FUSION

COMPUTER SCIENCE 343

PROJECT 2 REPORT

Jevon, M, Mr [24744417]	Documentation
Steyn, AH Mr [24713848]	Database
Pheiffer, JG, Mr [24718076]	Backend
Van Wyk, SL, Mr [24691550]	Backend
Solomon, C, Mr [25201247]	Frontend
Erasmus, D, Mr [24697397]	Frontend

Table of Contents

Introduction

Background

In the digital age, with the rise of remote work and study, the need for effective and seamless collaborative tools has surged. Note-taking, an integral part of collaborative efforts, especially in academic and research-oriented environments, becomes significantly more efficient and organized when digitized and offered as a collaborative platform.

Purpose of the Project

The collaborative markdown note-taking web application (Fusion) aims to provide a robust, user-friendly platform that allows users to create, edit, share, and collaborate on notes in real-time, enhancing the cooperative experience. Using technologies like React.js, Tailwind CSS, Node.js, Express/GraphQL, PostgreSQL, WebSocket, and Marked, the application not only allows for the streamlined creation and management of notes but also ensures live, synchronized collaboration among multiple users.

Objectives

- Collaborative Real-time Editing
- User Management
- Note Management
- Markdown Functionality

Scope

With a focus on real-time collaboration, user management, and note management, the application serves as a tool suitable for individuals and teams looking to collaborate on notes, documentation, and general content creation in an organized and efficient manner. The application intends to cater to academic groups, research teams, and any collaborative environments where note-sharing and joint content creation are prevalent.

