Computer Science 343 14 October 2024

Project 2 – Note Taking App

Group Members – Add student numbers

- Liam Abbott (24737615)
- Lara Jacobs (23260122)
- Joel Lee (23797207)
- Seisa Likotsi (24806935)
- Brad Slingers (17854865)

Introduction

In today's digital world, effective collaboration tools are essential. This note-taking app facilitates real-time collaboration, allowing users to co-author, edit, and share notes seamlessly. Key features include note creating/updating/deleting and real time collaboration via shared notes.

The app uses React for the frontend, Node.js for the backend, and Supabase for authentication and real-time data storage. WebSockets enable instant updates across collaborators, while Amazon S3 handles image storage.

Our goal is to enhance productivity and teamwork through a reliable, intuitive platform. This report outlines the app's key components, including use cases, data modeling, operating environment, authentication, and design patterns implemented in both frontend and backend.

Use Case Diagram

Actors

- **User**: The main actor interacting with the system. Users can sign up, sign in, and perform actions such as creating, editing, deleting, and sharing notes.
- **Database**: The secondary actor representing the system that stores all the data, including user credentials, notes, categories, and collaboration information. The database acts in response to HTTP requests at various endpoints.

Use Cases

- **Sign Up**: Users can create an account by signing up. If the details (username and email) are not already taken, then a user will be redirected to the login page.

- **Sign In**: Existing users can log into their account. If a user had previously enabled **remember me,** then cookies are used to skip this process.

- **Create Note**: After logging in, users can create new notes, which are stored in the database. Each note can have a title, category and content associated with it.

- **Edit Note**: Users can modify their existing notes; changing the content, title, or category. This action includes category management for assigning or reassigning categories.

- **Delete Note**: Users can delete notes. A deleted note is permanently removed from the system and cannot be retrieved.

- **Share Note**: Users can share notes with others for collaboration. Collaborators are added to the note via an email request. Collaborators can edit a note in real time and see the changes made by other collaborators.

- **Filter Notes**: Users can filter their notes by category, creation date and title. This allows to instantly search for relevant notes.

Error Handling

Several actions in the system involve error handling, which is extended from the primary use cases. For example:

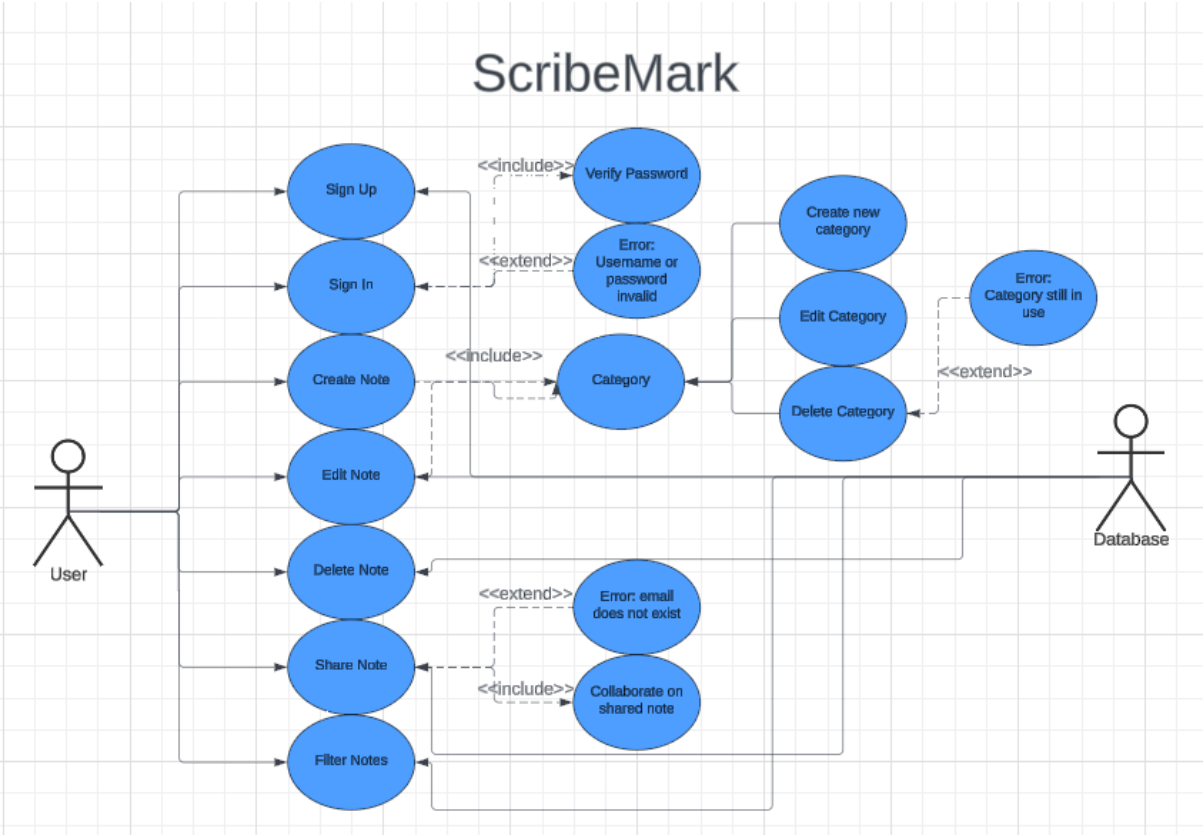
- **Error: Username or Password Invalid**: This error can occur during both sign-up and sign-in when the provided credentials are incorrect.
- **Error: Email Does Not Exist**: This error is triggered when attempting to share a note to an email address that is not registered in the system.
- **Error: Category Still in Use**: When attempting to delete a category that is still assigned to notes, category is not deleted.

