# Documentation

# Database

I chose to go with the embedded approach for the DB, because I figured I would have to do much more unnecessary queries and if I moved most things to their own collection than the DB would be no different than a SQL DB and I wanted to avoid that, because I wanted to learn something new.

### Users

Картина, която съдържа текст

Описанието е генерирано автоматично

### Post

Картина, която съдържа текст

Описанието е генерирано автоматично

### Chat

Картина, която съдържа маса

Описанието е генерирано автоматично

### Notes

I don’t think the schema is to embedded. I find it easier to use compared to a SQL schema where everything would be it’s own table and the schema would be twice the size. The one place where I think it might be too much is the **users** schema.

# Backend

### General

#### Technologies & tools

When I started the project I knew the focus would be the backend, but I still wanted to use a frontend framework, so I chose **Vue** for the because of it’s simplicity and how lightweight it is. I have used both **Angular** and **React** before, so I decided to learn Vue as well.

The backend is build with **TS, NodeJS, Express & MongoDB (ODM is mongoose)**. I was learning all these technologies at the time of creating the project and I chose them for that reason.

#### Folder structure

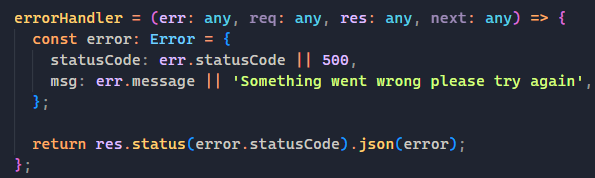
The structure for both the **client & server** is **Folder-by-Structure** style. I believe this style is superior to the older **Folder-by-Type** style, because you have everything you need immediately when you open the folder for that specific feature, you don’t have to fiddle around, wasting time, looking for the file that you need.

#### app.ts

The heart of the backend app. This is where the **routes, middlewares, db**  **& other libraries** are registered and initialized.

#### Error handling

Custom error handling middleware. Registered in **app.ts**.



Custom error classes for throwing exceptions.

Картина, която съдържа текст

Описанието е генерирано автоматично

#### Request cycle

Routers handle each request and call a method from the respective controller.



Each router is registered in the **app.ts** file.

Картина, която съдържа текст

Описанието е генерирано автоматично

Controller methods handle receiving data and then passing it onto the **service.**

Картина, която съдържа текст, монитор, екран, затвори

Описанието е генерирано автоматично

Services handle the **business logic** **& building the response model.**

Картина, която съдържа текст

Описанието е генерирано автоматично

Some service functions use **repositories**. Repositories handle **querying** the database. The base **Repository<TReturn> class** contains the base functions that all other repositories **extend** from. Some repositories have their own custom logic specific to them.

Картина, която съдържа текст, екран, екранна снимка, сребро

Описанието е генерирано автоматично

Картина, която съдържа текст

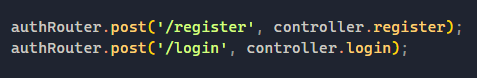
Описанието е генерирано автоматично

After the service finishes it’s work it returns a value to the controller, which then returns it as a response.

# Auth

The authentication is handled using **JWT**. The lifetime of the token is set to 24h.

### Routes



# Chat

The chat is built with **socket.io**. When a message is sent a notification is emitted and handled on the frontend.

### Routes

Картина, която съдържа текст

Описанието е генерирано автоматично

# Post

CRUD operations. Creation of comments and liking of posts is handled here as well.

### Routes

Картина, която съдържа текст

Описанието е генерирано автоматично

# Tags

Sets the initial tags, when a user is registering.

### Routes



# User

CRUD operations. Notifications and friend requests are handled here. The notifications are built using **socket.io**. When someone sends a friend request an event is emitted and handled on the frontend.

### Routes

### Картина, която съдържа текст Описанието е генерирано автоматично