FULL STACK DEVELPOMENT WITH MERN (project documentation)

Introduction

Project title: Online Learning Platform using **MERN**

Team members:

- **Ahamed Basheer** – **Project Manager & Testing** : Responsible for testing, overseeing the entire project, ensuring timelines are met, and leading the team.

- **Reyash Ahamed K** – **Database Administrator** : Manages and optimizes the database, ensuring data integrity and efficient querying.

- **Sridhar C** – **Backend Developer** : In charge of designing and implementing the server-side logic, APIs, and database interactions.

- **Furqqandeen R**– **Frontend Developer** : Develops and styles the user interface, ensuring a seamless and responsive user experience.

Project Overview

- **Purpose:**

  building an online learning platform with the MERN stack allows you to create a dynamic, scalable, and feature-rich platform, enabling students and instructors to interact with the course material efficiently

- **Features:**

  - ✓ **User Management**: Sign up, authentication, roles, profiles.
  - ✓ **Course Management**: Course creation, enrollment and tracking.
  - ✓ **Real-Time Interaction**: Live chat, forums, and live classes.
  - ✓ **Quizzes & Assessments**: Interactive quizzes, assignments, and grading.
  - ✓ **Payment Integration**: Secure payments, subscriptions, discounts.
  - ✓ **Certificates**: Automated certificates upon completion.
  - ✓ **Search and Discovery**: Course search, filtering, recommendations.
  - ✓ **Admin Features**: User management, course moderation, and analytics.

✓ **Scalability & Performance**: High availability, performance optimization.

Architecture

• Frontend:

- Developed using React, the app utilizes functional components and React Hooks for state management.

- The design is component-based, making it scalable and easy to maintain. React-Bootstrap is used for a sleek and responsive UI design.

• Backend:

- The server is built with Node.js and Express.js. It handles CRUD operations, user authentication, and integrates with a MongoDB database.

- RESTful API structure is implemented for clean and efficient communication between the frontend and backend.

• Database:

- MongoDB is used as the database. The schema is designed using Mongoose to handle data efficiently, with collections for users, properties, and bookings.

Setup instructions

Step 1: Clone the Repository

Clone the repository from GitHub using the following command:

```
git clone https://github.com/Deepanrajmuthu/Elearning-platform.git
```

Step 2: Navigate to the Project Directory

Once the repository is cloned, navigate to the project directory:

```
cd Elearning-platform
```

Step 3: Install Dependencies

Install the necessary dependencies by running the following command:

```
npm install
```

This will install all the required libraries and packages for both frontend and backend.

Step 4: Start the Development Server

Run the development server with the following command: npm start

The application will be live at http://localhost:3000.

Folder structure

• Frontend:

- /src: Main source directory for React components, pages, and services.
- /src/modules: Organized modules for various features.
- /src/modules/admin: Contains components and pages specific to the admin functionality.
- /src/modules/common: Contains reusable components accessible by multiple modules.
- /src/modules/user: Contains user-specific components and pages.
- /src/modules/user/Owner: Components and pages specific to property owners.
- /src/modules/user/renter: Components and pages specific to renters.

• Backend:

- /config: Configuration files for database and server settings.
- /controllers: Contains logic for handling API requests, organized by feature.
- /middlewares: Middleware functions for authentication, validation, and error handling.
- /routes: API route definitions, connecting endpoints to controller logic.
- /schemas: Mongoose schema definitions for data models.
- /uploads: Temporary storage for file uploads (e.g., property images).
- /index.js: Main entry point for the backend application.

Running the application

• Frontend:

• Backend:

- Access the app at: http://localhost:3000

```
npm start
```

API Documentation

```
npm start
```

Base URL: arduino

https://api.online-learning.com

Authentication

1. Sign Up (User Registration)

- **Endpoint:** POST /api/auth/signup
- **Description:** Registers a new user (student/teacher).
- **Request Body:**

json

{

"name": "John Doe",

"email": "john.doe@example.com",

"password": "securepassword123",

"role": "student" // or "teacher"

}

- **Response:**

json

{

"message": "User created successfully",

"user": {

"id": "user12345",

"name": "John Doe",

"email": "john.doe@example.com",

"role": "student"

}

}

- **Error Response:**
  - 400 Bad Request - Invalid input or missing fields.
  - 409 Conflict - Email already in use.

