1. Planning our models

Our database will consist of 3 different collections:

- A User collection, that has a Username, email, a hashed password, and a list of note references(ids).
- A Note collection, that has a unique Title per user, a content, a creator referencing to the user, a list of tags, a reference to a category created by the user, and createdAt, updatedAt timestamps.
- A category collection, that has a category name, a creator referencing to a user and a list of related notes references.

Although theoretically, since a user has access only to his own notes, using embedded documents would yield faster queries, the code required to achieve data fetching will be much more complicated and harder to manage, for example we will have to use aggregation with many stages to fetch an embedded document, where when using separate collections, a simple findOne() query would fetch the same result. Therefore, we resorted to using separate collections.

2. API Workflow

The following explains how the API is expected to be used:

- First a user must sign up with his email address and a password
- After validation the user will receive a welcome email, and then sign in
- When the user signs in, he will be authenticated using a bearer token that will self-destroy in an hour
- After sign in the authenticated user can add a new category
- Only after adding a category, the user can then add a note referencing it to an already created category
- A user can also delete and update his categories, if he chooses to delete, all notes in that category will also be deleted.
- A user can also update and delete his notes, if he chooses to delete, all references to such note will also be deleted.
- A user can fetch his notes using a certain category name
- A user can fetch notes that has a certain tag
- All data will be fetched using common fields like title and category name, so the front-end can directly forward the information from the user, and not keep track of different ids.

3. File management

The following explains how our code is split:

- App.js – will contain our error handler and the code responsible for running the server (listening)
- A folder “routes”, (auth.js, category.js, notes.js) each respectively containing the code responsible for routing a validating authentication, category, and notes routes.
- A folder “controllers”, (auth.js, category.js, notes.js) each respectively containing the code responsible for the logic for authentication, category, and notes routes functions.
- A folder “models”, (category.js, notes.js, user.js) each respectively containing the Schema used creating categories notes and users
- A folder “middleware”, (isAuth.js) containing the middleware for authenticating users

All technical information and instructions for using the API are included in the README folder.