**Full Stack Development with MERN**

**API Development and Integration Report**

|  |  |
| --- | --- |
| Date | 18-07-2024 |
| Team ID | PNT2022TMID SWTID1720175019 |
| Project Name | Nexus Learn - Online learning platform |
| Maximum Marks |  |

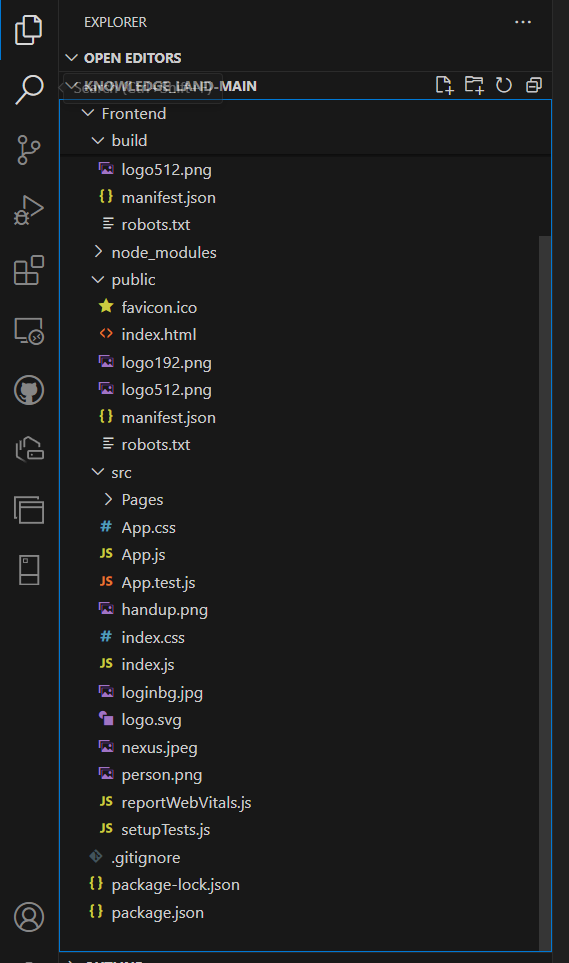
**Project Title:** Nexus Learn - Online learning platform  
**Date:** 18-07-2024  
**Prepared by:** Sathvik(Team Lead), Shanmukh, Sashank, Sudheer

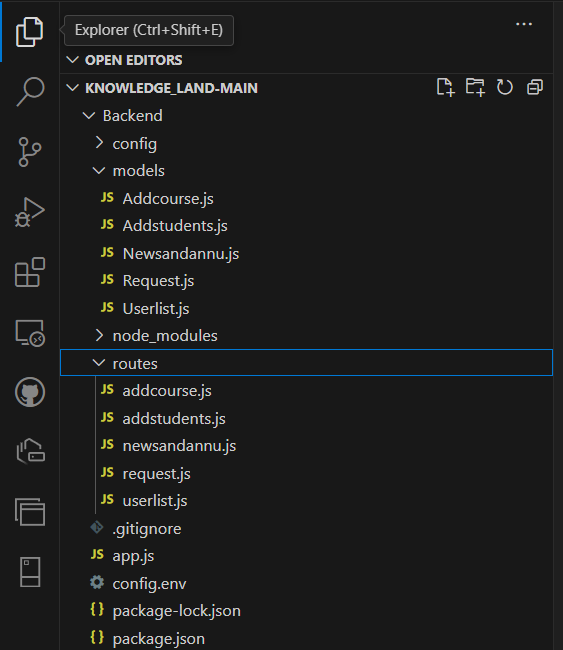
**Objective**  
The objective of this report is to document the API development progress and key aspects of the backend services implementation for the Nexus Learn - Online learning platform project.

**Technologies Used**

* **Frontend Framework:** React.js
* **State Management:** React hooks (useState, useEffect)
* **UI Framework/Libraries:** Bootstrap,Material UI
* **API Libraries:** Custom implementation using Axios
* **Testing Libraries:**
  + @testing-library/jest-dom
  + @testing-library/react
  + @testing-library/user-event
* **Other Libraries:**
  + Firebase
  + React Router DOM
  + React Toastify
  + Web Vitals

**Project Structure**

****

****

The Nexus Learn frontend project structure includes various directories and files to manage different aspects of the application.

