services offered by Fire Docs. A user can create new notebooks, which includes choosing a title, category and users to share the notebook with. These notes can be edited and they can be shared with more users later on. These notes can also be deleted. A list of a user's notebooks appears on the homepage under the title "Your Notebooks", and these list of notebooks can be searched through and filtered. This includes searching through the notebooks by title, filtering notebooks by category and sorting the list of notes by the last time any user edited it.

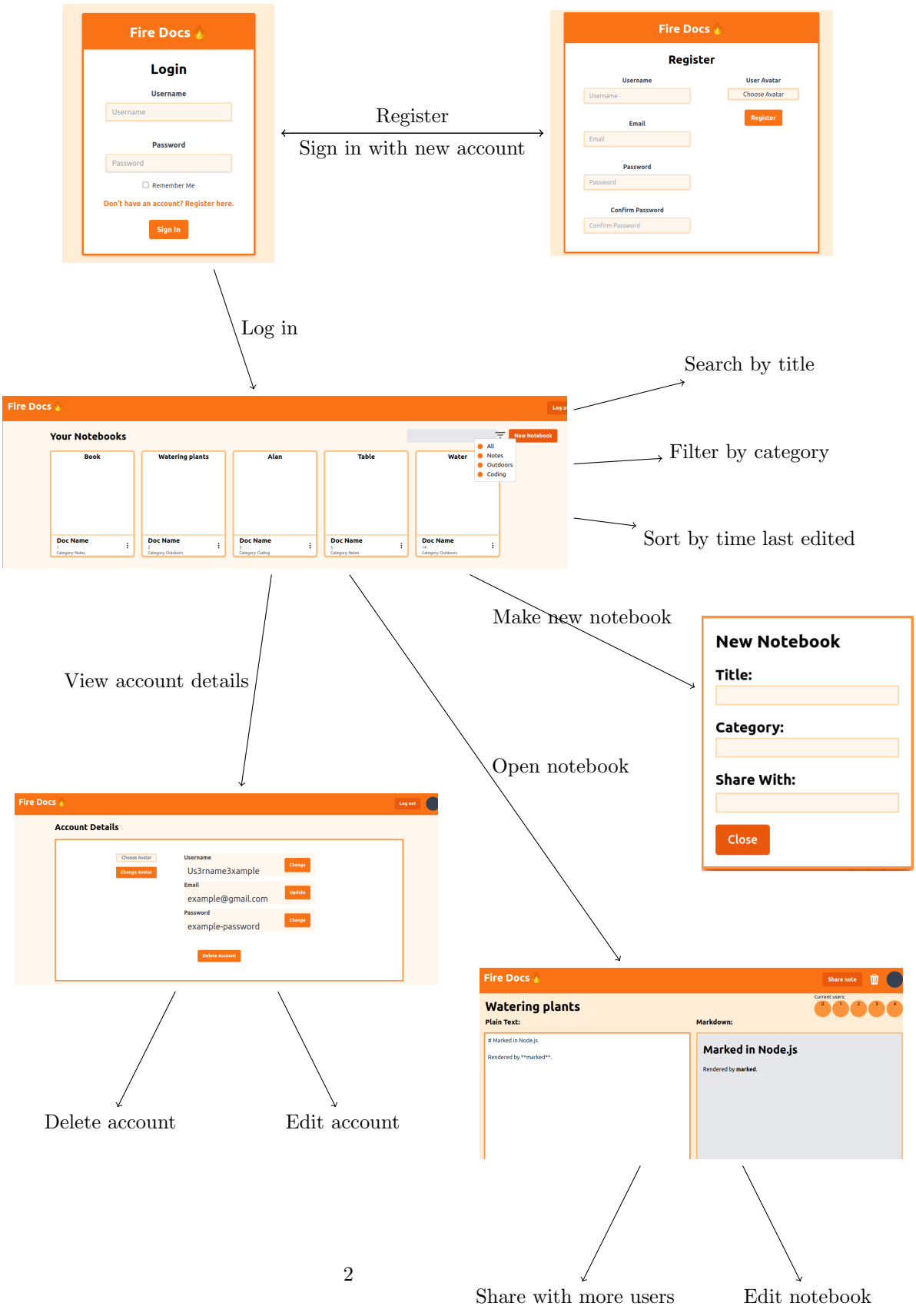
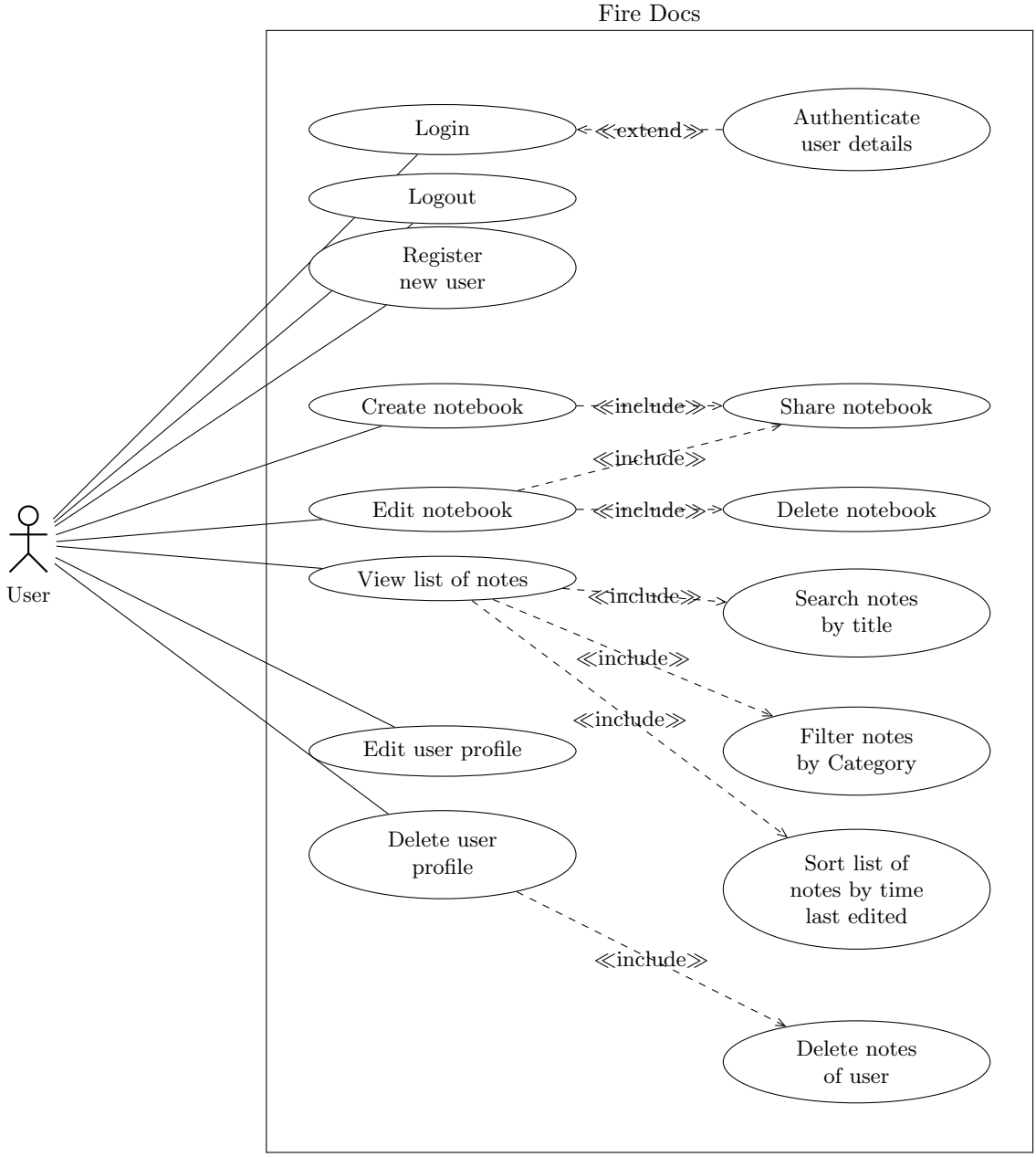


