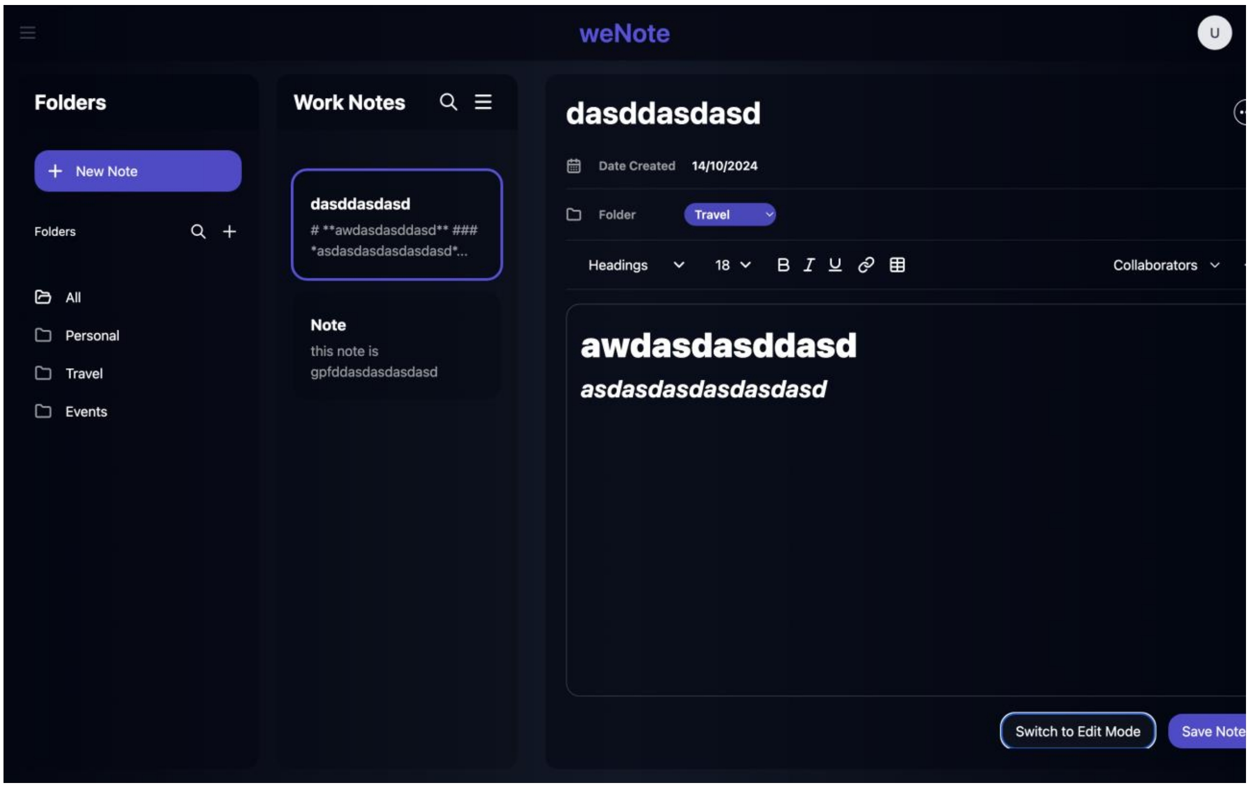
CS 343 Group 40 Report

DR de Goede 26451964 A du Plessis 25403389

JE Oosthuizen 26507404 GF Bosman 25020862

RS Bertschinger 22630945 LM de Vos 24764701



dasddasdasd
awdasdasddasd ### *asdasdasdasdasdasd*...

