Full Stack Development with MERN

API Development and Integration Report

Date	13-07-2024
Team ID	SWTID1720073159
Project Name	Project - TuneTrail
Maximum Marks	

Project Title: Music Streaming Webapp
Date: 13-07-2024
Prepared by: Jaiaditya Mathur, Saket DB, Riya Autade, Savithri Nair

Objective

The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Music Streaming Webapp project.

Technologies Used

- **Backend Framework:** Node.js with Express.js
- **Database:** MongoDB
- **Authentication:** [e.g., JWT, OAuth]

Project Structure

Key Directories and Files

1. **/controllers**
    - Contains functions to handle requests and responses.
2. **/models**
    - Includes Mongoose schemas and models for MongoDB collections.

      Files:

        1. Addsong.js
        2. Admin.js
        3. Playlist.js
        4. Wishlist.js
        5. user.js
3. **/routes**
    - Defines the API endpoints and links them to controller functions.
4. **/middlewares**
    - Custom middleware functions for request processing.
5. **/config**
    - Configuration files for database connections, environment variables, etc.

      Files:

        1. Config.js
6. **.ev**
    - Used to store environment variables
7. **.gitignore**
    - Specifies which files and directories Git should ignore

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

User Authentication

- **POST /api/user/register** - Registers a new user.
- **POST /api/user/login** - Authenticates a user and returns a token.

User Management

- **GET /api/user/-** Retrieves user information by ID.
- **PUT /api/user/-** Updates user information by ID.

Workout Plans

- **GET /api/workoutplans** - Retrieves all workout plans.

- **POST /api/workoutplans** - Creates a new workout plan.

Equipment

- **GET /api/equipment** - Retrieves all equipment.
- **POST /api/equipment** - Adds new equipment.

Monthly Plans

- **GET /api/monthlyplans** - Retrieves all monthly plans.
- **POST /api/monthlyplans** - Creates a new monthly plan.

Admin Operations

Admin Login

- **POST /alogin -** Authenticates an admin user by email and password.

Admin Registration

- **POST /asignup -** Registers a new admin user with name, email, and password.

View All Users

- **GET /users -** Retrieves a list of all users.

Delete User by ID

- DELETE /userdelete/:id - Deletes a user by their ID.

Delete Order by ID

- DELETE /userorderdelete/:id - Deletes an order by its ID.

Delete Item by ID

- DELETE /useritemdelete/:id - Deletes an item by its ID.

View All Sellers

- GET /sellers - Retrieves a list of all sellers.

Delete Seller by ID

- DELETE /sellerdelete/:id - Deletes a seller by their ID.

View All Orders

- GET /orders - Retrieves a list of all orders.

Song Operations

Add Song

- POST /addsong - Uploads a new song with title, genre, singer, image, and song URL.

View All Songs

- GET /mysongs - Retrieves a list of all songs.

Delete Song by ID

- DELETE /deletesong/:id - Deletes a song by its ID.

Update Song by ID

- PUT /updatesong/:id - Updates a song's details (including the song file) by its ID.

User Operations

User Login

- POST /login - Authenticates a user by email and password.

User Registration

- POST /signup - Registers a new user with name, email, and password.

View All Songs

- GET /songs - Retrieves a list of all songs.

View Wishlist

- GET /wishlist - Retrieves the user's wishlist items.

Add Item to Wishlist

- POST /wishlist/add - Adds an item to the user's wishlist.

Remove Item from Wishlist

- POST /wishlist/remove - Removes an item from the user's wishlist.

View Playlist

- GET /playlist - Retrieves the user's playlist items.

Add Item to Playlist

- Endpoint: POST /playlist/add - Adds a song to the user's playlist.

Remove Item from Playlist

- POST /playlist/remove - Removes a song from the user's playlist.

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

- **Error Handling:**
    1. Centralized error handling using middleware: Which helps to manage errors consistently across the app.
    2. Handling Async Errors: Uses a utility function to handle errors in asynchronous route handlers to ensure any unhandled promise rejections are caught

- **Validation:**
    1. Input validation using libraries like Joi or express-validator: It helps validate request data before it reaches your controllers
    2. Input Validation Using express-validator: Express-validator is another popular library for validation in Express.js applications

Security Considerations

Outline the security measures implemented:

- **Authentication:** Secure token-based authentication.
- **Data Encryption:** Encrypt sensitive data at rest and in transit.
- **Input Validation :** Validates user inputs to prevent SQL injection, XSS, and other attacks using Joi.
- **Input Sanitization :** Sanitize inputs to remove potentially malicious code using express-validator.
- **Secure Data Transmission :** Using HTTPS and cookies
- **Database Security**: By using environment variables to store database credentials and by implementing role-based access control for MongoDB users.
- **Server Security:** Use Helmet to set various HTTP headers for security and implementing rate limiting to prevent DDoS attacks.
- **Logging and Monitoring :** Implement logging and monitoring to detect and respond to security incidents.