**Full Stack Development with MERN**

**API Development and Integration Report**

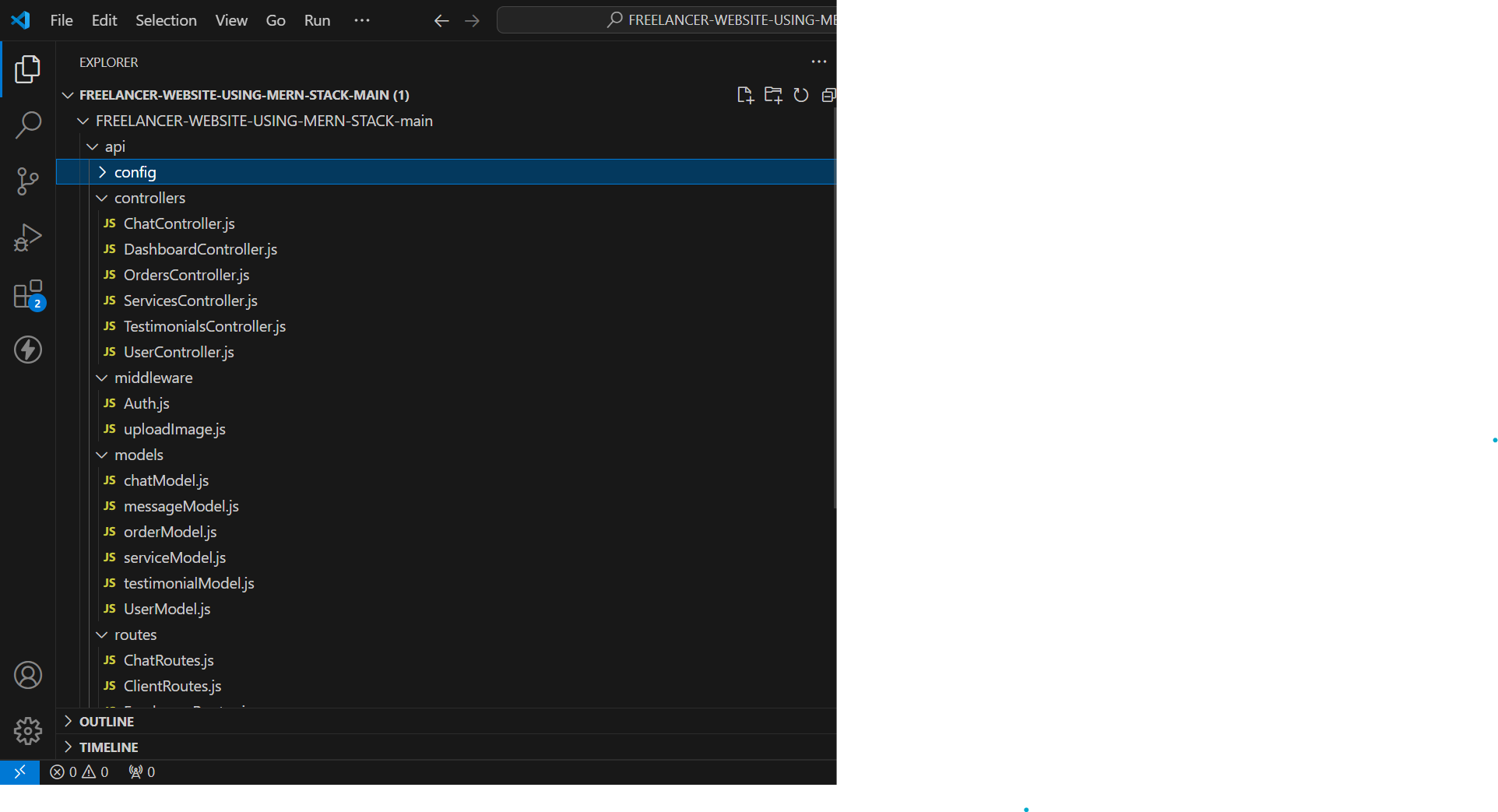
|  |  |
| --- | --- |
| Date | 16-07-2024 |
| Team ID | SWTID1720010107 |
| Project Name | Project – HIRE CONNECT |
| Maximum Marks | 10 |

