Note Web App - High-Level Design

1. High-Level Design

Main Components and Logical Interactions

Components

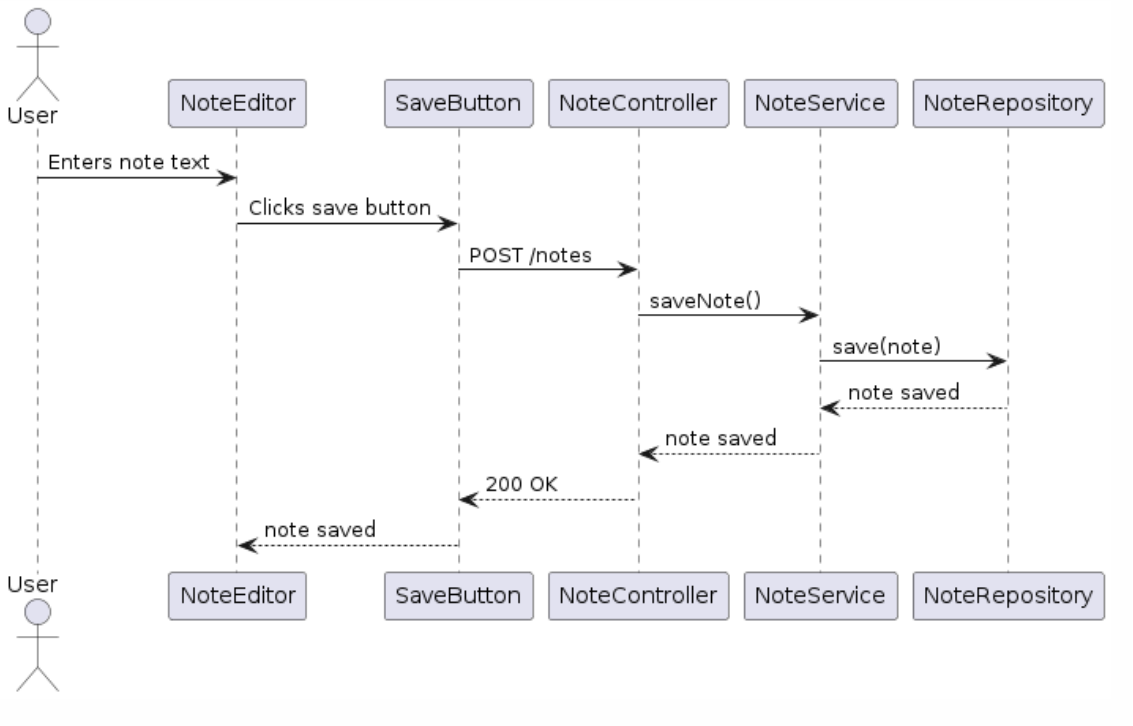
1. **Front-end (UI):**
    - **NoteEditor:** Component to create and edit notes.
    - **NoteList:** Component to list saved notes.
    - **NoteItem:** Component to display an individual note in the list.
    - **DeleteButton:** Component to delete a note.
    - **SaveButton:** Component to save a note.
2. **Back-end (API):**
    - **NoteController:** Controller to handle HTTP requests related to notes.
    - **NoteService:** Service containing business logic for note operations.
    - **NoteRepository:** Repository interacting with the database for CRUD operations on notes.
3. **Database:**
    - **Notes:** Table to store user notes.
    - **Users:** Table to store user information (out of scope but necessary for authentication).

