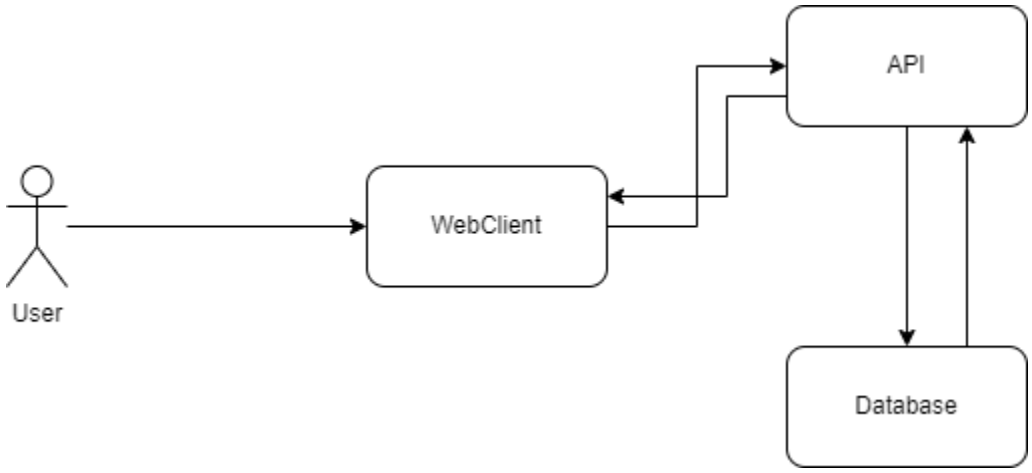
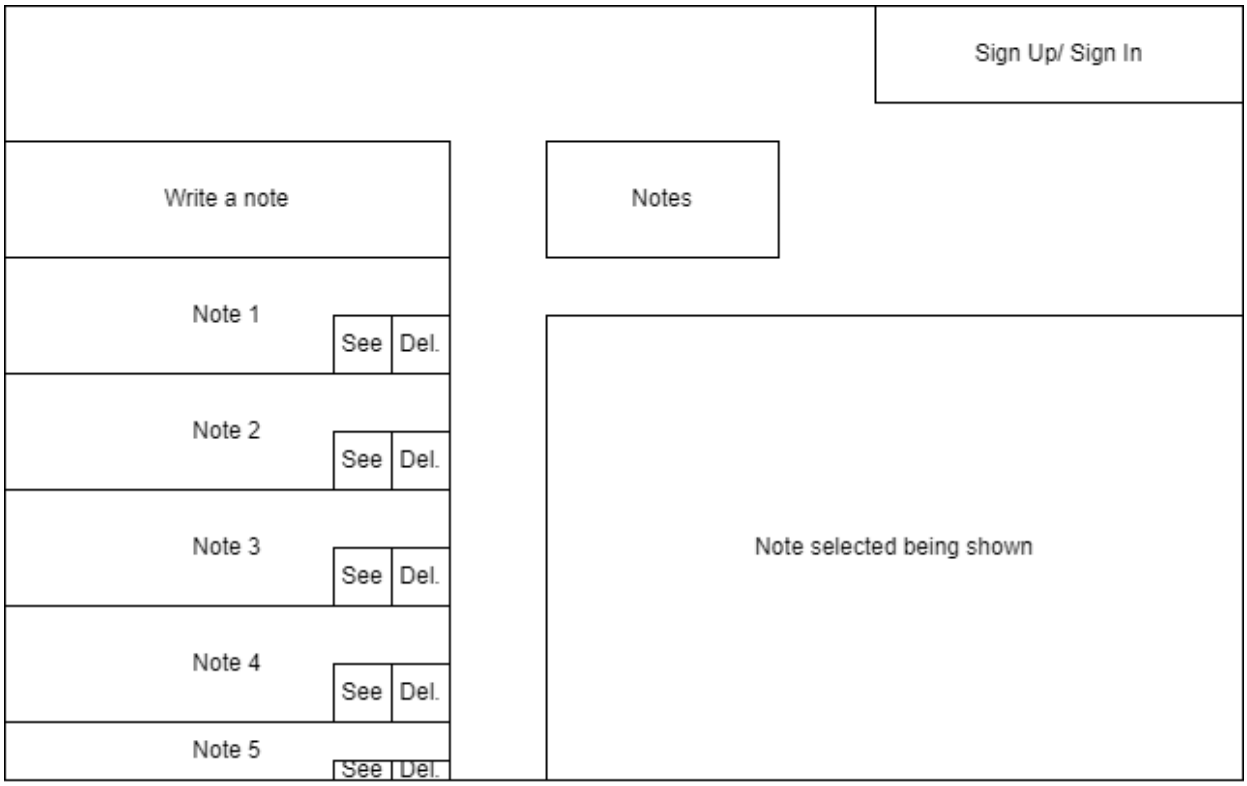
1. See the diagram attachment (opens with draw.io) for more detailed image:



