1. High-Level Design

Frontend:

The application's UI will consist of a blank area to display the user's notes in a grid format and a floating action button on the bottom right corner containing a plus icon, indicating the action of adding a new note.

Once the button is clicked, a modal will be displayed, containing a title "New Note", a text area below it and centered in the modal and two buttons aligned to the right bottom side, being the "Save" and "Cancel" buttons. The user's note will consist of a rectangular view, with centralized text (truncated with a "more" button for large texts) and a small X icon on the right top corner of it, indicating the action of deleting it.

On a mobile screen, the app would display the notes in a vertical list format, the fab icon and the modal could be maintained, just adjusting it's size so it can fit properly in a smaller resolution.

On the technical side, the app will be built using NextJs, as it is one of the more reliable React based frameworks. Since the user must be authenticated to perform actions in the app, the local storage could be used to store the token that will be required by the backend to validate the user's auth.

Backend:

The backend will consist of three routes designed for the required actions, a **GET** route with the desired page as param to retrieve the user's saved notes, a **DELETE** route with an ID as param to delete any of the saved notes and a **POST** route with a param called "content" to save a new note.

The project architecture will have it's API logic built in the controllers layer, the business logic in it's service layer and the persistence logic done by the repository, de data representation and handling will be done via DAO's and DTO's.

A middleware would do the work of validating the user's authentication, extracting the user id from the authentication token and pass it to the service layer, so the application can know what user will have it's notes saved, deleted or retrieved.

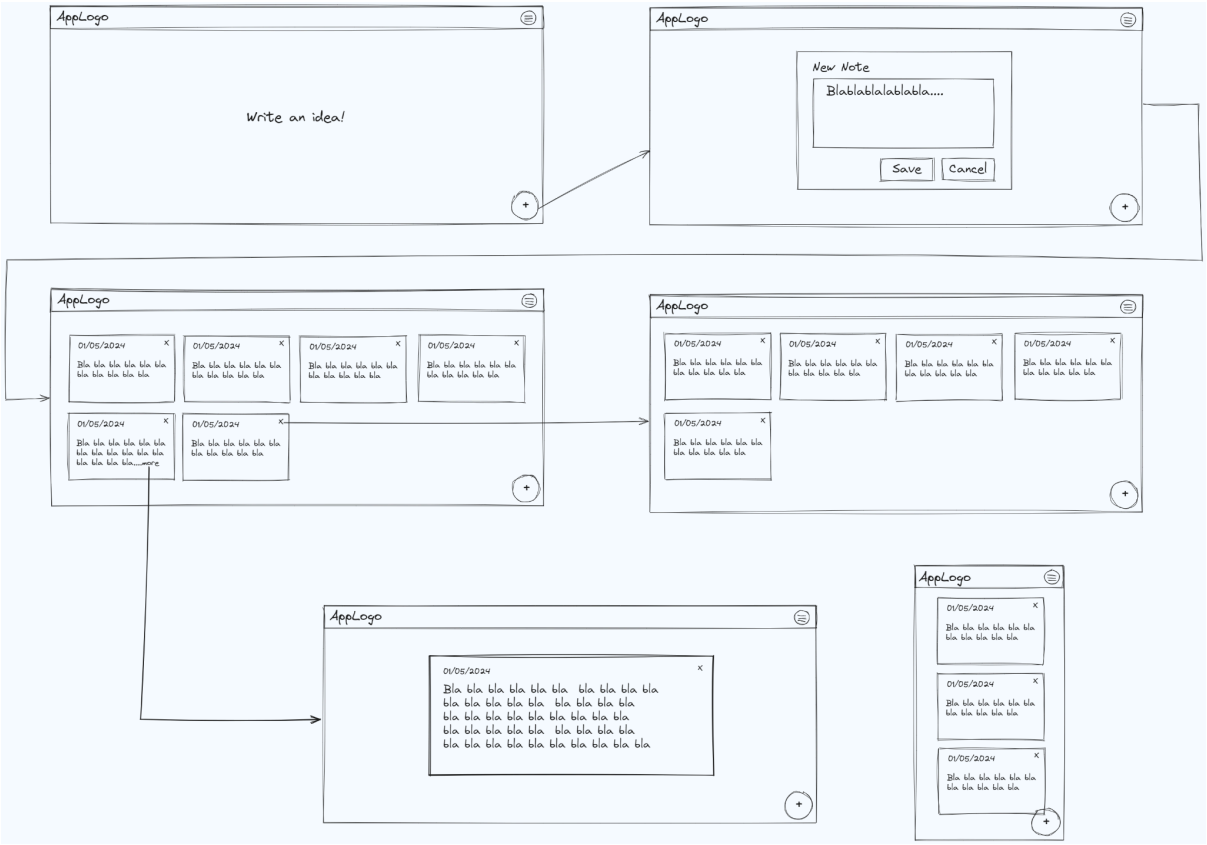
Being written in Java, the server-side app will be built using Springboot and Hibernate to manage the interactions with the database.