Figure 1: Interface modelling

Along with editing notebooks, a user's profile details can also be edited and their username, email, password or avatar can be changed. An account can also be deleted in which case all associated notes will also be deleted.

2.1 Use case diagram



2.2 Data modelling

At the heart of our data model are three primary entities: Users, Notes, and Categories. The Users table encompasses essential attributes such as username, email, password, and avatar_url. The Notes table contains information about the respective notes; this is obviously essential. Each has a NoteID which we use to link users to notes in the UserNotes table. The Categories table introduces a classification system for notes, allowing them to be organized under different categories for better user experience. The relationships between these tables are defined using primary and foreign keys, ensuring data integrity. For instance, a note is linked to a user (its owner) and a category, establishing a many-to-one relationship. Additionally, the User Notes junction table facilitates a many-to-many relationship between users and notes, allowing for collaborative features.

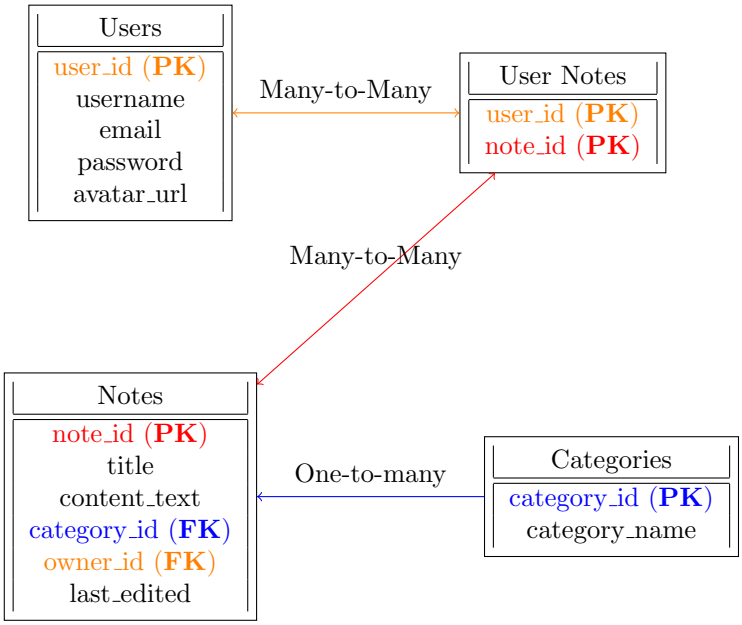


Figure 2: Data Model between users, notes and categories

2.3 Operating environment

We made use of the following in our project

- Marked
- React.js
- Tailwind CSS
- Express
- Node.js
- PostgreSQL

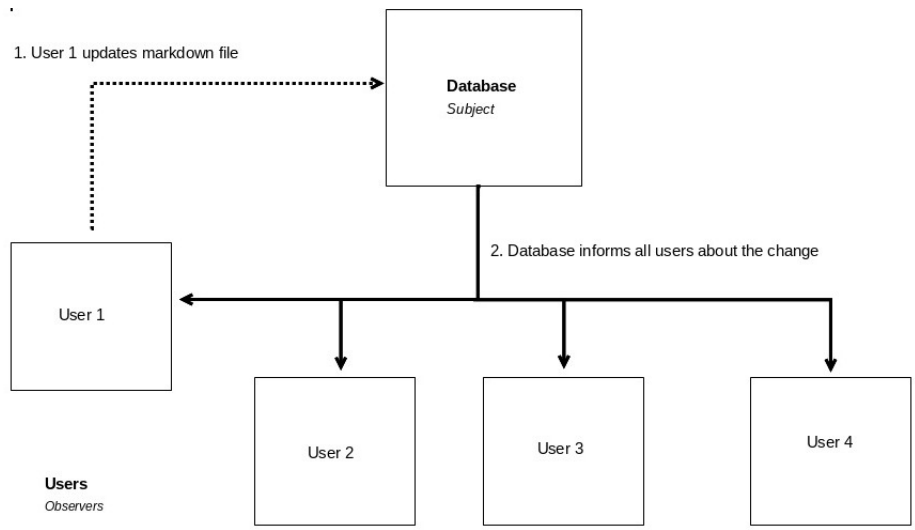
- Docker
- Websocket
- Selenium

Our applications front end is made using **React.js**, which is styled with **Tailwind CSS** allowing for an enhanced user experience. We chose to build the frontend using **Express** API with **Node.js**. **Docker** is used to easily set up and run the server. Very basic testing was also implemented with **Selenium**. **Websocket** was used to facilitate communication between a client and the server, most notably for when a user makes changes to a notebook and these changes are sent to the server, where it is then updated among all other users that have access to this notebook.

2.4 Design patterns

2.4.1 Client

Observer Design Pattern: The Observer design pattern is employed to facilitate real-time collaboration and instantaneous updates. Leveraging the capabilities of WebSockets, the server acts as the "subject," maintaining a list of connected clients as its "observers." When a user makes changes to a note, the server is informed of this change. In turn, the server, acting upon its role in the Observer pattern, notifies all other relevant clients of this modification. This ensures that all users viewing or editing the same note receive real-time updates, thereby allowing for collaborative editing of notes. Through this implementation of the Observer pattern, we ensure that changes are broadcasted to all relevant parties.