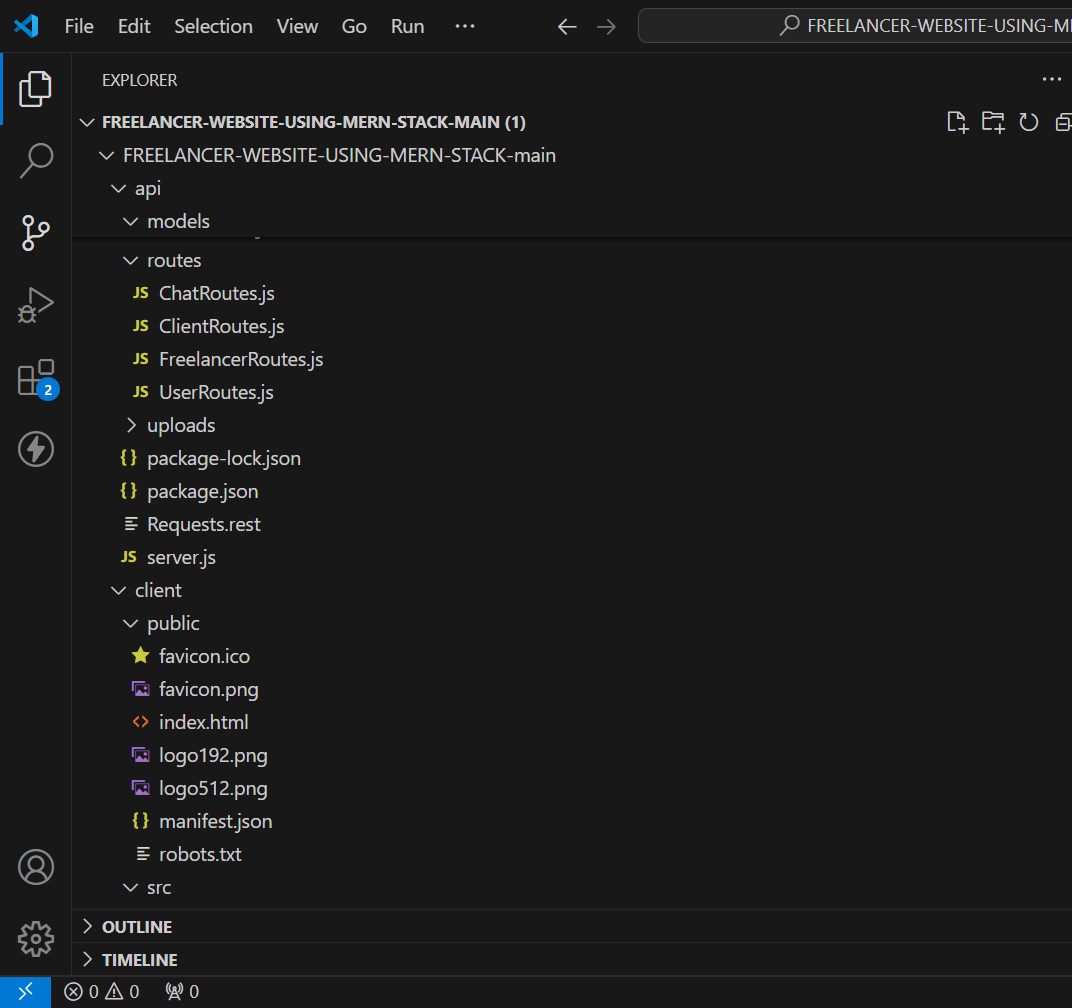
**Project Title:** Hire Connect  
**Date:** 16-07-2024  
**Prepared by:** Harika