Category Management

Categories are important for grouping notes. A user can create a new category, update, or delete an existing category. A category can only be deleted if it is not in use elsewhere.

Collaboration

The app supports real-time collaboration on shared notes. Users can invite others to collaborate, enabling multiple users to view or edit the same note. This action is included in the **Share Note** use case, as sharing is the primary way to add collaborators.



Data Modelling

User

The `User` entity represents individuals using the app. Collaborators are also regarded as users. Each user is identified by a unique `email` (the primary key), along with attributes like `username`, `password`, and `avatar_url`. The avatar_url is a link to the S3 storage of the image. The `reset_token` and `reset_token_expiry` fields support password recovery functionality. The `User` entity is not directly linked to the `Note` entity; instead, the connection is made by the `Collaborator` entity, which provides the names of all collaborators on the note.

Note

Note entity represents a single note in the system. Attributes include title, content, created_at, updated_at, and status to track a note's lifecycle.

Each note can be linked to a Category through the categoryId foreign key, allowing users to organize notes into categories. A note can only have one category, but the same category can be appplied to many different notes.

A collaborator can have zero to may notes and each note can have zero to many collaborators.

Collaborator

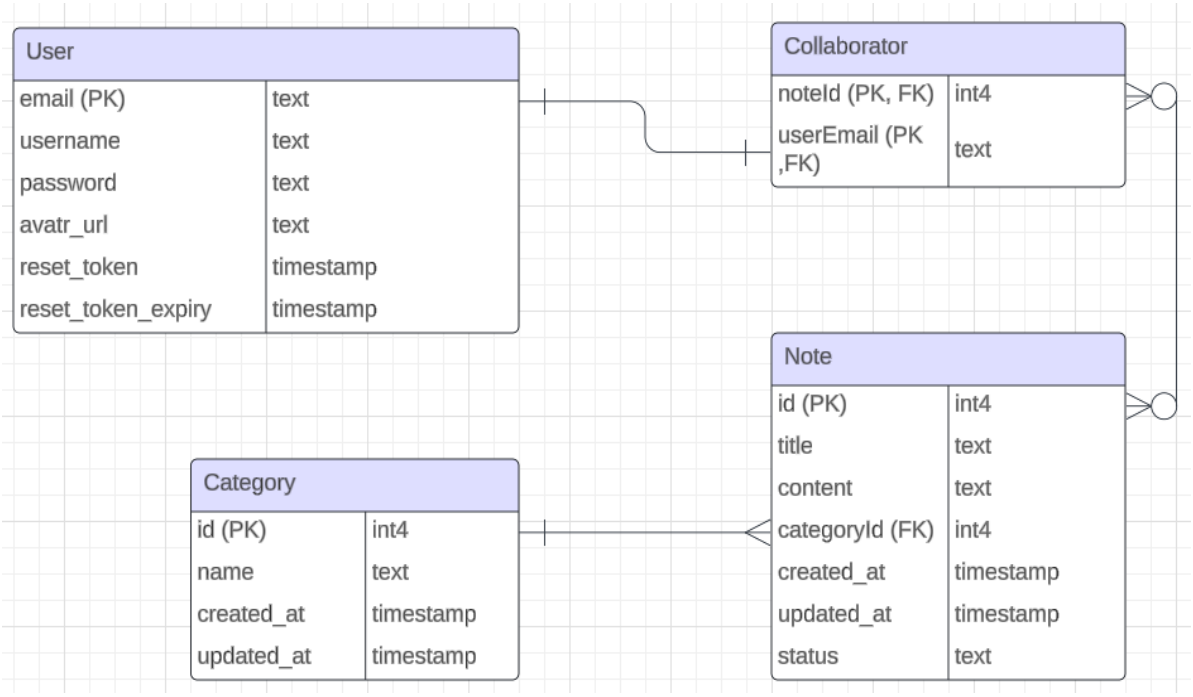
The Collaborator entity acts as a link between User and Note, it holds two foreign keys: noteId and userEmail, which reference the Note and User entities, respectively. This structure is essential for tracking which users have access to which notes.

A Note can have zero or many collaborators (indicating no one or many users collaborating on that note). A Collaborator is also linked to zero or many Notes, allowing users to be associated with multiple notes.

Only assigned collaborators have access to the note and have permissions to edit the note.

Category

The Category entity provides a way to group and organize notes. It consists of a unique id and a name field to describe the category. The one-to-many relationship between Category and Note ensures that each note belongs to a single category, while a category can encompass multiple notes. Categories help users to search through their content by topic.