The user interacts with the app through the web client (via web browser, in laptops, mobile phones, tablets…).
Each action the user needs to do makes a call to a backend API (registration, create session, create notes, list notes, delete note…).
Every API call makes some changes or queries to the database (create user, create session, notes CRUD operations).

2.



The left column shows a button to create a new note, and below shows the notes the user has already written, within two buttons on the right down corner, one to select a note to be shown and other to delete a note.
The right column shows the app title “Notes”, and a bigger area to show the note that the user selected to view or shows an input field where the user makes a new note.
The top right corner shows a sign in/ sign up button where the user goes to a form for input login details or registration data.

When a new note is being created, we cannot permit the user to create a blank note, nor a note with more than 200 characters, so we need to show an error message when some of these states happen.
When a new user signs up in the app, we need to collect their first name, email and a password. All of these fields are required and so we also need to show an error message if some of this information is not given.

3.
A note cannot have more than 200 characters and cannot be empty. When a note is saved, it is necessary to inform the note content, generate an id to identify the note on the database, a timestamp to know when the note was created and a foreign key column to identify the user that created the note.

When a new user signs up in the app, we must keep their first name, their email, a digest to validate their encrypted password and assign them a unique id to identify them on the system.

4.
Registration/Session routes:

POST - {{appURL}}/users/new -> creates a new user
Params: firstName, email, password
Response: Status 201

POST - {{appURL}}/sessions/new -> creates a new session
Params: email, password
Response: Status 200, userSessionId, userSessionToken

DELETE - {{appURL}}/sessions -> destroys a session
Params: userSessionId
Response: Status 204

Notes routes:

GET - {{appURL}}/notes -> list all the user notes
Params: userSessionToken
Response: Status 200, List containing all the user notes (noteId, noteContent)

POST - {{appURL}}/notes -> creates a new note
Params: userSessionToken, noteContent
Response: Status 201, noteId, noteContent

GET - {{appURL}}/notes/:noteId -> get a note from the user
Params: userSessionToken, noteId
Response: Status 200, noteId, noteContent

DELETE - {{appURL}}/notes/:noteId -> delete a note from the user
Params: userSessionToken, noteId
Response: Status 204

5.
POST - {{appURL}}/users/new -> creates a new user

The server will create a new user registry in the database using the first name and email provided, giving the user a unique id and also persisting a digest of the password provided, since it is not a good practice to save unencrypted passwords on the database directly. We must validate if a user with the same email already exists in our database since we do not desire to save the same user twice or more.

POST - {{appURL}}/sessions/new -> creates a new session

The server will create a new user session in the database, validating the email and password provided and creating only if they match to a registered user in the database, when creating a session, a userSessionId and a userSessionToken are created and persisted in the database within the user id, and sent to the API response.

DELETE - {{appURL}}/sessions -> destroys a session

The server will destroy a session register in the database with the given userSessionId provided.

Notes routes:

GET - {{appURL}}/notes -> list all the user notes

The server will validate the token provided, looking for a session with this token associated, and then query the database searching for all the notes belonging to the user associated with this session, getting all the notes and sending them in the response. (We could use pagination here, but for simplicity, we will get all of the notes in a single time)

POST - {{appURL}}/notes -> creates a new note

The server will validate the token provided as mentioned in the “list all the user notes” route, and will create a note in the database using the user id from the session, generating a note id, the note content provided in request. And then, sent back to the response the note id and the note content.

GET - {{appURL}}/notes/:noteId -> get a note from the user

The server will validate the token provided as mentioned in the “list all the user notes” route, and will query the database for a note with note id equal to the id provided in request. And then sent the note id and note content on the response.

DELETE - {{appURL}}/notes/:noteId -> delete a note from the user

The server will validate the token provided as mentioned in the “list all the user notes” route, and will delete in the database the note with the id given in the request. No response (a blank response) is sent back.