**Social Networking API with MongoDB**

**Implemented Requirements:**

* User Profiles: Sign Up, Sign In, Log Out, Get User Information, Update User Information, Delete User.
* Authentication: Implemented JWT-based authentication to secure the API.
* Posts: Users can create, view, update, and delete their posts. A post includes text content, a timestamp, and the ID of the user who created it.
* Following Mechanism: Users can follow and unfollow other users. Also ensured that there's an endpoint to retrieve the list of users a given user is following and who is following them.
* Efficient Data Retrieval: Made use of MongoDB's aggregation framework to implement efficient queries for the social feed, where users can see posts from people they follow, sorted by the most recent.
* Data Modelling: Designed MongoDB schemas for users, posts, and follows. Made use of references or embedded documents for optimal performance and query efficiency.
* Security and Validation: Ensured input validation to prevent injection attacks and other common security vulnerabilities (by escaping and sanitizing input). Implemented rate limiting to protect against brute force attacks (max 100 requests can be made using one IP).
* Error Handling: Made API to be working on proper conditions and handled most of the common errors, blank/no inputs, invalid inputs, etc. separately to ensure efficiency.
* Testing: A thorough testing was made before and after the deployment of the server, considered all the possible expected outcomes and actual outcomes to be same. Made sure the webserver doesn’t get crashed / stop due to some unexpected errors or harmful input injections.
* Documentation: Provided a comprehensive API documentation that includes endpoint descriptions, request/response formats, and example use cases.

**API Documentation:**

Note: All the request and response are in **JSON** format.

1. User Module:
2. Endpoint: */api/auth/signup*

Description: Creates a new user, and empty array of following and follower with “userId” as this user’s Object id in **follow** document of database. Hashes password using bcrypt and then stores in Database.

Handled errors for existing user, username already taken, etc.

Example:

**Request: POST Request**

{

"username": "user1",

"password": "123456789",

"bio": "Hey There, I’m user1",

"pfpurl": "https://www.imugur.com/813678"

}

*Note: The* ***username*** *and* ***password*** *are mandatory, but* ***bio*** *and* ***pfpurl*** *isn’t mandatory. pfpurl is for the URL of the profile pic.*

**Response:**

{

    "success": true,

    "message": "User Signed up successfully",

    "data": {

        "username": "user1",

        "bio": "Hey There, I’m user1",

        "pfpurl": "https://www.imugur.com/813678",

        "password": "$2b$10$2/zxFLpnCg8s3X1xxuKN4O.CeiA7KJm0bz0Q7chAf2S50LK4.IHEq",

        "\_id": "65f566af63633511b01e63a8",

        "createdAt": "2024-03-16T09:30:23.761Z",

        "updatedAt": "2024-03-16T09:30:23.761Z",

        "\_\_v": 0

    }

}

1. Endpoint: */api/auth/signin*

Description: Logs in user and generates a token using jwt token and sets Cookie.

Handles errors for invalid credentials, empty username, etc.

Example:

**Request: POST Request**

{

"username": "user1",

"password": "123456789",

"bio": "Hey There, I’m user1",

"pfpurl": "https://www.imugur.com/813678"

}

**Response:**

{

    "success": true,

    "message": "User Logged in successfully",

    "data": {

        "\_id": "65f566af63633511b01e63a8",

        "username": "user1",

        "bio": "Hey There, I’m user1",

        "pfpurl": "https://www.imugur.com/813678",

        "createdAt": "2024-03-16T09:30:23.761Z",

        "updatedAt": "2024-03-16T09:30:23.761Z",

        "\_\_v": 0

}

*Note: For Next endpoints, we need token value. So, after signing in, copy the* ***value*** *from cookies named token.*

1. Endpoint: */api/auth/updateuser*

Description: Updates the user details (username / bio / pfpurl / password) if the user is logged in and cookies is set. In order to set cookie, head towards Headers, and set Key as **Cookie** and paste the value as copied **value** from last endpoint with prefixed as **token**=

Handled errors for empty inputs.