Note
this note is gpfddasdasdasdasd

dasddasdasd

Date Created 14/10/2024

Folder Travel

awdasdasddasd

asdasdasdasdasdasd

Table of Contents

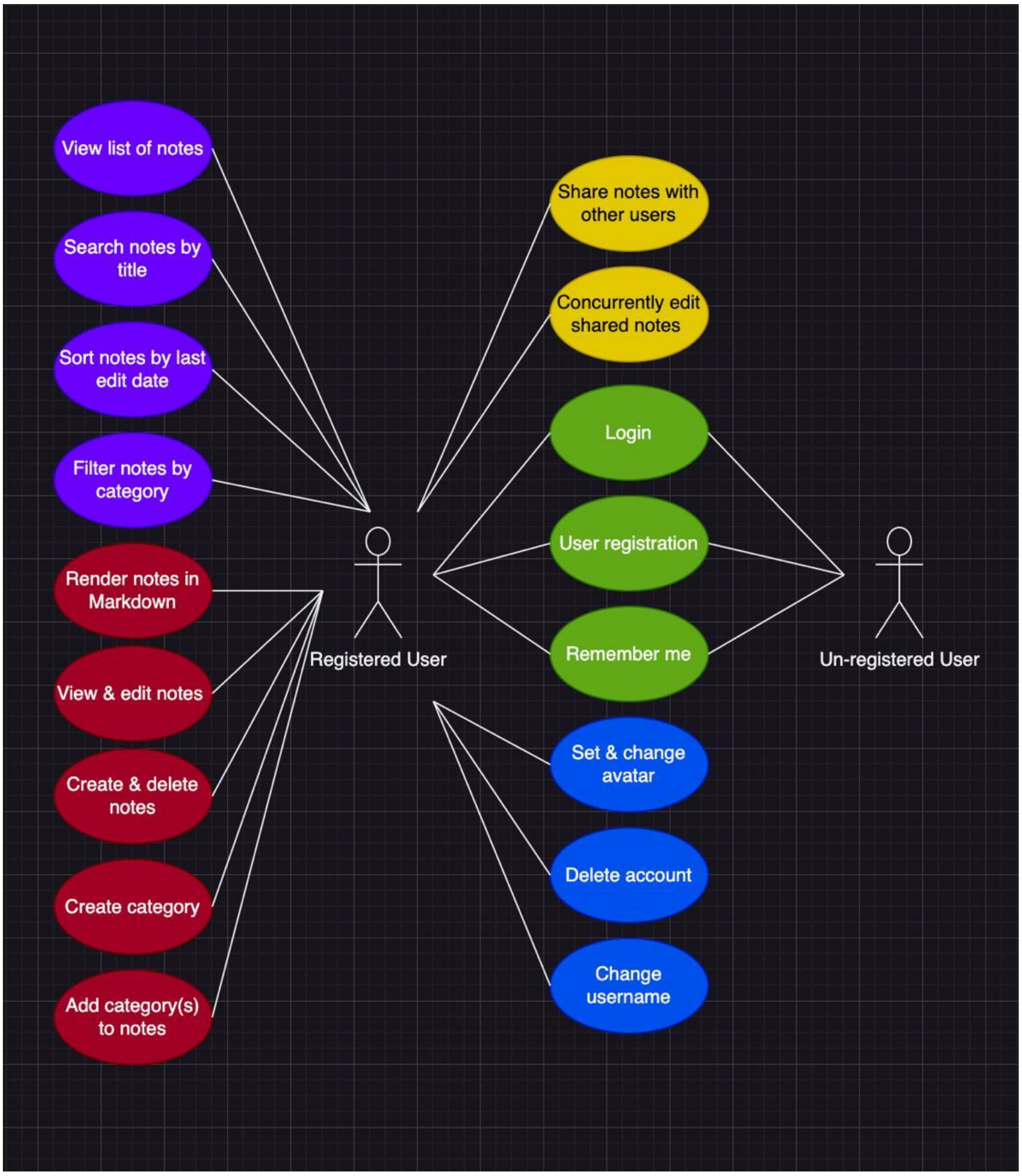
Introduction

This report presents the work done by Group 40 for the CS 343 project 2. The goal of this project was to create a platform where users can create, organize, and share notes with others, all while providing a user-friendly interface and secure authentication system. The users should also be able to work on notes together in realtime and render them to *Markdown* format.