---

2. Login

- **Endpoint:** POST /api/auth/login
- **Description:** Logs in a user and returns a JWT token.
- **Request Body:**

  json

  {

  "email": "john.doe@example.com",

  "password": "securepassword123"

  }
- **Response:**

  json

  {

  "message": "Login successful",

  "token": "jwt_token_here"}

Error Response:

- 401 Unauthorized - Invalid credentials.

---

3. Logout

- **Endpoint:** POST /api/auth/logout
- **Description:** Logs out the user by invalidating the JWT token.
- **Request Headers:**

  json

  {

  "Authorization": "Bearer jwt_token_here"

  }
- **Response:**

Courses 1. Get All Courses

Endpoint: GET /api/courses json

{

"message": "Logout successful"

}

- **Description:** Retrieves all available courses.
- **Response:**

  json

  [

  {

  "id": "course1",

  "title": "Introduction to Programming",

  "description": "Learn the basics of programming.",

  "teacher": "Teacher Name",

  "price": 99.99,

  "rating": 4.5

  },

  {

"id": "course2",

"title": "Advanced React",

"description": "Learn advanced React concepts.",

"teacher": "Teacher Name",

"price": 129.99,

"rating": 4.7

}

]

2. Get Single Course

- **Endpoint:** GET /api/courses/:courseId
- **Description:** Retrieves details of a specific course.
- **Parameters:** ○ courseId (Path parameter) - The ID of the course to retrieve.
- **Response:**

  json

  {

  "id": "course1",

  "title": "Introduction to Programming",

  "description": "Learn the basics of programming.",

  "teacher": "Teacher Name",

  "price": 99.99,

  "rating": 4.5,

  "lessons": [

  {

  "title": "Lesson 1: Introduction",

  "duration": "15 minutes"

  },

  {

  "title": "Lesson 2: Variables and Data Types",

"duration": "20 minutes"

}

]

}

3. Create Course (For Teachers)

- **Endpoint:** POST /api/courses
- **Description:** Allows teachers to create a new course.
- **Request Body:**

  json

  {

  "title": "New Course Title",

  "description": "Course description here.",

  "price": 199.99,

  "lessons": [

  {

  "title": "Lesson 1 Title",

  "duration": "30 minutes"

  }

  ]

  }

- **Response:**

  json

  {

  "message": "Course created successfully",

  "course": {

  "id": "courseId",

  "title": "New Course Title",

  "teacher": "Teacher Name",

  "price": 199.99

}

}

- **Error Response:**

○ 403 Forbidden - Only users with the "teacher" role can create courses.

---

4. Enroll in a Course

- **Endpoint:** POST /api/courses/:courseId/enroll **Description:** Enrolls a student in a specific course.
- **Parameters:** ○ courseId (Path parameter) - The course to enroll in.
- **Request Headers:**

  json

  {

  "Authorization": "Bearer jwt_token_here"

  }
- **Response:**

  json

  {

  "message": "Enrollment successful",

  "courseId": "course1",

  "userId": "student12345"

  }

User Management 1. Get User Profile

- **Endpoint:** GET /api/users/:userId
- **Description:** Retrieves the profile details of a user (student or teacher).
- **Parameters:** ○ userId (Path parameter) - The ID of the user to retrieve.
- **Request Headers:**

json

{

"Authorization": "Bearer jwt_token_here"

}

- **Response:**

json

{

"id": "user12345",

"name": "John Doe",

"email": "john.doe@example.com",

"role": "student",

"enrolledCourses": ["course1", "course2"]

}

2. Update User Profile

- **Endpoint:** PUT /api/users/:userId
- **Description:** Allows a user to update their profile information.
- **Parameters:** ○ userId (Path parameter) - The ID of the user to update.
- **Request Body:**

  json

  {

  "name": "Updated Name",

  "email": "updated.email@example.com"

  }

- **Response:**

  json

  {

  "message": "Profile updated successfully",

  "user": {

  "id": "user12345",

"name": "Updated Name",

"email": "updated.email@example.com"