Operating environment and major dependencies

Hosting

The app is deployed on Heroku, a cloud platform that automates deployment and scaling. The frontend and backend are managed separately, each with its own dependencies (handled by two separate package.json files). Heroku's environment variables handle secure configuration, and the app connects to Supabase and Amazon S3

Environment variables

The app uses environment variables in a .env file to store sensitive information like Supabase credentials, S3 bucket details, and WebSocket server URLs. This keeps secrets secure and allows for flexible configuration across different environments.

Frontend dependencies

The frontend uses React with Tailwind CSS for styling and WebSockets for real-time collaboration between users. We used npm and commands sucha as npm install to update frontend dependencies found in package.json

Backend dependencies

The backend runs on Node.js, which is used to create the RESTful API. It is also used to manage WebSocket communication. Supabase is used to help handle user authentication. Prisma was used to create an initial database schema. AWS (Amazon Web Services) SDK was used to set up Amazon S3 storage for user profile pictures.

Database

The app's data is stored in a PostgreSQL database managed by Supabase. Prisma was used to set up Object Relational Mapping (ORM). To stay updated with the latest model changes, we had to run npx prisma generate which is used to read the Prisma schema file and generate a Prisma client.

Authentication

User authentication is a critical aspect of the note-taking app, ensuring that only authorized users can access, edit, and collaborate on notes. Rather than using JSON Web Tokens (JWT) for authentication, the app utilizes **cookies** to manage user sessions.

Why Cookies Instead of JWT Tokens?

JWT tokens have an expiration date, meaning that once a token expires, the user must re-authenticate. For security reasons, all requests require authentication, so re-authenticating is inconvenient.

JWT tokens are stateless meaning that they must be included and validated with every request. This validation process can cause frequent token expiration and requires users to refresh their tokens.

cookies allow for stateful session management, where the server keeps track of the session state. By storing session information on the server, users can stay authenticated across multiple sessions without needing to constantly refresh tokens. This provides a more seamless user experience.

How Cookies Work in the App

When a user signs in, a session is created on the server and a cookie is sent to the user's browser. This cookie is then automatically included in all future requests, allowing the server to authenticate the user without requiring the client to handle tokens directly.

By storing the session information server-side, the app ensures that users remain authenticated over time, even after a session might expire on a client with a JWT token. This reduces the need for re-authentication, providing a smoother experience for the user.

Cookie Management

Cookies are essential to the app's authentication system. The cookies must be included in all requests made by the client, ensuring that the server can verify the user's session for each interaction. This allows the server to validate the session state and authorize actions like note creation, collaboration, and updates.

User Flow: Signup and Sign-In

The user authentication flow includes two main components: **sign up** and **sign in**.

- **Sign Up**: New users can register for an account by providing their email, username, and password. Once signed up, the user is automatically authenticated and issued a session cookie to maintain their login status.
- **Sign In**: Existing users can sign in to the app, and upon successful authentication, a session cookie is set in their browser. This cookie remains active across requests, enabling the user to remain logged in across sessions.

Once authenticated, users are redirected to the main dashboard where they can start collaborating on notes.

High-Level Description of Design Patterns for the Client and API

The note-taking app was built with a focus on usability, scalability, and reliability. The following design patterns were chosen to support these goals:

Client-Side Design

For the frontend we chose React as it allows components to be reused, and its state management helps keep data consistent. For example, the note card component is used for single user notes and collaborator notes. State management was very important for components such as dropdown lists and pop-up modals.

API Design

The backend uses a RESTful API, ensuring a clear separation between client-side and server-side logic. This provides a scalable interface for managing users, notes, and collaboration. Cookies manage authentication, providing persistent, server-side sessions and minimizing re-authentication. WebSockets enable instant updates for all connected users, keeping notes synchronized in real time.

The combination of a RESTful API and React's component-based design ensures flexibility and ease of maintenance.

Research References

- Web sockets API
  - MozDevNet, mozilla (2012) *The Websocket API (WebSockets) - web apis: MDN*, *MDN Web Docs*. Available at: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API (Accessed: 6 October 2024).
- AWS S3 Storage
  - Tutorials, T.T. (2021) *Amazon/AWS S3 (Simple Storage Service) Basics | S3 Tutorial, Creating a Bucket | AWS for Beginners*, *YouTube*. Available at: https://www.youtube.com/watch?v=mDRoyPFJvlU (Accessed: 4 October 2024).
- Supabase Tutorial
  - Duhan, G. (2023) *Learn Supabase (Firebase Alternative) – Full Tutorial for Beginners*, *YouTube*. Available at: https://www.youtube.com/watch?v=dU7GwCOgvNY (Accessed: 22 September 2024).
- React:tutorials:
  - org, M. (2014) *Learn React*, *React Blog RSS*. Available at: https://react.dev/ (Accessed: 14 October 2024).
- Node.js::

- Hamedani, M. (2016) *Tutorial Review - node.js tutorial for beginners: Learn node in 1 hour*, *gitconnected*. Available at: https://gitconnected.com/learn/node-js/nodejs-tutorial-for-beginners-learn-node-in-1-hour-75776f (Accessed: 14 October 2024).