**Full Stack Development with MERN**

**Frontend Development 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**

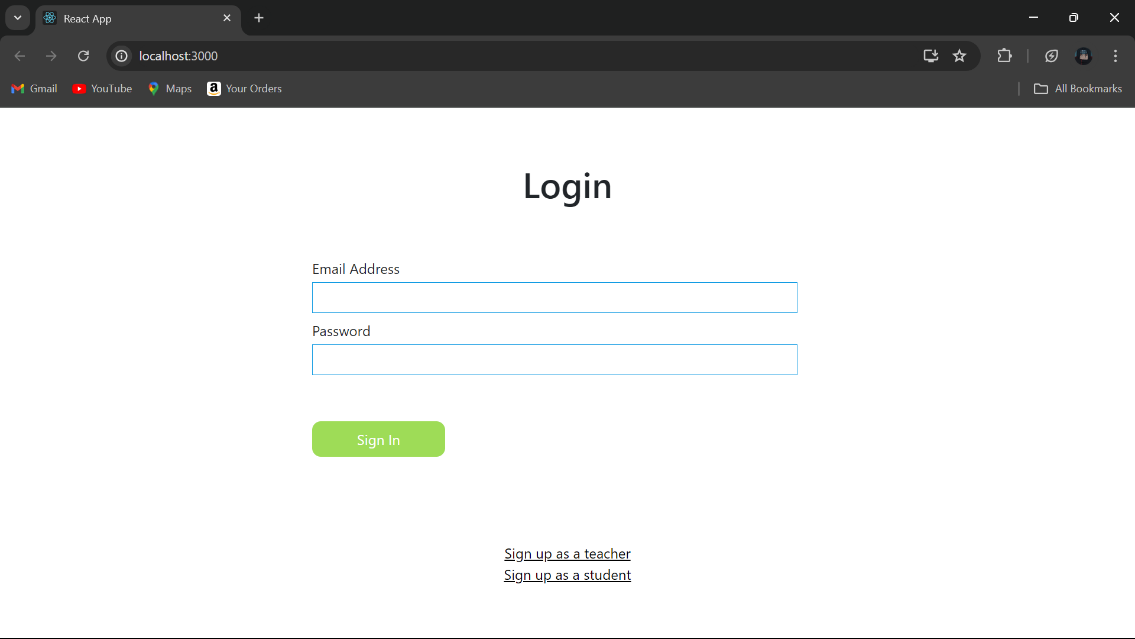
The objective of this report is to document the frontend development progress and key aspects of the user interface implementation for the Nexus Learn project.