Example:

**Request: POST Request**

{

    "pfpurl": "randomURL",

    "bio": "Testing the endpoints"

}

**Response:**

{

    "success": true,

    "msg": "User Details updated successfully",

    "updatedData": {

        "\_id": "65f566af63633511b01e63a8",

        "username": "user1",

        "bio": "Testing the endpoints",

        "pfpurl": "randomURL",

        "createdAt": "2024-03-16T09:30:23.761Z",

        "updatedAt": "2024-03-16T09:50:17.733Z",

        "\_\_v": 0

    }

}

1. Endpoint: */api/auth/logout*

Description: Logs out the user and resets the token. The user must be Logged In (i.e. set the token in cookies. See (c) Endpoint)

Handled errors if user isn’t logged in via jwt authentication.

Example:

**Request: GET Request** *with token in cookies*

**Response:**

{

    "success": true,

    "message": "Logged Out successfully"

}

1. Endpoint: */api/auth/deleteuser*

Description: Deletes the user and resets the cookie. The user must be Logged In (i.e. set the token in cookies. To set, see (c) Endpoint). Also deletes the Deleted user’s userId from all other user’s following and follower. And all the posts of Deleted User are deleted.

Handled error is user isn’t logged in via jwt authentication.

Example:

**Request: GET Request**

**Response:**

{

    "success": true,

    "message": "User Deleted and Logged Out successfully"

}

1. Post Module

*Note: For all the endpoints further, the user must be logged in, i.e. the cookie token must be set. To set see endpoint (c) from User Module.*

The logged in user can create his/her own post with text content. User also has access to view all, delete, and update his/her posts. The reference variable as userId is used to specify which posts belong to whom, and only authenticated users are able to alter them.

1. Endpoint: */api/post/createpost*

Description: The user can create a new post with just **content** in it (text format)

Handled error if user isn’t logged in via jwt authentication, or no content.

Example:

**Request: POST Request**

{

    "content": "Aditya's Post 1"

}

**Response:**

{

    "success": true,

    "msg": "Post created successfully",

    "post": {

        "userId": "65f5355cac38ad1fd362d3ba",

        "content": "Aditya's Post 1",

        "\_id": "65f57b3a24368f16554c2e84",

        "createdAt": "2024-03-16T10:58:02.651Z",

        "updatedAt": "2024-03-16T10:58:02.651Z",

        "\_\_v": 0

    }

1. Endpoint: */api/post/viewpost*

Description: Show’s all the posts this user has created till now. Only shows his/her own posts, not the others.

Handled error is user isn’t logged in via jwt authentication.

Example:

**Request: GET Request**

**Response:**

{

    "success": true,

"msg": "User's Post's fetched Successfully",

    "posts": [

        {

            "\_id": "65f57b3a24368f16554c2e84",

            "userId": "65f5355cac38ad1fd362d3ba",

            "content": "Aditya's Post 1",

            "createdAt": "2024-03-16T10:58:02.651Z",

            "updatedAt": "2024-03-16T10:58:02.651Z",

            "\_\_v": 0

        }

    ]

}

1. Endpoint: */api/post/updatepost*

Description: Updates the existing post which user has posted beforehand. Requires the **postId** of the post to be updated.

Handled error if user isn’t logged in via jwt authentication, or no content.

Example:

**Request: POST Request**

{

    "postId": "65f57b3a24368f16554c2e84",

    "content": "My First Post"

}

**Response:**

{

    "success": true,

    "msg": "Post Updated Successfully",

    "updatedPost": {

        "\_id": "65f57b3a24368f16554c2e84",

        "userId": "65f5355cac38ad1fd362d3ba",

        "content": "Aditya's Post 1",

        "createdAt": "2024-03-16T10:58:02.651Z",

        "updatedAt": "2024-03-16T10:58:02.651Z",

        "\_\_v": 0

    }

}

1. Endpoint: */api/post/deletepost*

Description: Deletes the existing post which user has posted beforehand. Requires the **postId** of the post to be deleted.

Handled error if user isn’t logged in via jwt authentication.

