Report

Co-Scribe - A Collaborative Note-Taking Web App

Group 27 Report

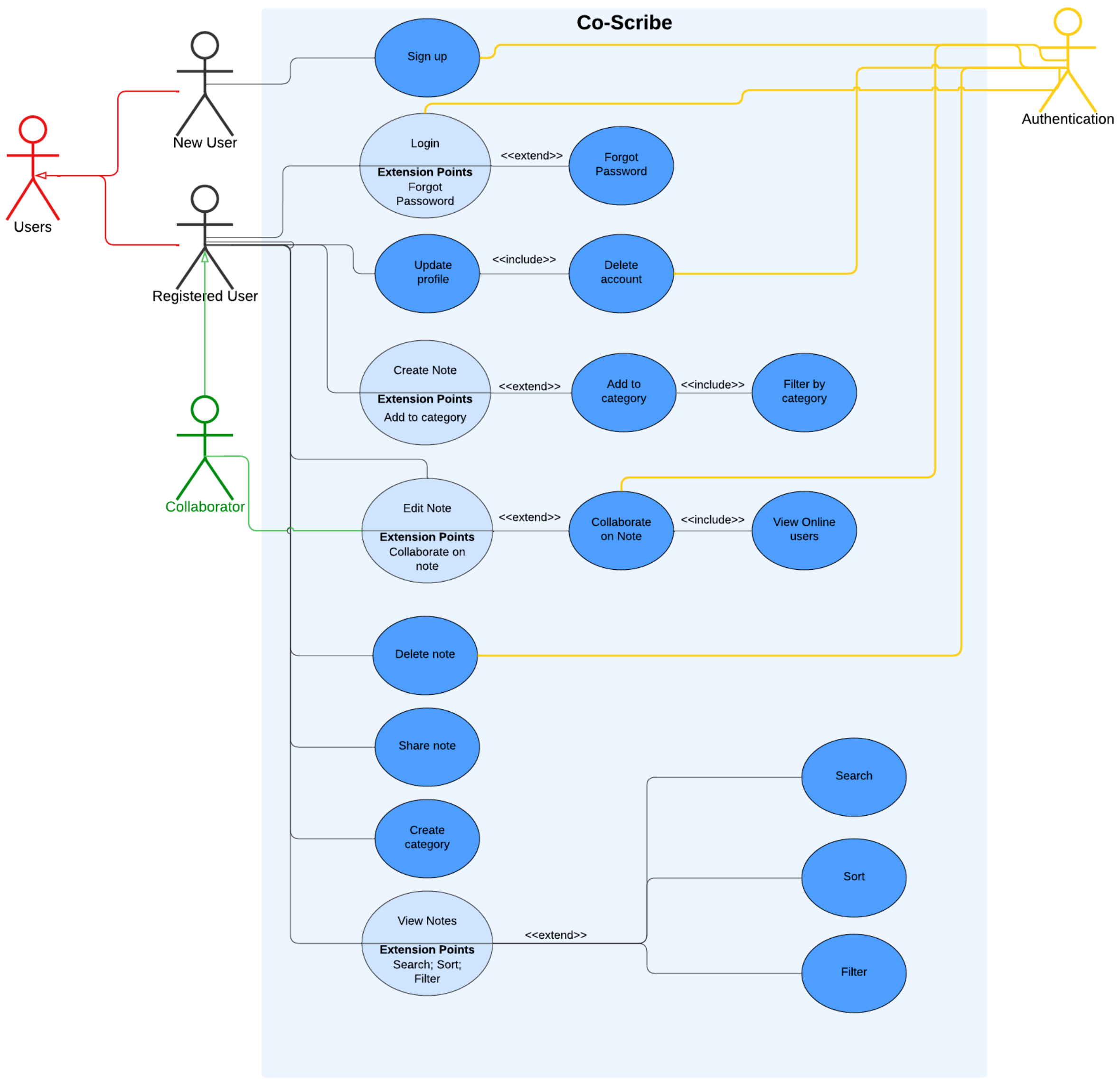
Team Members

- Amir Patel (26034670)
- Britton Petersen (25952277)
- Kiara Moodley (24754919)
- Su'ad Price (26476878)
- Sanidhya Soni (26069776)

Introduction

Our team developed a collaborative note-taking web application as part of our CS 343 course. The application allows users to create, edit, and share notes in real-time, supporting markdown formatting. We've implemented a React.js user interface with Tailwind CSS for styling. We used an Express API backend server using TypeScript, and Supabase for our PostgreSQL database.