Database:

The project will use a relational Mysql database, containing a table called "Note". The entity model will be explained in detail in a later section of this documentation, but would consist of Id, user Id, content and creation date columns.

2. High-Level Design

- **Components needed:**
    - Customizable buttons, so we can have buttons for both save and cancel an input;
    - A modal component, for the text input;
    - A fabicon for the “write a new note” action button;
    - A card component to present the saved note;
- **Validations:**
    - The user cannot save an empty note.



3. Data Model

note		
Name	**Type**	**Observation**
id	INT	AUTO INCREMENT, NOT NULL
user_id	INT	FOREIGN KEY, NOT NULL
content	TEXT	NOT NULL
creation_date	TIMESTAMP	NOT NULL, DEFAULT CURRENT_TIMESTAMP

4. Restful API

- **How would the web app get the user's notes?**
  Once the user has signed in, the web app would make a **GET** request to the server-side, sending as a header the authentication-token provided by the login action and the desired page as a param. The notes would then be returned in a paginated response.
- **How would the web app save a user note?**
  The web app would make a **POST** request to the server-side, sending as a header the authentication-token provided by the login action. Also the content of the note would be sent as part of the body of the request. The response status code would be used to give the user a feedback if the request was successfully done or something went wrong.
- **What are the URL for the note resource(s) and verbs to expose the actions?**
  - GET /notes: Retrieve a list of user's notes;
  - POST /notes: Save a new note;
  - DELETE /notes/{note_id}: Delete a note by its ID.

5. Web Server

- **Consider how each action will be implemented?**
  In all the scenarios described below, the user is successfully authenticated, otherwise the middleware will block the operation and return a status 401 Unauthorized.
  - **GET /notes?pageNo=xx:**
    Once the server receives the request, the controller responsible for the route extracts the user id from the authentication object and gets the page number from the query param, then it passes both to the service method responsible for getting the notes.

The service layer then proceeds to call the repository responsible for being the interface with the database and passing the user id to be used as filter and the number of the current page, the repository will return a pageable result of notes.

- **POST /notes:**
  When the request arrives at the server, the controller will extract the user id from the authentication object and gets the content of the note from the request body.
  Then, both the informations will be provided to the service method responsible for saving new notes, which will call the repository responsible for the note entity persistence so a new note can be created in the database, saving the content and using the user id as a relational key.
- **DELETE /notes/{note_id}:**
  The controller will get the id of the note to be deleted via path param and pass it along with the user id retrieved from the authentication object to the service method responsible for deleting a note from the database.
  The repository will then be called, receiving the id's as parameter for the database operation.

- **What (if any) business logic is required?**
  - A note with empty body cannot be accepted by the backend;
  - Storing the note data along with metadata, such as creation date (it could use the default value in the database);
  - The application must return a paginated result in the GET /notes action .

- **How the notes are saved?**
  The repository, utilizing Hibernate ORM, persists the note entity into the MySQL database. The note is saved with its content, and the user ID serves as a relational key, associating the note with the respective user.