Our app's frontend is built using React and styled with Tailwind CSS. On the backend, we use Express.js to manage API requests and server-side logic. For the database, we use Supabase, a service built on PostgreSQL.

In terms of user authentication, we've designed a system that ensures secure login and session management. This is done using bcrypt for password hashing and JSON Web Tokens (JWT) for handling session tokens. The authentication system ensures that only registered users can access and interact with the app.

Use Case Diagram



Data Modelling

ER diagram

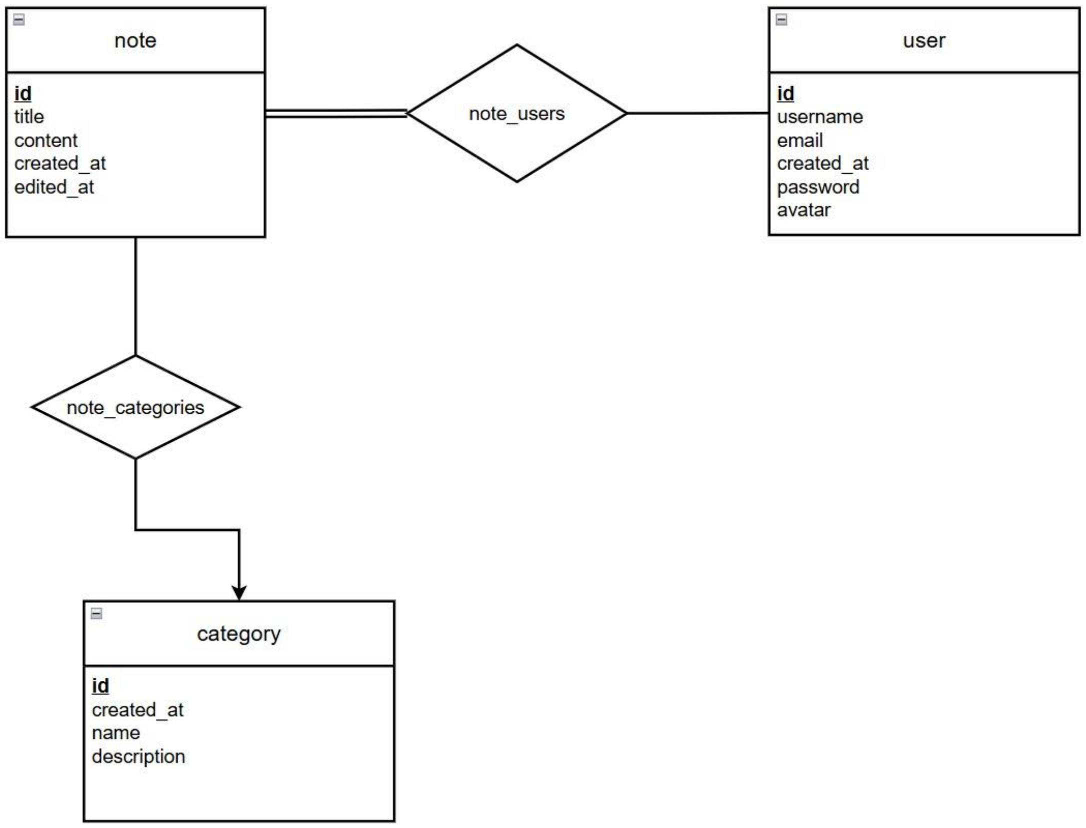
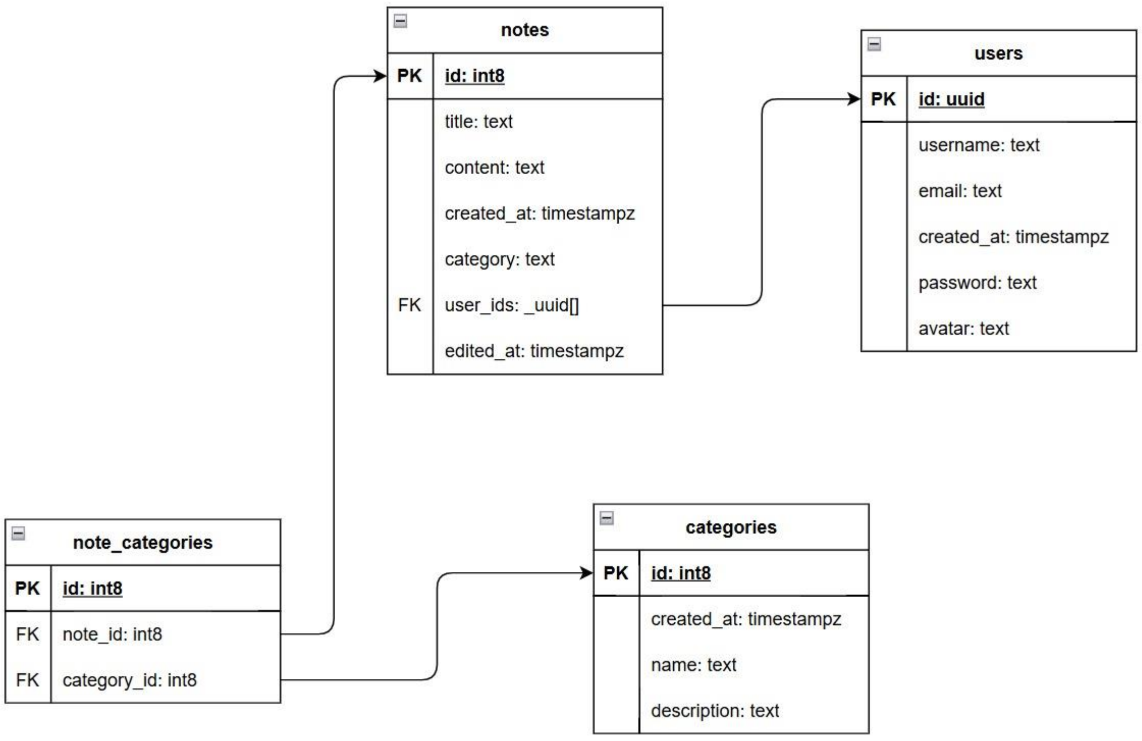