Use Cases

Use Case Diagram

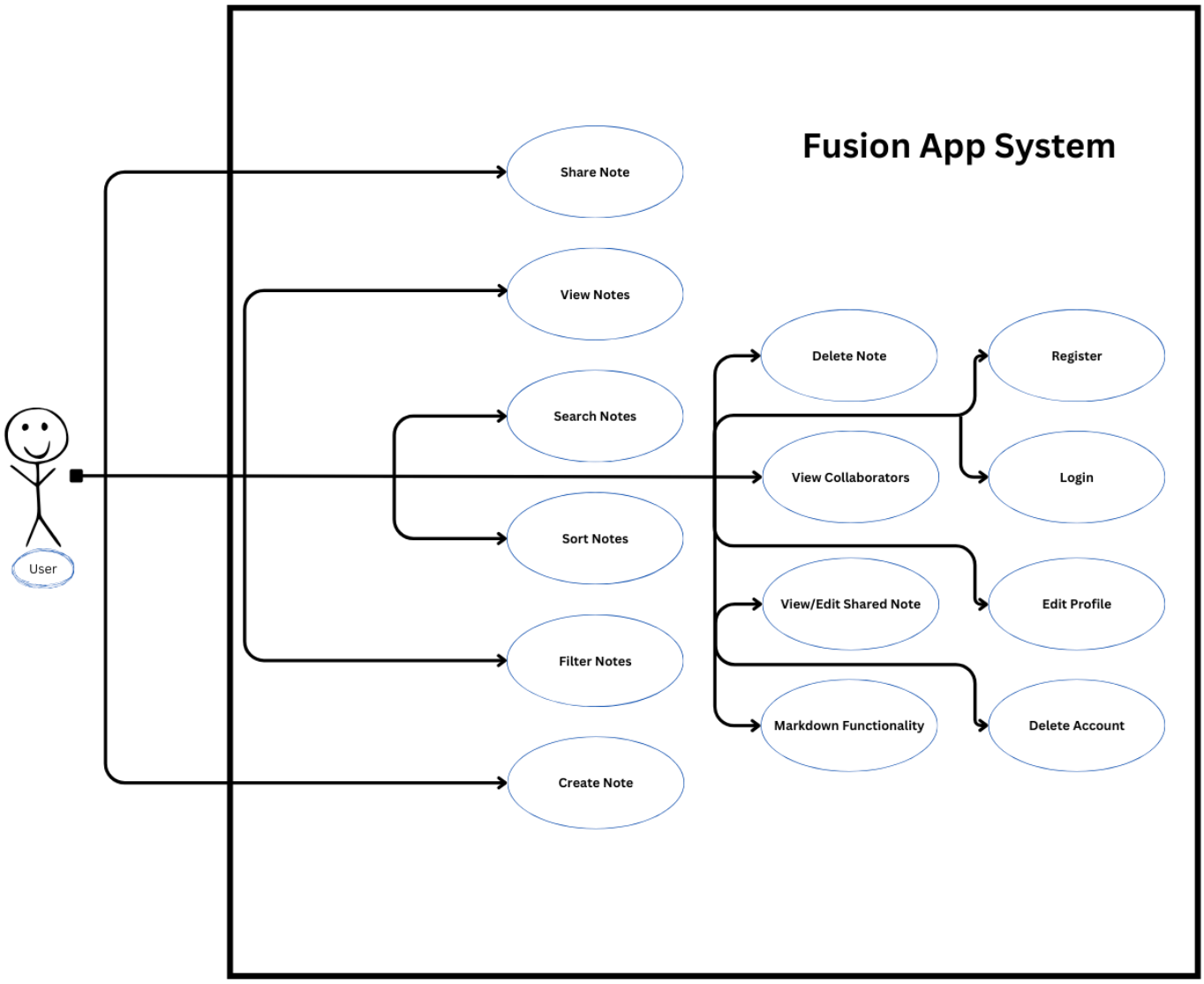

Figure 1: Use Cases

Use Case Descriptions

Actors:

1. **User**
2. **System**

Use Cases:

1. **Register:** Users can create an account providing a unique username, unique email, password, and avatar.
2. **Login:** Users can log in using their credentials and have a "remember me" option.
3. **View Notes:** Users can view a list of notes.
4. **Search Notes:** Users can search notes by title.
5. **Sort Notes:** Users can sort notes by time last edited (most recent first).
6. **Filter Notes:** Users can filter notes by category.
7. **Edit Profile:** Users can edit their profile details.
8. **Delete Account:** Users can delete their account and all associated notes.
9. **Create Note:** Users can create a new note.
10. **Delete Note:** Users can delete a note.
11. **Markdown Functionality:** Users can view and edit notes using markdown.
12. **Share Note:** Users can share notes.
13. **View/Edit Shared Note:** Users can view and edit notes shared with them.
14. **View Collaborators:** Users can view other users currently editing the note.

Data Modeling

Identified Entities and Attributes

User

- user_id (PK) Int
- username (Unique) VarChar
- email (Unique) VarChar
- password_hash VarChar

Category

- category_id (PK) Int
- name VarChar
- user (FK) Text

Note

- note_id (PK) Int
- title VarChar
- content Text
- last_edited DateTime
- user_id (FK) Int
- category_id (FK) Int

Shared_Notes

- shared_note_id (PK) Int
- note_id (FK) Int
- shared_with_user_id (FK) Int

Relationships

- **User-Note Relationship:** One-to-many from User to Note.
- **Category-Note Relationship:** One-to-many from Category to Note.
- **User-Shared_Notes Relationship:** Many-to-many between User and Note via Shared_Notes.