Use Case Diagram



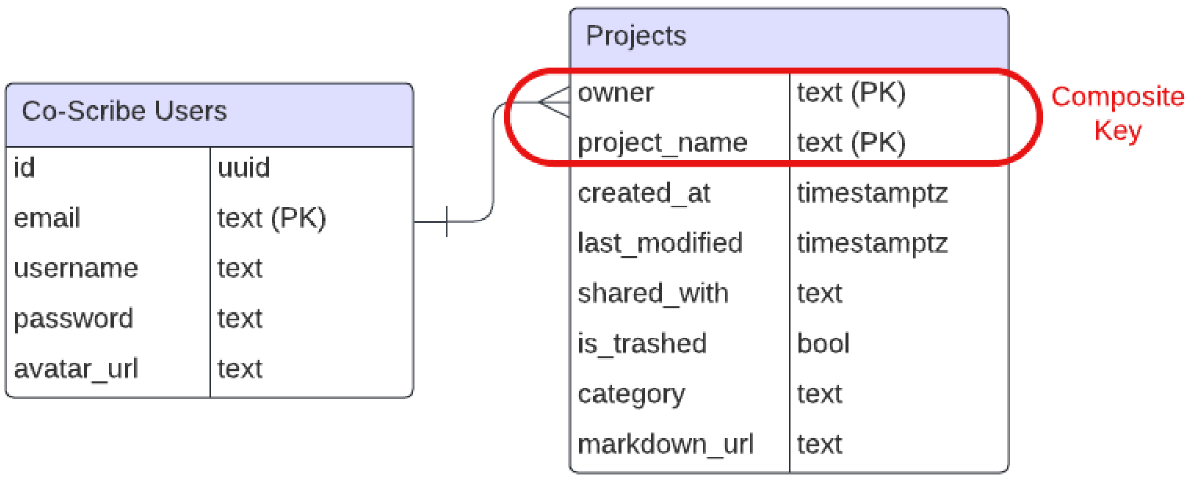
Data Modeling

Our database is normalized to 3NF and consists of the following tables:

1. **Co-Scribe Users**: This table stores a list of user accounts that have signed up to Co-Scribe. It manages the following fields:
    - **id**: The unique user id that is used to create an authentication token.
    - **email**: Unique email of the user which is crucial to login and authentication. This is the primary key.
    - **password**: Hashed and salted password. There were 10 round of salting (number of iterations of hash function), while making use of a Blowfish cipher.
    - **username**: Unique username used for display purposes on the GUI.
    - **avatar_url**: The link to the user's profile picture. This is used for retrieving the profile picture in Supabase Storage.
2. **Projects**: This table stores a list of projects currently in the database.
    - **owner**: This field forms a part of a composite primary key. It stores the email of the user that owns the project.
    - **project_name**: This field forms a part of a composite primary key. It stores the name of the project. Togther with the owner, it uniquely identifies a project as an owner cannot have multiple

projects with the same name.
- **created_at**: This field indicates when the project was created.
- **last_modified**: This field indicates when the project was last edited by a user.
- **shared_with**: This field indicates the emails of the people with which the project is shared.
- **is_trashed**: This field indicates if the project was placed in the trash by the owner (not deleted yet).
- **category**: This field indicates the category/tag to which the project belongs.
- **markdown_url**: This field indicates the link to the project file which is stored on Cloudinary. This is used to retrieve the contents of the markdown file.