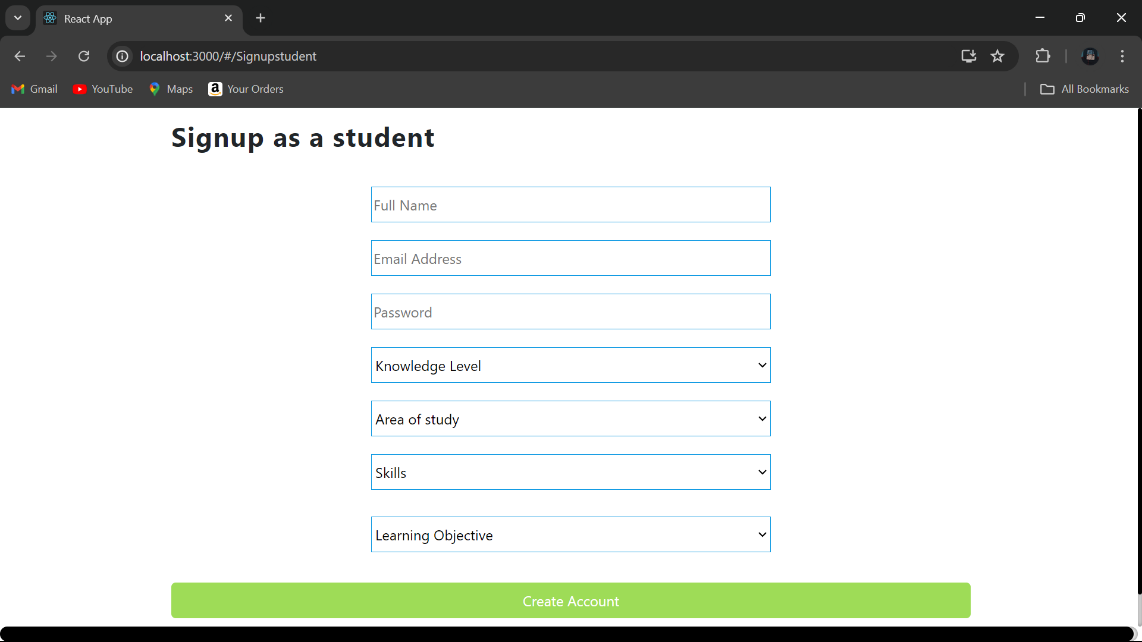
**Technologies Used**

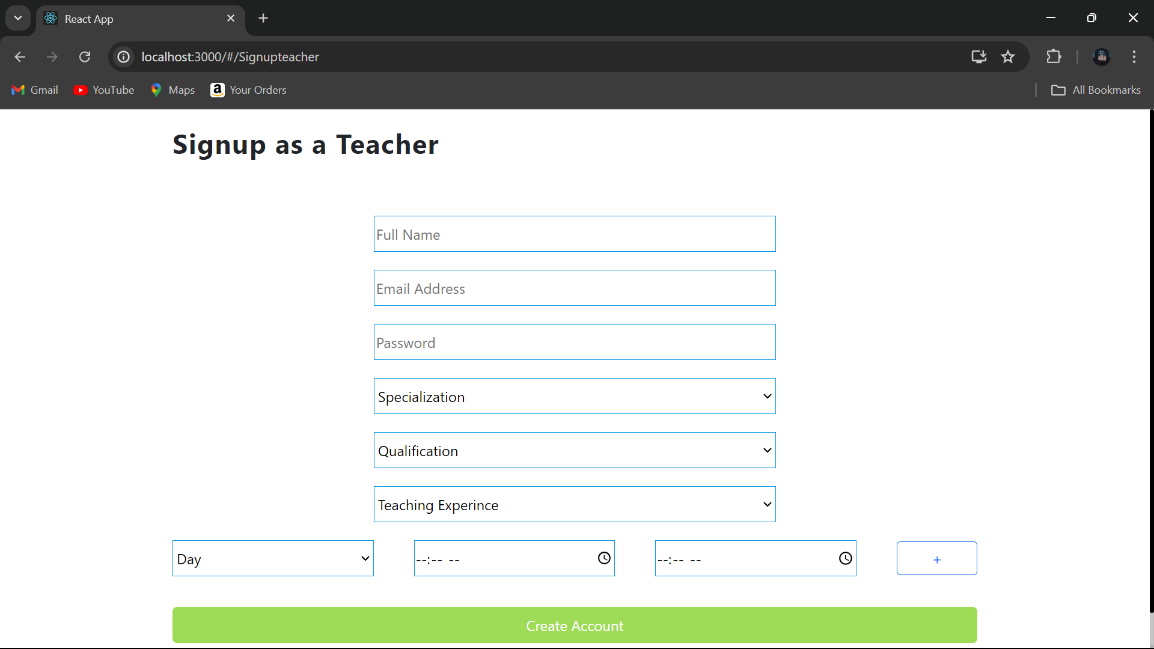
* **Frontend Framework:** React.js
* **State Management:** React hooks (useState, useEffect)
* **UI Framework/Libraries:** Bootstrap,Material UI
* **API Libraries:** Custom implementation using Axios
* **Testing Libraries:**
  + @testing-library/jest-dom
  + @testing-library/react
  + @testing-library/user-event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