}

}

Course Progress

1. Get Course Progress

- **Endpoint:** GET /api/courses/:courseId/progress
- **Description:** Retrieves the current progress of a student in a specific course.
- **Parameters:** ○ courseId (Path parameter) - The ID of the course.
- **Request Headers:**

  json

  {

  "Authorization": "Bearer jwt_token_here"

  }

- **Response:**

  json

  {

  "courseId": "course1",

  "userId": "student12345",

  "progress": 60, // percentage completion

  "completedLessons": 6,

  "totalLessons": 10

  }

Error Handling

All error responses follow the format:

json

{

"error": "Error description",

"code": "error_code"

}

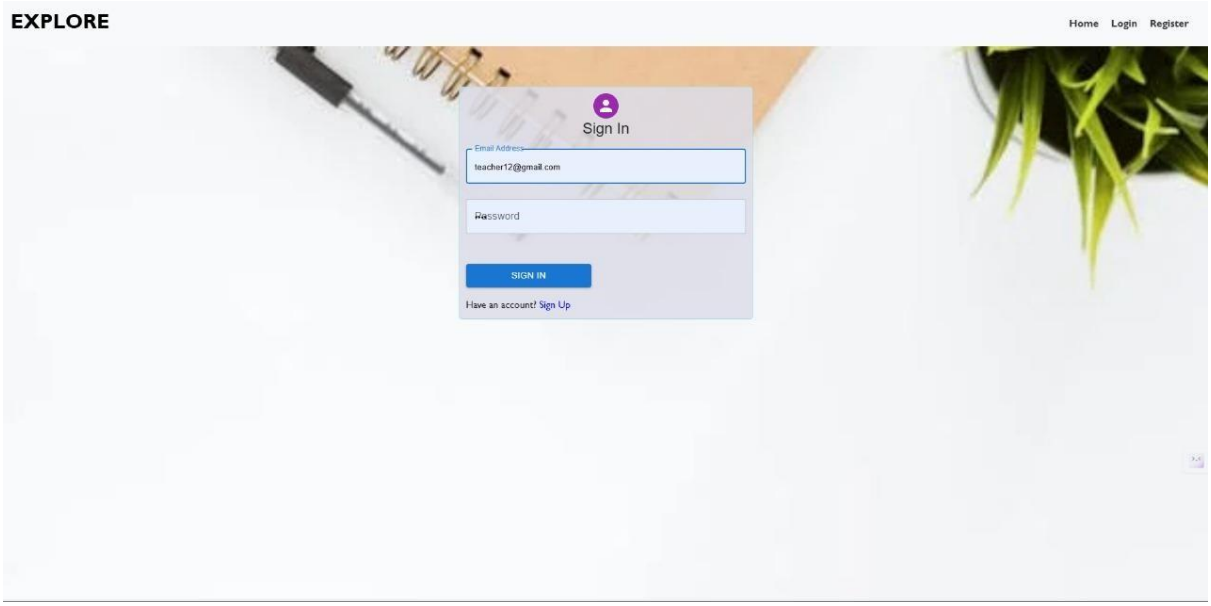
Authentication

• Token-based Authentication:

- Upon successful login, users receive a JSON Web Token (JWT), which is stored in local storage.

- Protected routes on the backend require a valid token, verified using middleware.

User Interface



Testing

• Testing: Unit and integration tests are implemented using tools like Jest for the frontend and Mocha for the backend.

Demo

https://drive.google.com/drive/folders/1chP3HHuRstCuj8w32lGPBA_Su0aeJKSy

Known Issues

1. **Scalability and Performance**
    - **Issue**: High traffic can lead to performance bottlenecks.
2. **Real-time Interaction**
    - **Issue**: Implementing real-time features like live chat or video conferencing.
3. **Video Streaming and File Management**
    - **Issue**: Handling large video files and bandwidth issues.

Future Enhancement

1. AI-Powered Personalization

- **Enhancement: Leverage machine learning algorithms to personalize learning experiences based on user behavior, preferences, and progress.**
- **Benefit: Adaptive learning paths, tailored recommendations for courses or resources, and increased user engagement.**

2. Gamification

- **Enhancement: Implement gamified features such as leaderboards, badges, rewards, and progress tracking to motivate learners.**
- **Benefit: Improved student retention and engagement by making learning more interactive and fun.**