2.4.2 API

Middleware Pattern: We use the Middleware pattern to streamline the processing of requests and responses within the Express framework. Middleware

functions have the capability to access and manipulate the request and response objects, and they play a crucial role in defining the flow of control in the application. By chaining multiple middleware functions, the application can execute sequential operations such as authentication, logging, data validation, or even error handling. For instance, before a user can access certain routes or endpoints, a middleware function might check the validity of a user's authentication token. If the token is valid, the function will pass control to the next middleware or route handler; if not, it might redirect the user to a login page or return an error. This approach, facilitated by the Middleware pattern, not only promotes code reusability but also ensures that the application remains maintainable, scalable, and easily extensible. Each middleware function has a distinct responsibility, and by orchestrating them effectively, we achieve an efficient request-response lifecycle.

Singelton Design Pattern: The Singleton pattern is employed to ensure both efficiency and consistency in resource management. The primary example of this is the Express application. When the Express application is initialized using express(), it creates a single, unique instance, referred to as app or server. This instance becomes the central hub for registering routes, middleware, and managing all incoming requests. As the application scales and handles multiple requests, this single app instance ensures that there's a unified point of control and management. Additionally, while the provided files don't explicitly detail the database connection process, it's a standard practice in Node.js applications to employ the Singleton pattern for database connectivity. Establishing a database connection can be resource-intensive, and it's inefficient to create a fresh connection for every database operation. Instead, a single connection or a connection pool is created and reused throughout the application's lifecycle. This approach not only optimizes resource usage but also prevents potential connection leaks, ensuring the application's robustness and responsiveness.

3 Assignment of tasks

The distribution of tasks was split into 3 parts: frontend and backend. We decided to have 2 members (Leonard and Dietrich) responsible for frontend, 2 members (Iliya and Ryan) responsible for backend and the remaining 2 would be fullstack (Danielle and Stanley). Report writing was split among those who finished their assigned work and had time to spare, mainly Danielle, Leonard, Ryan and Stanley.