Logical Interaction Flow

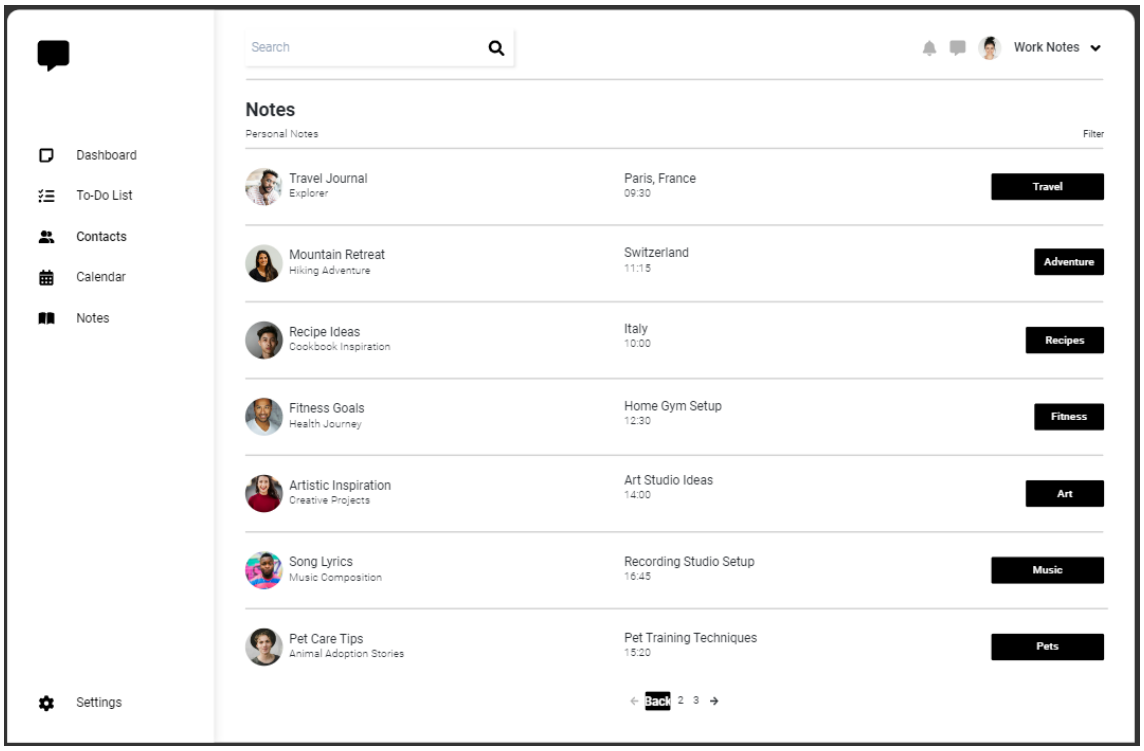
1. **Create/Edit Note:**
    - The user enters note text in the `NoteEditor`.
    - The user clicks the `SaveButton`.
    - The note is sent to the `NoteController` via a POST request.
    - The `NoteController` calls the `NoteService` to save the note.
    - The `NoteService` interacts with the `NoteRepository` to persist the note in the database.
    - The response is sent back to the front-end to confirm that the note has been saved.
2. **List Notes:**
    - The user accesses the note list page.
    - The `NoteList` makes a GET request to the `NoteController`.
    - The `NoteController` calls the `NoteService` to fetch the notes.
    - The `NoteService` interacts with the `NoteRepository` to get the notes from the database.
    - The notes are sent back to the front-end and displayed in the `NoteList`.
3. **Delete Note:**
    - The user clicks the `DeleteButton` next to a note.
    - The `DeleteButton` sends a DELETE request to the `NoteController`.
    - The `NoteController` calls the `NoteService` to delete the note.
    - The `NoteService` interacts with the `NoteRepository` to remove the note from the database.

- The response is sent back to the front-end to confirm that the note has been deleted.

Logical Interaction Diagram



2. Web App UI



UI Components

1. **NoteEditor:**
    - TextArea: Area for user to type the note.
    - SaveButton: Button to save the note.
    - DeleteButton: Button to delete the note (visible only when editing an existing note).
2. **NoteList:**
    - List of saved user notes.
3. **NoteItem:**
    - Displays the content of an individual note.
    - Includes a button to delete the note.

Validation

1. **NoteEditor Validation:**
    - Ensure the note text is not empty before allowing the user to save the note.

3. Data Model

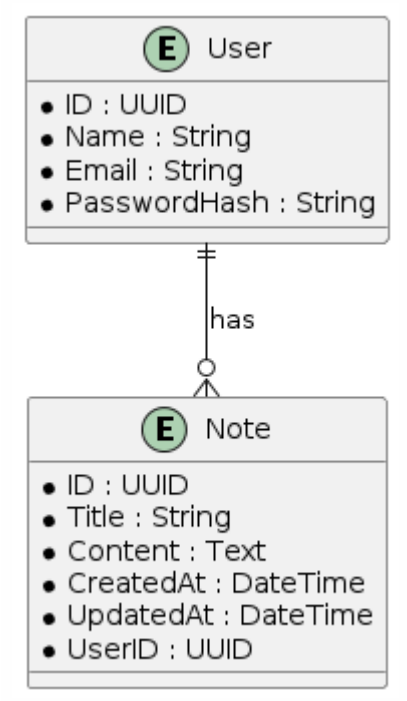
Required Properties for a Note

The note will be modeled with the following properties:

1. **ID** (unique identifier of the note)
2. **Title** (title of the note)
3. **Content** (content of the note)
4. **CreatedAt** (date and time of creation)
5. **UpdatedAt** (date and time of the last update)
6. **UserID** (identifier of the user who created the note)

Data Model in PlantUML

Here is the PlantUML code for the data model:



Explanation of the Data Model

1. **User**:
   - **ID**: Unique identifier of the user (UUID).
   - **Name**: Name of the user.
   - **Email**: Email address of the user.
   - **PasswordHash**: Hash of the user's password.
2. **Note**:
   - **ID**: Unique identifier of the note (UUID).
   - **Title**: Title of the note.
   - **Content**: Content of the note (can be long text).
   - **CreatedAt**: Date and time of creation of the note.
   - **UpdatedAt**: Date and time of the last update of the note.
   - **UserID**: Identifier of the user who created the note (relationship with the user).

Relationship

- A user can have multiple notes (`User ||--o{ Note : "has"`), but a note belongs to a single user.