Example:

**Request: POST Request**

{

    "postId": "65f57b3a24368f16554c2e84"

}

**Response:**

{

    "success": true,

    "msg": "Post deleted successfully",

    "deletedpost": {

        "\_id": "65f57b3a24368f16554c2e84",

        "userId": "65f5355cac38ad1fd362d3ba",

        "content": "My First Post",

        "createdAt": "2024-03-16T10:58:02.651Z",

        "updatedAt": "2024-03-16T11:03:47.729Z",

        "\_\_v": 0

    }

}

1. Follow Module:
2. Endpoint: */api/follow/followuser*

Description: Follows another user. Needs userId of the user to be followed. The **follow** documentin database is updated accordingly (following and follower). It means that if user1 follows user 2, then userId of user1 is added to **follower** of user 2, and userId of user2 is added to **following** of user1.

Handled errors of self-following, already following, invalid userIdToFollow, etc.

Example:

**Request: POST Request**

{

    "userIdToFollow": "65f488f0c70f208ff2456ad9"

}

**Response:**

{

    "success": true,

    "message": "User 65f488f0c70f208ff2456ad9 followed successfully"

}

1. Endpoint: */api/follow/unfollowuser*

Description: Unfollows another user. Needs userId of the user to be unfollowed. The **follow** documentin database is updated accordingly (following and follower). It means that if user1 unfollows user 2, then userId of user1 is removed from **follower** of user 2, and userId of user2 is removed from **following** of user1.

Handled errors of self-unfollowing, not following, invalid userIdToUnfollow, etc.

Example:

**Request: POST Request**

{"userIdToUnfollow": "65f488f0c70f208ff2456ad9"}

**Response:**

{

    "success": true,

    "msg": "User 65f488f0c70f208ff2456ad9 unfollowed successfully"

}

1. Endpoint: */api/follow/followdetails*

Description: This Endpoint displays all the userId(s) of following and follower of logged in user

Handled error of blank input and invalid input.

Example:

**Request: GET Request**

**Response:**

{

    "success": true,

    "msg": "User's Follow details fetched successfully",

    "following": [

        "65f53579ac38ad1fd362d3be",

        "65f488f0c70f208ff2456ad9"

    ],

    "followers": []

}

1. Endpoint: */api/follow/userfeed*

Description: Used MongoDB’s aggregation framework to implement social feed, where the logged in user can see posts from people they follow, sorted by most recent.

Handled error if user isn’t logged in via jwt authentication. Also display empty array if no posts are there.

Example:

**Request: GET Request**

**Response:**

{

    "success": true,

    "msg": "User's feed fetched successfully",

    "data": [

        {

            "\_id": "65f53645ac38ad1fd362d3c8",

            "userId": "65f488f0c70f208ff2456ad9",

            "content": "Madhuri's Post 1",

            "createdAt": "2024-03-16T06:03:49.749Z",

            "updatedAt": "2024-03-16T06:03:49.749Z",

            "\_\_v": 0

        },

        {

            "\_id": "65f535dbac38ad1fd362d3c4",

            "userId": "65f53579ac38ad1fd362d3be",

            "content": "Aryan's Post 1",

            "createdAt": "2024-03-16T06:02:03.605Z",

            "updatedAt": "2024-03-16T06:02:03.605Z",

            "\_\_v": 0

        }

    ]

}

**Functionality:**

* The API meets all the specified requirements and function correctly across all endpoints.
* Code Quality is clean, well-organized, and properly commented. And followed the best practices for Node.js and MongoDB development.
* This application follows security best practices to protect user data and prevent common vulnerabilities. Made use of validator and sanitizeHtml for security.
* Efficiently used MongoDB for data storage with well-designed schemas for user, post and follow according to application needs.
* Efficient data retrieval for social feed functionality for leverage.
* All the API endpoints are comprehensively tested and re-checked for all the possible errors and future errors.
* A Readme file is added to the GitHub repository with setup instructions, environment variables and steps to get application running.
* The API Endpoint documentation is also provided that includes endpoint description, request format, response format with example use-cases.