Disfo - Discussion Forum (Part 1)

Concepts - Joi Validations, Environment Variables, Middleware, MongoDB, Mongoose Models & Schema, Creating and Reading.

We shall be building a server for a discussion forum that can store a discussion and the comments under it from different users.

## Steps to start

* Create a new folder named mern-2-takehomes/disfo-<your-name>, for example, disfo-vivek-nigam **inside your ~/workspace directory**.
  + **Note: You must create the folder inside ~/workspace directory, you can lose your progress otherwise**
* Initialise a new node project inside it.
* Install nodemon, express, mongoose, dotenv, and joi dependencies.
* Create 2 new scripts -
  + start - node index.js
  + dev - nodemon index.js
* Create a new index.js file to start with.

## Checkpoint 1

* Prepare your express application and listen to PORT 8082.
* Log a message “Server Listening at PORT 8082” when the server starts successfully.
* Connect to your MongoDB database at "mongodb://127.0.0.1:27017" using mongoose
* Log a message if the DB successfully connects, log the error if the connection fails.
* If everything goes well, you should be able to start the server in dev mode using the “npm run dev” script you created.

## Checkpoint 2

* We need to next prepare the user routes to create a new user and also to get the user from the database.
* Create a new file user.routes.js in a new routes folder and initialise the express router inside it.
* Create 3 new routes.

### POST /user/register

Adds a new user to the database after validating the schema of the request. Study the request body to create a Mongoose Schema for User document.

**Request**

* **Body**

| **Key** | **Description** |
| --- | --- |
| fullName | String, max characters - 50, default = “” |
| username | String, unique, required, max characters - 25 |
| email | String, unique, required, valid email |

**Response**

* **200 (OK) -** If a new user is successfully created in the database and JSON of the new user object.
* **409 (Conflict)** - If a user with the same username or email already exists with JSON { message: "Failed to create new user", reason: "Already Exists in DB" }
* **500 (Internal Server Error)** - In all other scenarios like failing to save the user to the DB etc.

### GET /user/all

Finefaluds and returns all the user documents from the database only if the header “x-api-key” is found with the value “Abracadabra”. This way the route is protected from unauthorised access.

Here “x-api-key” is a custom header we shall send with our request. All custom headers follow the naming convention

* Prefixed with “X-”
* Use lowercase kebab case. Examples: “x-api-key”, “x-my-private-header” etc.

**Request**

* **Headers** - {“x-api-key”: “Abracadabra”}

**Response**

* If the API Key is found and matches the value “Abracadabra” and users are found in the database then
  + **200 (OK) -** And an array of all the user documents.
* Else if API Key fails to match then
  + **403 (Forbidden)** - And JSON {message: “Unauthorised Access”}
* If API Key matches but no users are found then
  + **404 (Not Found)** - And {message: “No Users found”}
* **500 (Internal Server Error)** - In all other scenarios like failing to return the users from the DB etc.

### GET /user/:username

Returns the user object by matching the username field of the user in the DB.

**Request**

* **Params -** {username: string}

**Response**

* **200 (OK) -** If the user is found in the DB and JSON of the user document.
* **404 (Not Found)** - If a user with the given username is not found in the DB. JSON - { message: "User not found!", username }. Here username is the passed username with the request parameters.
* **500 (Internal Server Error)** - In all other scenarios like failing to fetch the user from the DB etc.

### Hints & Suggestions

* Create 3 routes in the routes/user.routes.js file and all these user routes should be available at <workspace-ip>/user
* Use middlewares for validating the request schema. Make use of the Joi library for validating the keys of the request object. Documentation Link - <https://joi.dev/api/?v=17.5.0>
* Create a separate file validations/user.validator.js to create the userValidationSchema and the validateUser function that uses the schema to validate any incoming request.
* Create a middleware to validate the x-api-key header in the request. This middleware shall match the string by retrieving it from the .env file. Remember to initialise the dotenv package.
* Store all your middlewares for validating the incoming requests in a separate middlewares folder.
* Create a separate folder for models with the user.model.js file to store the mongoose schema and export the mongoose model from it.
* Each route should be controlled by a controller stored in the corresponding .controller.js file. *Example* - all user route controllers in controllers/user.coantroller.js