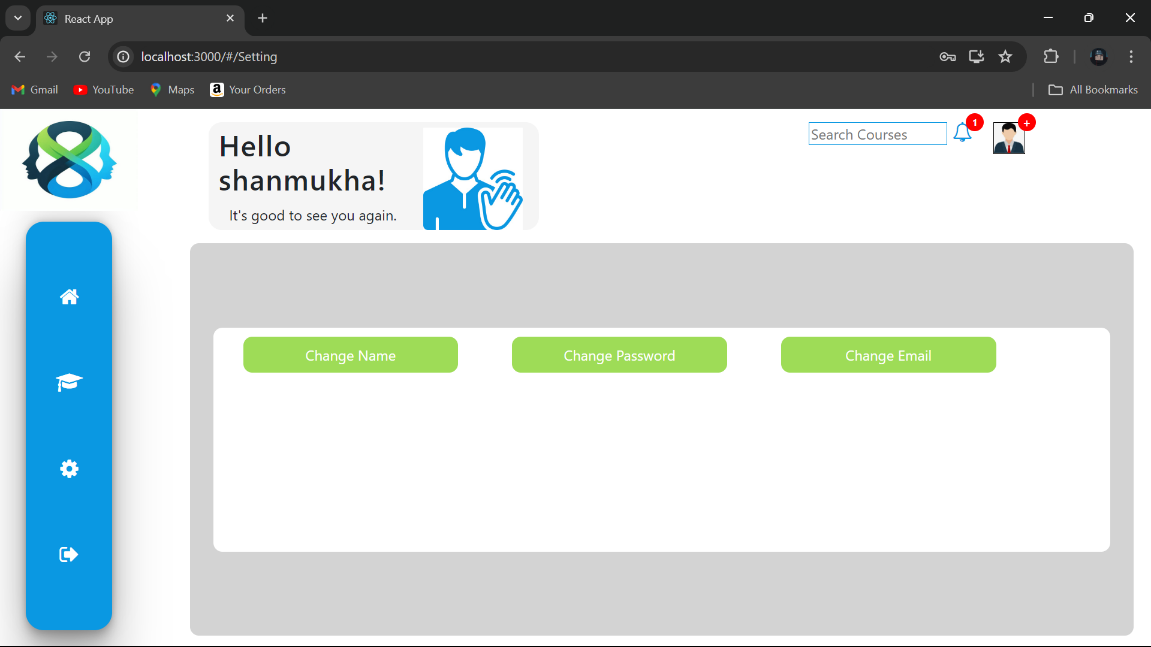
**Project Structure**

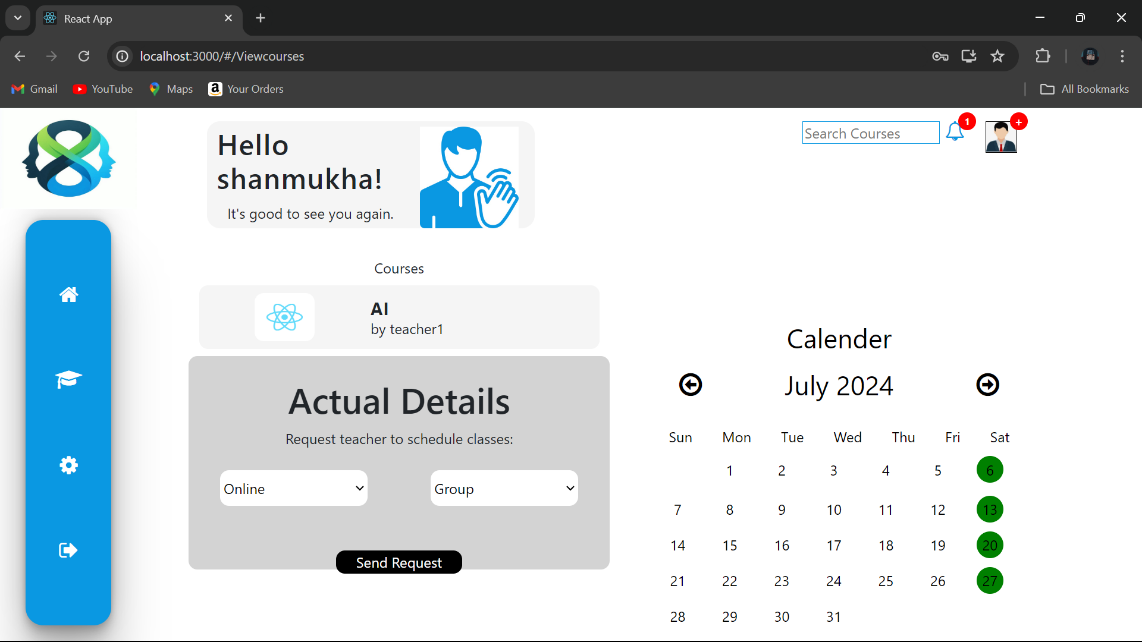
The Nexus Learn frontend project structure includes various directories and files to manage different aspects of the application.