Database Schema



Co-Scribe Users	
id	uuid
email	text (PK)
username	text
password	text
avatar_url	text

Projects	
owner	text (PK)
project_name	text (PK)
created_at	timestamptz
last_modified	timestamptz
shared_with	text
is_trashed	bool
category	text
markdown_url	text

Normalisation

We normalised our database to the Third Normal Form by ensuring the following steps hold:

- **1NF**:
  - Tables have primary keys and atomic values.
  - No duplicated values.
- **2NF**:
  - The database is in 1NF
  - There are no partial dependencies (non-key attributes being dependent on a primary key)
- **3NF**:
  - The database is in 2NF
  - There are no transitive dependencies (non-key attributes depending on other non-key attributes)

Storage of Avatars and Notes

- **Avatars**: User profile pictures (avatars) are stored in Supabase storage. This enables secure and efficient management of image files, with links to these files saved in the 'Co-Scribe Users' table (avatar_url). By storing avatars in Supabase, the app can easily manage, access, and display user images without taking up database storage space.
- **Notes**: Notes are stored in Cloudinary, a cloud-based media management service. This allows users to upload, store, and manage various file types, including text and images. By leveraging Cloudinary, the app can efficiently handle large files and media content, ensuring quick retrieval and secure storage. The URLs for these notes are stored in the 'Projects' table (markdown_url), facilitating easy access and collaboration.

Operating Environment and Major Dependencies

Frontend

- **React.js**: The frontend of the app is built using React.js, a JavaScript library for building user interfaces. React allows the creation of reusable UI components, making the development process efficient and maintainable. This ensures that the app has a dynamic and responsive interface, allowing users to interact with their notes in real-time.
- **Tailwind CSS**: Tailwind CSS is used for styling the application. It provides utility-first classes that help in building custom designs without actually writing extensive CSS. This makes the styling process faster and easier while also allowing a consistent design across the app.
- **SimpleMD**: To handle markdown rendering, the app utilizes SimpleMD, which allows users to write notes in markdown format and see a live preview. This feature enhances the user experience by providing a familiar and flexible note-taking environment.

Backend Structure

The backend is structured as a RESTful API, developed using Express.js. This approach ensures a clear and modular way of handling different endpoints, where each endpoint corresponds to a specific functionality within the app. With a RESTful design, the backend can easily scale to accommodate more features or handle increased traffic. Each endpoint can be independently maintained as well as tested, making it simpler to expand or modify specific parts of the application as needed. The use of TypeScript adds type safety, which helps catch errors during development and improves code readability. It also enhances the code development process.

- **Express.js**: The core web application framework for handling HTTP requests and responses.
- **Supabase**: We're using Supabase as our backend-as-a-service platform, which provides a managed PostgreSQL database and other features.
- **CORS**: Cross-Origin Resource Sharing is enabled to allow requests from the frontend application.
- **Cloudinary**: Used for storing note content.
- **Mailgun**: Integrated for sending email notifications for note-sharing updates.
- **Multer**: Middleware used for handling file uploads, enabling users to upload files as part of their notes.
- **Password Hashing**: bcrypt is used for securely hashing and comparing passwords.
- **Swagger**: Used for API documentation, allowing us to provide descriptions for the available endpoints, parameters, and responses easily

Database

- **PostgreSQL (managed by Supabase)**: PostgreSQL is the database management system for this Co-Scribe, due to its robustness, scalability, and advanced features. Supabase, a managed service for PostgreSQL, is used to simplify database management. This ensures data persistence, security, and efficient data handling.

Real-time Collaboration

- **WebSockets (Socket.io)**: For real-time, bidirectional note collaboration on projects, WebSockets are utilized through the Socket.io library. This allows multiple users to edit and view notes simultaneously, with changes appearing live across all connected clients.

Authentication

JWT-based Authentication System

Co-Scribe implements a secure authentication system using JSON Web Tokens (JWT). This stateless authentication mechanism allows us to verify the identity of users without storing session information on the server.

Token Generation and Storage

When a user successfully logs in, our system generates a JWT containing the user's ID and other non-sensitive information. The token is signed using a secret key known only to the server, ensuring its integrity. The generated token is then sent back to the client and stored based on the user's choice:

1. Session Storage (Default):
   - Used when "Remember Me" is not selected
   - Token expiration: 1 hour
   - Cleared when the browser tab is closed or upon logout.
2. Local Storage (Remember Me):
   - Used when "Remember Me" is selected
   - Token expiration: 30 days
   - Persists even when the browser is closed
   - Only clears upon logout.

Token Verification

On each subsequent request to protected routes, the client includes the token in the Authorization header. Our server-side middleware verifies the token.

