Disfo - Discussion Forum (Part 2)

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

In Part 1 we built the user routes for our discussion forum server and their models & validations, along with protected routes. Next, we'll be adding the /discussion functionality to it.

Steps to start

- Copy your folder at `mern-2-takehomes/disfo-<your-name>`, for example, disfo-vivek-nigam **inside your ~/workspace directory** to `~/workspace/mern-2-takehomes/disfo-<your-name>-2`
    - **Note: You must create the folder inside ~/workspace directory, you can lose your progress otherwise**

Example –

```
crio-user@vivek-nigam-crio-users:~/workspace$ cp -r disfo-vivek-nigam/ disfo-vivek-nigam-2
```

We'll use the same folder structure that we had earlier

- routes - for all the routes (files names like `<module>.routes.js`)
- controllers - for the controllers of the routes (files named like `<module>.controller.js`)
- middlewares - for any middlewares we apply to routes (files named like `<module>.middeware.js`)
- validations - for the Joi validation schemas (files named with `<module>.validator.js`)
- models - for all the mongoose models (files named like `<module>.models.js`)

Checkpoint 1

For the first checkpoint, we shall create routes to create a new discussion and get the discussion by id and by a user.

Now such endpoints may later be helpful to list down all the discussions on the frontend and also be helpful to find and display all the discussions by a user on their profile page.

NOTE: All the checks and validations are to be performed before reaching the controller for the route and also in the order given from top to bottom.

`POST /discussions/new`

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

Request

The request object contains a body that is of the structure described below

Key	Description
title	String, max characters - 150, required
author	String, required, immutable
content	String, default = “”

Response

- **200 (OK) -** If a new discussion is successfully created in the database and JSON of the new discussion document.
  NOTE - All discussion objects MUST have a createdAt and updatedAt field in them.
- **500 (Internal Server Error)** - In all other scenarios like failing to save the discussion to the DB etc Also return the error object (if any)

Check and validations

1. **fetchUserInCollection -** Checks if the “author” is registered in the database or not. If found, the request proceeds. Else return 404 with `{message: “user not found”, author}`
2. **validateDiscussion** - Validates the request object to check if the keys and their respective values are correct and within all constraints. If yes, then the request proceeds, else return 422 along with error object (Hint: error from Joi validation result)

`GET /discussions/all1`

Finds and returns all the discussions from the database only if the header “`x-api-key`” is found with the value “Abracadabra”. This way the route is protected from unauthorized 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: “Unauthorized Access”}`
- If API Key matches but no discussions are found then
    - **404 (Not Found)** - And `{message: “No Discussions found”}`
- **500 (Internal Server Error)** - In all other scenarios like failing to return the discussions from the DB etc.

Checks and Validations

1. **checkAdminKey** - Check if the header is present and matches the value. If yes then the request proceeds else return back **403** and JSON `{message: “Unauthorized Access”}`

`GET /discussiotitns/user/:username`

Returns the discussion documents for the given user by matching the `username` field of the user in the DB.

Request

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

Response

- **200 (OK) -** If the discussions for the given username are found in the DB then return the JSON of the discussion documents.
- **404 (Not Found)** - If no documents for the given username are found in the DB. JSON - `{ message: "No discussions found for this user",`

`username, error }`. Here username is the passed username with the request parameters.

- **500 (Internal Server Error)** - In all other scenarios like failing to fetch the discussions from the DB etc.

`GET /discussions/id/:id`

Returns the discussion document with the given `id` in the DB.

Request

- **Params -** `{id: string}`

Response

- **200 (OK) -** If the discussions for the given id is found in the DB then return the JSON of the discussion document.
- **404 (Not Found)** - If no documents for the given id is found in the DB and JSON - `{ message: "No discussions found with this id", discussionId: id, error }`. Here `id` is the passed `id` with the request parameters.
- **500 (Internal Server Error)** - In all other scenarios like failing to fetch the discussion from the DB etc.

Checkpoint 2 (Optional)

Here on we get into a little more depth on updating the discussions by editing them and deleting them.