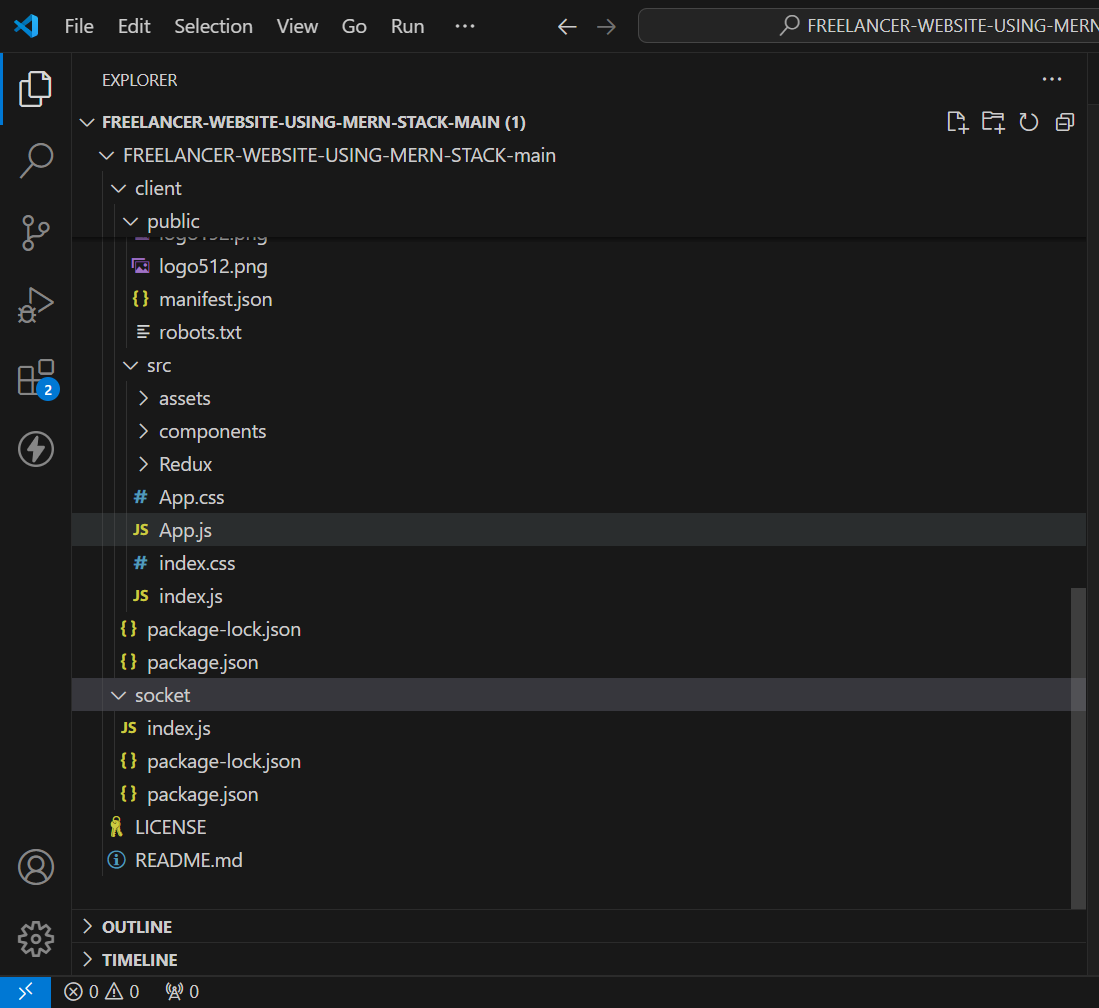
**Objective**  
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Hire Connect project.

**Technologies Used**

* **Backend Framework:** Node.js with Express.js
* **Database:** MongoDB
* **Authentication:** JSON Web Tokens (JWT**)**, with dotenv for managing sensitive environment variables.

**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.
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.

**API Endpoints**  
 **Authentication Endpoints**:

* **POST /login**: Authenticates the user and returns a JWT token if the credentials are valid. This token is then used for subsequent requests to access protected resources.
* **POST /register**: Registers a new user and may return a JWT token or a confirmation message.

 **Protected Endpoints** (requires JWT token for access):

* **GET /profile**: Retrieves user profile information. The JWT token in the authorization header is verified to ensure that the request is coming from an authenticated user.
* **GET /user/:id**: Retrieves information about a specific user. Requires authentication to ensure the requesting user has the right permissions to access the data.
* **PUT /user/:id**: Updates information for a specific user. The JWT token is used to verify the identity and permissions of the user making the request.
* **DELETE /user/:id**: Deletes a specific user. This endpoint requires a valid JWT token to ensure that only authorized users can perform deletions.

 **Token Verification Middleware**:

* **Middleware for Protected Routes**: Middleware like tokenVerification is used to protect routes by checking the validity of the JWT token. If the token is valid, the request proceeds; otherwise, an error message is returned.