Token Renewal

To enhance security while maintaining user convenience, we implement a token renewal strategy:

1. For session storage tokens/remember me (1 hour expiration):
   - We renew the token on each API call if the token is close to expiration (e.g., less than 10 minutes left)
2. For local storage tokens (30 day expiration):
   - We renew the token daily when the user visits the site
   - If the token is older than 30 days, we force a re-login

Authentication (Authn) vs Authorization (Authz)

By clearly separating authentication and authorization, we can ensure that Co-Scribe not only verifies user identities but also properly controls access to resources, maintaining both security and appropriate user permissions throughout the system.

Authentication (Authn)

Authentication is the process of verifying the identity of a user. In our app, this involves:

1. User login with username/email and password
2. Verifying credentials against stored (hashed) passwords
3. Generating and providing a JWT token upon successful verification

Goal of authentication is to answer the question: "Who are you?"

Authorization (Authz)

Authorization is the process of determining whether an authenticated user has permission to access a specific resource or perform a particular action. In our app, this involves:

1. Checking and verifying the user's role or permissions (stored in the JWT or fetched from the database)
2. Allowing or denying access to certain API endpoints or app features based on these permissions.

Goal of authorization is to answer the question: "What are you allowed to do?"

Implementation in Co-Scribe

1. Authentication:
   - Handled by the login process and JWT generation
2. Authorization:
   - Implemented through role-based access control
   - Checked in route handlers after authentication

High-Level Description of Design Patterns

Architecture

The application follows a standard client-server architecture.

Client-Side

- **Component-based architecture**: Frontend was designed using a modular, component-based approach using Reat. Each part of the user interface is encapsulated within independent React components, promoting reusability, maintainability, and scalability. For example, the project header component is encapsulated within the project component, hence promoting modularity.

API

- The API is designed to serve as the backend, providing secure and efficient access to various functionalities. Built using Express.js and TypeScript, it follows a RESTful architecture that ensures modularity, scalability, and ease of maintenance. Below are the core aspects of the API design:

Endpoints

The REST API defines various routes for managing notes, users, and collaboration features. For example:

- **User Management**: CRUD operations for user registration, login, and profile management.
- **Note Operations**: CRUD operations for notes, allowing users to create new notes, get existing ones, make edits, and delete them.
- **Collaboration Features**: Endpoints to manage permissions, share notes with other users, and handle user roles within collaborative projects.

Authentication

- The API implements secure authentication using JWTs and each request to the endopoint must contain a valid token in its header.
- Middleware handles token verification, to improve security.

Integration with Additional Services

- **Cloudinary**: this is integrated for storing and retrieving note content through endpoints.
- **Mailgun**: This is integrated to allow sending of email notifications to reset passwords and for note-sharing notifications.

Middleware

The middleware functions are responsible for setting up and maintaining various aspects of our application.

- **CORS**: Cross Origin Resource Sharing allows the backend to handle requests from different origins. In our case, the frontend is running on a different port and CORS allows us to handle requests from

it.
- **Authentication**: Verifies JWT tokens to ensure secure access to protected routes.

Additional Features Implemented

From the project specification:

- Hosting the application on a public-facing server. At (http://159.203.189.208/login)
- Browser-based notifications
- Resetting of lost passwords with a verification code sent to the user's registered email address

Not from the project specification:

- Soft delete of notes with a restore option
- Filtering notes by your notes and shared notes
- The ability for a user to view their profile image
- Loading screen animation
- A different layout of components for mobile view further enhancing responsiveness
- Notes can be downloaded as a PDF

Individual Contributions

- Amir Patel (26034670): Implemented markdown compilation with front end integration.
- Britton Petersen (25952277): Implemented the backend with integration with frontend and managed the git structure of our project.
- Kiara Moodley (24754919): Implemented the frontend with integration with backend and contributed to the backend API.
- Su'ad Price (26476878): Implemented the user inteface with responsiveness.
- Sanidhya Soni (26069776): Implemented backend/API.

Conclusion

Co-Scribe represents a well-integrated, feature-rich platform designed to enhance productivity through real-time collaboration. By using a modern tech stack, the application ensures scalability, efficiency, and a seamless user experience. We were able to acquire vital industry knowledge with regards to web applications and build a strong foundation for future applications.