**Key Directories and Files**

* **/controllers**: Contains functions to handle requests and responses.
  + **userController.js**: Handles user-related requests (signup, signin, etc.).
  + **courseController.js**: Manages course-related requests (create, update, delete, etc.).
  + **sectionController.js**: Handles section-related requests (add, update, delete, etc.).
  + **enrollmentController.js**: Manages student enrollment requests (enroll, unenroll, etc.).
  + **progressController.js**: Handles course progress tracking (retrieve, update).
  + **certificateController.js**: Manages certificate generation and retrieval.
* **/models**: Includes Mongoose schemas and models for MongoDB.
  + **userModel.js**: Mongoose schema and model for users.
  + **courseModel.js**: Mongoose schema and model for courses.
  + **sectionModel.js**: Mongoose schema and model for sections.
  + **enrollmentModel.js**: Mongoose schema and model for enrollments.
  + **progressModel.js**: Mongoose schema and model for progress.
  + **certificateModel.js**: Mongoose schema and model for certificates.
* **/routes**: Defines API endpoints and links them to controller functions.
  + **userRoutes.js**: API endpoints for user operations (signup, signin, etc.).
  + **courseRoutes.js**: API endpoints for course management (create, update, delete, etc.).
  + **sectionRoutes.js**: API endpoints for managing sections.
  + **enrollmentRoutes.js**: API endpoints for student enrollments.
  + **progressRoutes.js**: API endpoints for course progress.
  + **certificateRoutes.js**: API endpoints for certificates.
* **/middlewares**: Custom middleware functions.
  + **authMiddleware.js**: Middleware for authentication and authorization.
  + **errorMiddleware.js**: Middleware for handling errors.
  + **validationMiddleware.js**: Middleware for request data validation.
* **/config**: Configuration files.
  + **dbConfig.js**: Database connection settings.
  + **authConfig.js**: Authentication settings (e.g., JWT secrets).
  + **envConfig.js**: Environment variables.

**API Endpoints**

**User Authentication**

* **POST** /api/user/signup - Registers a new user with validation.
* **POST** /api/user/signin - Authenticates a user and returns a token.
* **GET** /api/user/signout - Signs out the user.

**User Management**

* **GET** /api/user/:userId - Retrieves user information by ID (requires authentication).
* **PUT** /api/user/:userId - Updates user information by ID (requires authentication).
* **GET** /api/users - Retrieves all users (requires admin authentication).
* **DELETE** /api/user/:userId - Deletes a user by ID (requires admin authentication).

**Course Management**

* **POST** /api/course/create - Creates a new course (requires admin or teacher authentication).
* **GET** /api/course/:courseId - Retrieves course details by ID.
* **PUT** /api/course/:courseId - Updates a course by ID (requires admin or teacher authentication).
* **DELETE** /api/course/:courseId - Deletes a course by ID (requires admin or teacher authentication).
* **GET** /api/courses - Retrieves all courses.
* **GET** /api/courses/categories - Retrieves all unique course categories.

**Section Management**

* **POST** /api/section/create - Adds a new section to a course (requires teacher authentication).
* **PUT** /api/section/:sectionId - Updates a section by ID (requires teacher authentication).
* **DELETE** /api/section/:sectionId - Deletes a section by ID (requires teacher authentication).
* **GET** /api/section/:sectionId - Retrieves section details by ID.

**Student Interactions**

* **POST** /api/enroll - Enrolls a student in a course (requires authentication).
* **GET** /api/enrollments/:studentId - Retrieves courses a student is enrolled in (requires authentication).
* **GET** /api/enrollment/:enrollmentId - Retrieves enrollment details by ID (requires authentication).
* **DELETE** /api/enrollment/:enrollmentId - Unenrolls a student from a course (requires authentication).

**Course Progress**

* **GET** /api/progress/:enrollmentId - Retrieves progress for a specific course (requires authentication).
* **PUT** /api/progress/:enrollmentId - Updates progress for a specific course (requires authentication).

**Certificates**

* **GET** /api/certificate/:enrollmentId - Retrieves a course completion certificate (requires authentication).

**Admin Functions**

* **GET** /api/admin/users - Retrieves all users (requires admin authentication).
* **GET** /api/admin/courses - Retrieves all courses (requires admin authentication).
* **GET** /api/admin/enrollments - Retrieves all student enrollments (requires admin authentication).

**Integration with Frontend**

**User Authentication:**

* **Tokens:** Use JSON Web Tokens (JWT) for authentication. Tokens are issued by the backend upon successful login and are included in the Authorization header for subsequent requests.

**Data Fetching:**

* **API Calls:** Frontend components make HTTP requests to the backend API endpoints to retrieve and display necessary data.

**Error Handling and Validation**

**Error Handling:**

* **Centralized Error Handling:** Implement middleware on the backend to handle errors consistently. This middleware catches and processes errors, ensuring that the frontend receives appropriate error responses.

**Validation:**

* **Input Validation:** Use a library like express-validator to validate incoming data on the backend, ensuring data integrity and security before processing.

**Security Considerations**

**Authentication:**

* **Token-Based Authentication:** Use JWT to authenticate users. Tokens are used to secure access to protected endpoints.

**Data Encryption:**

* **Encryption:** Encrypt sensitive data both at rest and in transit to protect it from unauthorized access.