**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.

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

1. **Environment Configuration**:
   * Use a .env file to store sensitive information like the JWT secret key.
2. **Middleware**:
   * **Authentication Middleware**: A function that validates the JWT token from incoming requests. It ensures that requests to protected routes are from authenticated users.
3. **Authentication Routes**:
   * **Login Endpoint**: Validates user credentials and issues a JWT token if authentication is successful.
   * **Registration Endpoint**: Allows new users to create an account (often not directly related to JWT but part of user management).
4. **Protected Routes**:
   * **Secured Endpoints**: Use authentication middleware to ensure only requests with valid JWT tokens can access these endpoints.
5. **Server Integration**:
   * **Express Server**: Integrates middleware and routes into the Express application, handling incoming requests and routing them accordingly.

**Principles Undertaken**

1. **JWT Tokens**:
   * **Authentication**: JWT tokens are issued upon successful login and include user information. They are used to authenticate requests.
   * **Statelessness**: JWTs are self-contained and do not require server-side sessions. They include all necessary information for verifying the user's identity.
2. **Security**:
   * **Token Verification**: Ensure that tokens are verified against a secret key to confirm their validity and integrity.
   * **Secure Storage**: Store secret keys and tokens securely, using environment variables for sensitive data.
3. **Authorization**:
   * **Access Control**: Use JWTs to enforce access control by protecting routes and ensuring only authorized users can access certain resources.
4. **Scalability**:
   * **Stateless Authentication**: JWTs enable scalable authentication by avoiding the need for server-side session storage, making it easier to distribute requests across multiple servers.
5. **Error Handling**:
   * **Response Codes**: Provide appropriate HTTP status codes and messages for invalid tokens or authentication failures to guide users and developers.
6. **Error Handling and Validation**  
   **Consistent Error Responses**:
   * Ensure that error responses are consistent across your API. Use a standardized format for error messages to make it easier for clients to handle errors.

Use appropriate HTTP status codes to indicate the nature of the error:

* + - **400 Bad Request**: For invalid input or malformed requests.
    - **401 Unauthorized**: For authentication failures, such as invalid or missing tokens.
    - **403 Forbidden**: For authenticated users who do not have permission to access the resource.
    - **404 Not Found**: For non-existent resources.
    - **500 Internal Server Error**: For unexpected server errors.

1. **Error Logging**:
   * Log errors for debugging and monitoring purposes. Capture stack traces and relevant details to diagnose issues.
   * Use logging libraries or services (e.g., Winston, Morgan, or external monitoring tools) for better visibility.
2. **User-Friendly Messages**:
   * Provide clear and user-friendly error messages to guide users on how to resolve issues, especially for authentication errors.
   * Avoid exposing internal error details that could be exploited.
3. **Middleware for Error Handling**:
   * Implement centralized error handling middleware in Express to catch and manage errors in one place.

**Validation Mechanisms**

1. **Input Validation**:
   * Validate user inputs to ensure they meet expected formats and constraints. This helps prevent invalid or malicious data from affecting your system.
   * Use libraries like express-validator, joi, or validator to simplify input validation.
2. **Token Validation**:
   * Verify the integrity and authenticity of JWT tokens using a secret key or public/private key pairs.
   * Ensure tokens are correctly signed and have not expired.
   * Sanitize inputs to prevent injection attacks (e.g., SQL injection, XSS). Remove or escape potentially dangerous characters from user inputs.
3. **Rate Limiting**:
   * Implement rate limiting to prevent abuse or excessive requests from a single client. Use middleware like express-rate-limit to manage request rates.
4. **Schema Validation**:
   * For complex data structures or databases, use schema validation to ensure that data conforms to expected formats and constraints.
   * Example with Mongoose for MongoDB:

**Security Considerations**

security measures implemented:

* **Authentication:** Secure token-based authentication.
* **Data Encryption:** Encrypt sensitive data at rest and in transit.
*  **Secure Secret Management**:
* **Environment Variables**: Store sensitive information such as JWT secret keys in environment variables (.env file) rather than hardcoding them in your source code.
*  **Token Security**:
* **Signing Tokens**: Use a strong and secret key to sign JWT tokens, ensuring that tokens are tamper-proof.
* **Token Expiry**: Set an expiration time for tokens to limit the window of opportunity for token misuse. For example, set tokens to expire in 1 hour.
*  **Token Verification**:
* **Integrity Check**: Verify the token’s signature using the secret key to ensure it has not been altered.
* **Validity Check**: Check the token’s expiration time to ensure it is still valid.