ER Diagram

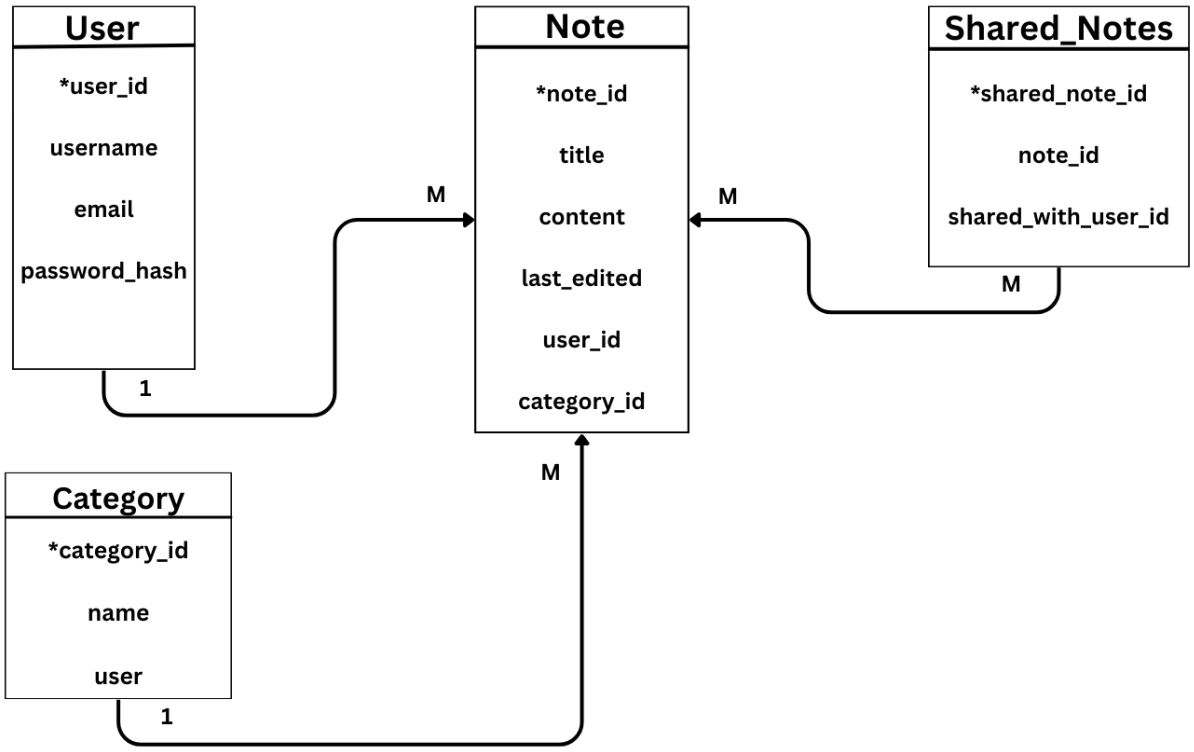

Figure 2: ER Diagram

Operating Environment

Technology Stack

- **Frontend:**
    - **React.js:** Utilized for building the user interface of the application.
    - **Tailwind CSS:** Employed for styling the user interface.
    - **WebSocket:** Facilitates real-time collaboration on shared notes.
- **Backend:**
    - **Node.js:** Acts as the backend runtime environment.
    - **Express/GraphQL API:** Manages API requests and data retrieval/transmission.
- **Database:**
    - **PostgreSQL:** Stores and manages the application data, offering advanced data types and robustness.
- **Markdown Rendering:**
    - **Marked:** A markdown parser and compiler used for rendering markdown content.

Development Environment

- **Version Control:**
    - **Git:** Used for version control to track changes and manage collaborative work on the project.
- **Development Tools:**
    - **Git Board:** Utilized for managing issues and dividing workload.
- **Dependency Management:**
    - **npm (Node Package Manager):** Manages packages and dependencies used in the project.

Deployment Environment

- **Database Hosting:**
    - **Microsoft Azure Flexible PostgreSQL:** Development server with 32GB storage and 2GB RAM to host the PostgreSQL database, offering a balance of scalability, flexibility, and manageability.

Security and Authorization

- **Authentication:**
    - **JWT Authentication:** JWT tokens are used for authenticating users. Tokens are generated for sessions and user credentials are verified before a token is issued.
    - **Password Hashing:** Passwords are hashed before they are stored in the database, enhancing security by ensuring actual passwords are not stored.
- **Authorization:**
    - Tokens are verified to ensure that requests are made from authenticated sessions.
- **Data Security:**
    - Utilizing GraphQL to validate data on the server-side before it's processed or stored prevents SQL injections.

Collaboration and Real-Time Update Mechanism

- WebSockets are implemented to allow real-time bidirectional event-based communication between clients and the server.

Design Patterns

In the development of the Fusion app, the application of design patterns can facilitate a clean, efficient, and scalable codebase.

Client-Side (React.js with Tailwind CSS)

- **Component Pattern:**
    - **Rationale:** React itself is built around the component pattern, encouraging the development of reusable, independent components that manage their own state and rendering.
    - **Application:** UI elements like buttons, input fields, notes, and models are implemented as individual components, promoting reusability and maintainability.
- **Container Component Pattern:**
    - **Rationale:** To separate concerns of data handling and UI rendering, ensuring cleaner and more maintainable code.
    - **Application:** Implement container components that manage data fetching, state, and logic, and pass data to presentational components via props.

API (Node.js with Express/GraphQL)

- **Model-View-Controller Pattern:**
    - **Rationale:** To separate the concerns of data, presentation, and control logic, providing a systematic and organized approach to code structure.
    - **Application:**
        - **Model:** Manages data and business rules, interacting with the database.
        - **View:** Generates output to the client based on the data from the models.
        - **Controller:** Handles input from the client, interacting with the model and view.
- **Repository Pattern:**
    - **Rationale:** To separate the logic that retrieves data from the database from the business logic of the application.
    - **Application:** Implement repositories that handle CRUD operations for the entities, isolating the database access code from the business logic, and providing a substitution point for unit tests.

- **Singleton Pattern:**
    - **Rationale:** To control access to some shared resources, like configuration, connection pools, or logging services.
    - **Application:** Ensure that a class has only one instance and provides a global point of access to that instance, like managing a single database connection shared among different parts of the application.

Real-Time Collaboration (WebSocket)

- **Observer Pattern:**
    - **Rationale:** To allow multiple clients to react to changes in shared notes without polling the server for updates.
    - **Application:** Implement WebSocket as an observer that notifies all connected clients about changes in the shared note, allowing real-time updates.
- **Strategy Pattern:**
    - **Rationale:** To define a family of algorithms for real-time collaboration features, making them interchangeable.
    - **Application:** Define collaboration strategies that the WebSocket server can use to manage concurrent edits to a note, ensuring consistency across all clients.