Table design



Operating Environment and Major Dependencies

Operating environment:

We use an environment that utilizes React for building interactive user interfaces, alongside TypeScript for type safety and maintainability. The frontend is styled with Tailwind CSS, a utility-first CSS framework that allows for rapid design development. On the backend, we use Express.js, a lightweight Node.js framework, to handle API requests and routing. The database is managed through Supabase, a PostgreSQL-based service that offers real-time database updates. We use Auth.js for handling authentication and session management, ensuring secure access to our application.

1. **Authentication and Authorization**:
    a. @auth/core and @auth/express: For authentication and authorization, specifically credential-based authentication.
    b. jsonwebtoken: For handling JSON Web Tokens (JWT)
    c. bcrypt: For password hashing and salting.
2. **Backend & API**:
    a. express: Web framework used to create server-side logic, API endpoints, and handle routing.
    b. cors and cookie-parser: Middleware for handling cross-origin requests and cookies in express.
3. **Database**:
    a. prisma: For interacting with databases using Prisma ORM.
    b. @supabase/supabase-js: For database interactions.
4. **Frontend (React)**:
    a. react-toastify: For displaying toast notifications
    b. tailwindcss-animate and tailwind-merge: Utilities for using Tailwind CSS with animations.
    c. lucide-react: Icon package for React.

5. **Real-time Communication**:
    a. socket.io and socket.io-client: For real-time communication and handling WebSockets
6. **File Handling**:
    a. multer: Middleware for handling file uploads.
7. **Validation**:
    a. zod: A schema validation library used to validate data inputs like forms.
8. **Utilities**:
    a. dotenv: For loading environment variables from .env files.
    b. uuid: For generating unique identifiers.

