**Full Stack Development with MERN**

**API Development and Integration Report**

|  |  |
| --- | --- |
| Date | 17 July 2024 |
| Team ID | SWTID1719933836 |
| Project Name | Connectify |
| Maximum Marks |  |

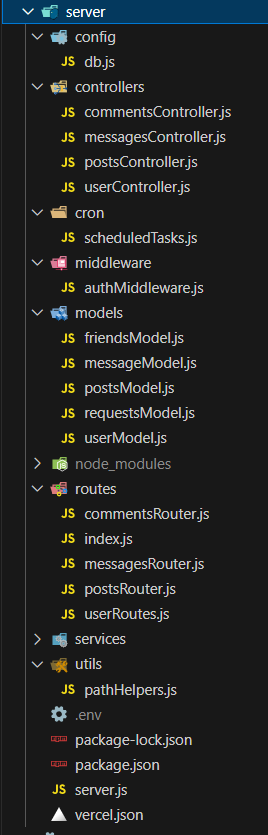
**Project Title:** Connectify  
**Date:** 17 July 2024  
**Prepared by:** Ashank Sethi

**Objective**  
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Connectify project.

**Technologies Used**

* **Backend Framework:** Node.js with Express.js
* **Database:** MongoDB
* **Authentication:** JWT

**Project Structure**

****

**Key Directories and Files**

1. **/config**
   * Configuration files for database connections, environment variables, etc.
2. **/controllers**
   * Contains functions to handle requests and responses. Controller functions handle the business logic and interact with models.
3. **/cron**
   * Scripts that perform scheduled tasks and cron jobs.
4. **/middleware**
   * Custom middleware functions for request processing. Contains middleware functions that process requests before they reach the controllers.
5. **/models**
   * Includes Mongoose schemas and models for MongoDB collections.
6. **/routes**
   * Defines the API endpoints and links them to controller functions.
7. **/services**
   * Contains service files that encapsulate the business logic and interact with external APIs or internal modules.
8. **/utils**
   * Contains utility functions and helper modules that are used throughout the application.

**API Endpoints**  
A summary of the main API endpoints and their purposes:

**User Authentication**

* **POST /api/user/signup** - Registers a new user.
* **POST /api/user/signin** - Authenticates a user and returns a token.

**User Management**

* **GET /api/getusers/ -** Retrieves user information by userID.
* **GET /api/user/getprofile** - Retrieves user profile by userID.
* **GET /api/user/getpfp –** Retrieves user profile picture by userID.
* **POST /api/user/updatepfp –** Updates user profile picture.

**Friend Management**

* **GET /api/user/getfriends –** Retrieves all the friends of a user.
* **GET /api/user/sentrequests –** Retrievesthe active friend requests sent by a user.
* **GET /api/user/receivedrequests –** Retrieves the active friend requests received by a user.
* **POST /api/user/acceptrequest –** Accepts friend requests sent by other users.

**Posts Management**

* **POST /api/posts/createpost** – Creates a new post by authenticated user.
* **GET /api/posts/getposts** – Retrieves posts to display in user feed.
* **GET /api/posts/userposts** – Retrieves all the posts created by a specific user through userID.
* **POSt /api/posts/like/:postID –** Allows the user to like a post.

**Comments Management**

* **GET /api/comments/:postID** – Retrieves all the comments on a post by postID.
* **POST /api/comments/:postID** – Creates a new comment on a post.

**Direct Messaging**

* **GET /api/comments/:postID** – Retrieves all the comments on a post by postID.
* **POST /api/comments/:postID** – Creates a new comment on a post.

**Integration with Frontend**  
The backend communicates with the frontend via RESTful APIs. Key points of integration include:

* **User Authentication:** Tokens are passed between frontend and backend to handle authentication.
* **Data Fetching:** Frontend components make API calls to fetch necessary data for display and interaction.

**Error Handling and Validation**  
Describe the error handling strategy and validation mechanisms:

* **Error Handling:** Centralized error handling using middleware.
* **Validation:** Input validation using libraries like express-validator.

**Security Considerations**  
Outline the security measures implemented:

* **Authentication:** Secure token-based authentication.
* **Input Validation:** Using express-validator to prevent SQL Injections and XSS attacks.
* **CORS:** Cross-origin resource sharing to restrict domains that can access API.
* **Protection of API Endpoints**