4. Restful API

Endpoints and Verbs

1. **GET /notes**
    - Description: Returns all notes of the authenticated user.
    - HTTP Verb: GET
    - URL: `/notes`
    - Example Response:


```
[
  {
    "id": "1",
    "title": "First Note",
    "content": "This is the content of the first note.",
    "createdAt": "2023-01-01T00:00:00Z",
    "updatedAt": "2023-01-01T00:00:00Z",
    "userId": "123"
  },
  {
    "id": "2",
    "title": "Second Note",
    "content": "This is the content of the second note.",
    "createdAt": "2023-01-02T00:00:00Z",
    "updatedAt": "2023-01-02T00:00:00Z",
    "userId": "123"
  }
]
```


2. **POST /notes**

- Description: Creates a new note for the authenticated user.
- HTTP Verb: POST
- URL: `/notes`
- Example Request:

```
{
  "title": "New Note",
  "content": "This is the content of the new note."
}
```

Example Response:

```
{
  "id": "3",
  "title": "New Note",
  "content": "This is the content of the new note.",
  "createdAt": "2023-01-03T00:00:00Z",
  "updatedAt": "2023-01-03T00:00:00Z",
  "userId": "123"
}
```

3. **DELETE /notes/{id}**

- Description: Deletes a specific note of the authenticated user.
- HTTP Verb: DELETE
- URL: `/notes/{id}`
- Example Response:

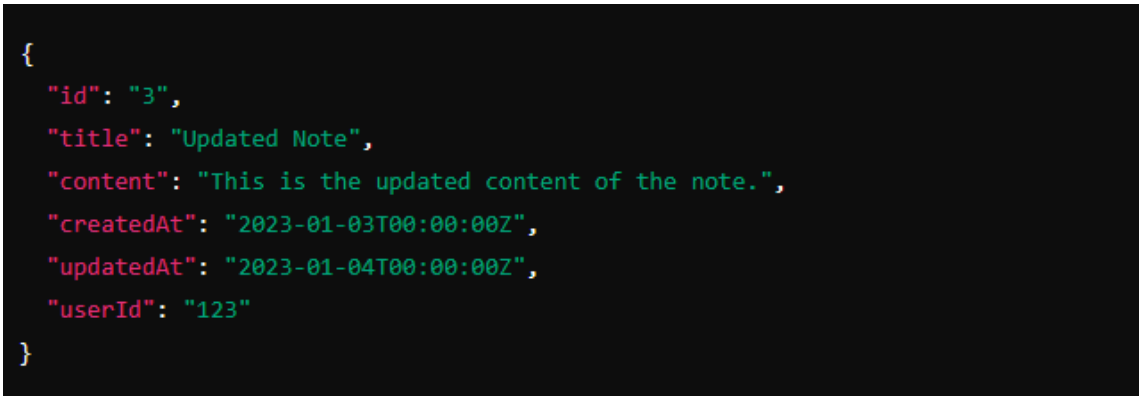
```
{
  "message": "Note deleted successfully."
}
```

4. **PUT /notes/{id}**

- Description: Updates an existing note of the authenticated user.
- HTTP Verb: PUT
- URL: `/notes/{id}`
- Example Request:

```
{
  "title": "Updated Note",
  "content": "This is the updated content of the note."
}
```

Example Response:



Description of Actions

1. **GET /notes**
    - **Action:** Retrieves all notes associated with the authenticated user.
    - **Business Logic:** Verifies user authentication and returns all notes from the database that belong to the user.
2. **POST /notes**
    - **Action:** Creates a new note.
    - **Business Logic:** Verifies user authentication, validates input (title and content), creates a new note in the database, and returns the created note.
3. **DELETE /notes/{id}**
    - **Action:** Deletes an existing note.
    - **Business Logic:** Verifies user authentication, validates the existence of the note and the user's permission to delete it, and removes the note from the database.
4. **PUT /notes/{id}**
    - **Action:** Updates an existing note.
    - **Business Logic:** Verifies user authentication, validates input (title and content), validates the existence of the note and the user's permission to update it, and updates the note in the database.

URLs and Verbs

- **GET /notes**
- **POST /notes**
- **DELETE /notes/{id}**
- **PUT /notes/{id}**

---

5. Web Server

Web Server Implementation

1. **Endpoint Configuration:**

    - Configure RESTful endpoints on the web server using a framework like Express (Node.js), Spring Boot (Java), or Flask (Python).
2. **Authentication and Session:**
    - Implement middleware to verify user authentication before allowing access to note endpoints.
    - Maintain user session using JWT tokens or cookie-based sessions.
3. **CRUD Actions:**
    - **GET /notes:** Implement logic to retrieve notes from the database associated with the authenticated user.
    - **POST /notes:** Implement logic to create a new note in the database, validating input and associating the note with the authenticated user.
    - **DELETE /notes/{id}:** Implement logic to delete an existing note, validating user permission and removing the note from the database.
    - **PUT /notes/{id}:** Implement logic to update an existing note, validating input and user permission, and updating the note in the database.
4. **Business Logic:**
    - Implement business logic in the service layer, which will be called by the controller layer.
    - Ensure all database operations are performed transactionally to maintain data consistency.
5. **Persistence:**
    - Use an ORM (Object-Relational Mapping) like Sequelize (Node.js), Hibernate (Java), or SQLAlchemy (Python) to interact with the database.
    - Configure data persistence in the database and ensure all CRUD operations are supported.