`DELETE /discussions/id/:id`

Finds a discussion document with the given `id` and `author` from the DB and deletes it. Also returns the deleted discussion object.

Request

- **Params -** `{id: string}`
- **Body** - `{author: string}`

Response

- **200 (OK) -** If the discussion for the given `id` is found and deleted from the DB then return the JSON of the deleted discussion document.
- **500 (Internal Server Error)** - In all other scenarios like failing to delete the discussion from the DB etc.

Checks and Validations

1. **verifyAuthor** - Checks, if the document with `id` (from request params) belongs to the `author` (from the request body) or not.
   First fetches the discussion with `id`

- If not found then return **404** `{message: "Discussion not found"}`
- else if the discussion is found but the author (from the document) does not match the author passed from the body return back **403** and JSON `{message: "Unauthorized Access"}`
- Else proceed with the request.
- If any of the steps cause an error, return **500**, `{message: "Unable to verify author"}`

PATCH /discussions/id/:id

Finds a discussion document with the given `id` and `author` from the DB and update it. Also returns the updated discussion object.

Request

- **Params -** `{id: string}`
- **Body** - `{author: string}` along with other changes to be patched.

Response

- **200 (OK) -** If the discussion for the given `id` is found and updated from the DB then return the JSON of the updated discussion document.
- **500 (Internal Server Error)** - In all other scenarios like failing to u the discussion from the DB etc.

Checks and Validations

1. **verifyAuthor** - Checks, if the document with `id` (from request params) belongs to the `author` (from the request body) or not.

   First fetches the discussion with `id`

- If not found then return **404** `{message: "Discussion not found"}`
- else if the discussion is found but the author (from the document) does not match the author passed from the body return back **403** and JSON `{message: "Unauthorized Access"}`
- Else proceed with the request.
- If any of the steps cause an error, return **500**, `{message: "Unable to verify author"}`

Checkpoint 3 (Optional)

We can now CRUD our discussion documents. To make it even more interesting, let's add the functionality to comment on a discussion too. Other registered users of our service can comment on an existing discussion.

We will need to update our discussion model and validation schema to support the comments array. Also, we shall add a new comment mongoose schema and validation Schema for it.

Updated Document Schema (with timestamps and _id)

Key	**Description**
title	String, max characters - 150, required
author	String, required, immutable
content	String, default = ""
comments	Array [*commentSchema*], default - [ ]

New Comment Schema (with timestamps and _id)

Key	**Description**
author	String, required, immutable
content	String, required, max characters - 500

`PUT /discussions/:id/comment`

Finds a discussion document with the given `id` from the DB and updates its comment array. Also returns the updated discussion object.

Request

- **Params -** `{id: string}`
- **Body** - `{author: string, content: string}`

Response

- **200 (OK) -** If the discussion for the given `id` is found and updated from the DB then return the JSON of the updated discussion document.
    - **Hint** - You can update an array within a document using mongoose operators like `$push`. Example Link
    - Also, use remember to pass options to return the updated document with your update function. Reference
- **500 (Internal Server Error)** - In all other scenarios like failing to u the discussion from the DB etc.

Checks and Validations

1. **fetchUserInCollection -** Checks if the “author” is registered in the database or not. If found, the request proceeds. Else return 404 with `{message: “user not found”, author}`
2. **fetchDiscussion** - Checks whether the discussion with passed id exists or not. If found, the request proceeds. Else return 404 with `{message: “discussion not found”, discussionId: id}`
3. **validateComment** - Validates the request body to check if the keys and their respective values are of correct data type and within all constraints.
   If yes, then the request proceeds, else return 422 along with error object (Hint: error from Joi validation result)

References

- Joi Validations - https://joi.dev/api/?v=17.5.0
- Mongoose $push - https://stackoverflow.com/a/33049923/3718061

- findOneAndUpdate - https://mongoosejs.com/docs/tutorials/findoneandupdate.html#getting-started
- Configurable Middlewares - https://medium.com/@jaeger.rob/configurable-express-middleware-3982d70af1e7