9. **Build-tools:**
    a. Vite
10. Package manager:

a. pnpm

Authentication

The authentication system in the application is built using bcrypt for password hashing, JSON Web Tokens (JWT) for session management, and Supabase as the database provider.

1. User Registration

When a new user registers, the system:

- Checks if a user with the same email already exists in the Supabase database.
- If no such user exists, it generates a unique user ID (`uuidv4`) and hashes the user's password using bcrypt with a salt factor of 10.
- The new user's details (ID, email, username, and hashed password) are then inserted into the Supabase `users` table.
- If the registration is successful, the system returns a success message, otherwise, an error message is sent back.

2. User Login

When a user logs in, the system:

- Retrieves the user's data by searching for their email in the database.
- It verifies the user's password by comparing the input password with the hashed password stored in the database using bcrypt.
- The system generates a JWT with a 30-day expiration date, since that is the maximum time a cookie can persist (with 'RememberMe' selected), and it's wrapped inside the cookie anyway so it will be rendered invalid when the cookie expires. The cookie acts as a session-cookie if 'rememberMe' is not selected, and persists for 30-days if it is selected.
- The generated JWT contains the user's ID and is returned to the user upon a successful login.

3. User Logout

To log a user out:

- The system clears the `access_token` cookie, effectively ending the user's session.

4. JWT Verification Middleware

For protected routes:

- A middleware function verifies the JWT stored in the `access_token` cookie.
- If the token is valid, the user's ID is attached to the `req.user` object, allowing access to user-specific routes and actions.

- If the token is missing or invalid, the middleware responds with an "Unauthorized" or "Invalid token" message.

Security Features:

- **Password Hashing:** All passwords are securely hashed using bcrypt, preventing plain-text passwords from being stored.
- **JWT Authentication:** Tokens are signed using a secret key, allowing stateless authentication and maintaining session integrity.
- **Token Expiry:** The use of token expiration (with conditional `rememberMe`) ensures that sessions are limited in time, improving security by reducing the risk of token misuse.

High-level Design Description

API endpoints

/notes

- (GET) (credentials required)
    - Get notes associated with the logged in user .
- (POST) (credentials required)
    - Create a new blank note.
- /:noteId (PUT)(credentials required)
    - Update the information of a note denoted by its unique id.
- /:noteId/share (POST) (credentials required)
    - Share the note with id noteId with a user specified in the body of the request.
- /:noteId/collaborators (GET) (credentials required)
    - Get a list of collaborators associated with the note with id noteId.
- /:noteId (DELETE) (credentials required)
    - Delete the note with id noteId.

/categories

- (GET) (credentials required)
    - Get a list of all categories
- (POST) (credentials required)
    - Create a new category.
- /:categoryId (GET) (credentials required)
    - Get information associated with the category with id categoryId.
- /:categoryId (PUT) (credentials required)
    - Update the information associated with the category with id categoryId.
- /:categoryId (DELETE) (credentials required)
    - Delete the category with id categoryId.

/notecat

- (POST) (credentials required)
    - Create a new note- category relation.
- /note/:noteId/category/:categoryId (GET) (credentials required)
    - Get the information associated with the relation between the note with id noteId and the category with id categoryId.
- /note/:noteId (GET) (credentials required)
    - Get the relations associated with the note with id noteId.
- /category/:categoryId (GET) (credentials required)
    - Get the relations associated with the category with id categoriyId.
- /:id (DELETE) (credentials required)
    - Delete the relation associated with the unique id, id.

/users

- /:userId (GET) (credentials required)

    - Get the information associated with the user with id userId.
- /:userId (PUT) (credentials required)
    - Update the information associated with the user with id, userId.
- /:userId (DELETE) (credentials required)
    - Delete the user account with the id, userId.
- /email/:email (GET) (credentials required)
    - Get the information of the user associated with the provided email address.
- /me (GET) (credentials required)
    - Get the information associated with the logged in user.