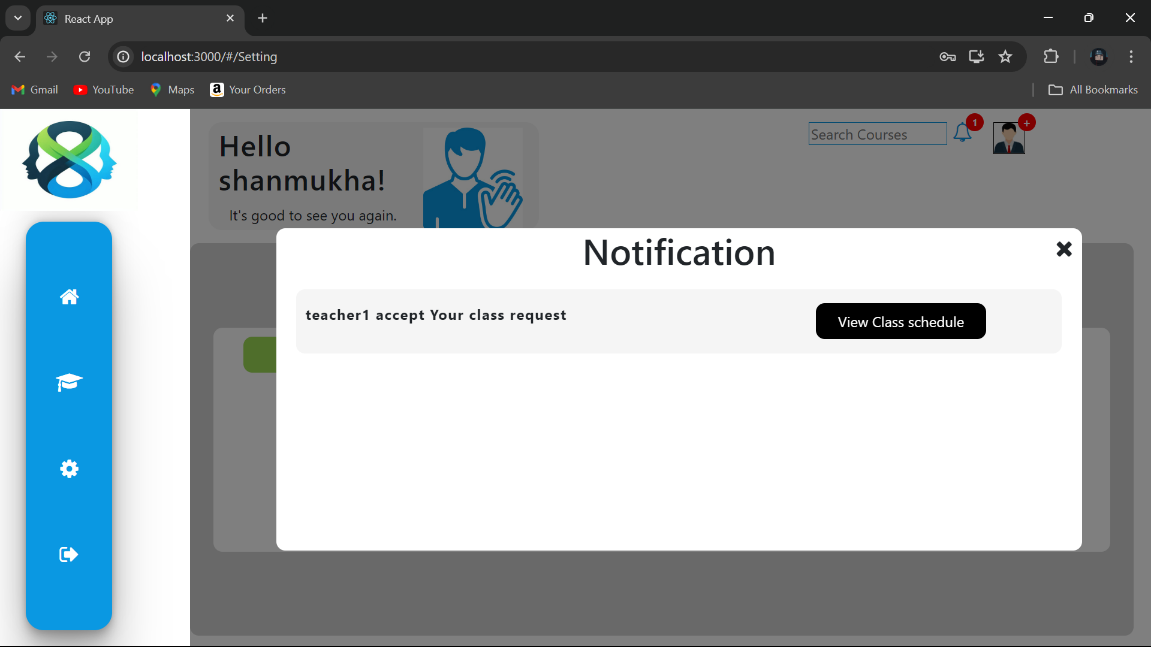


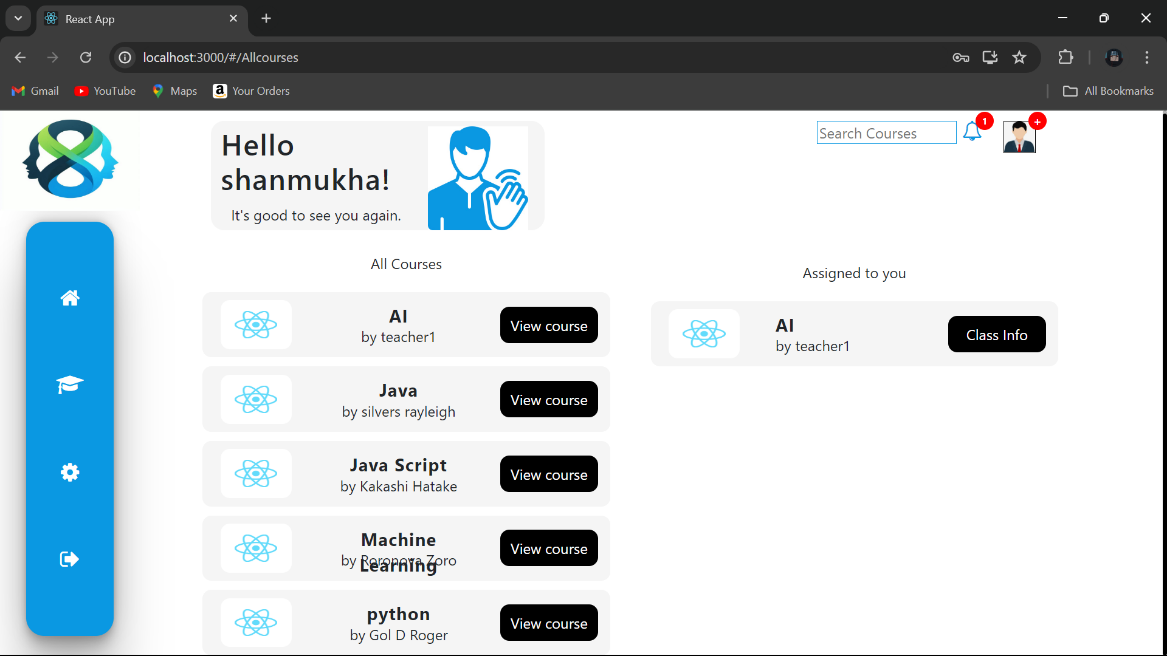


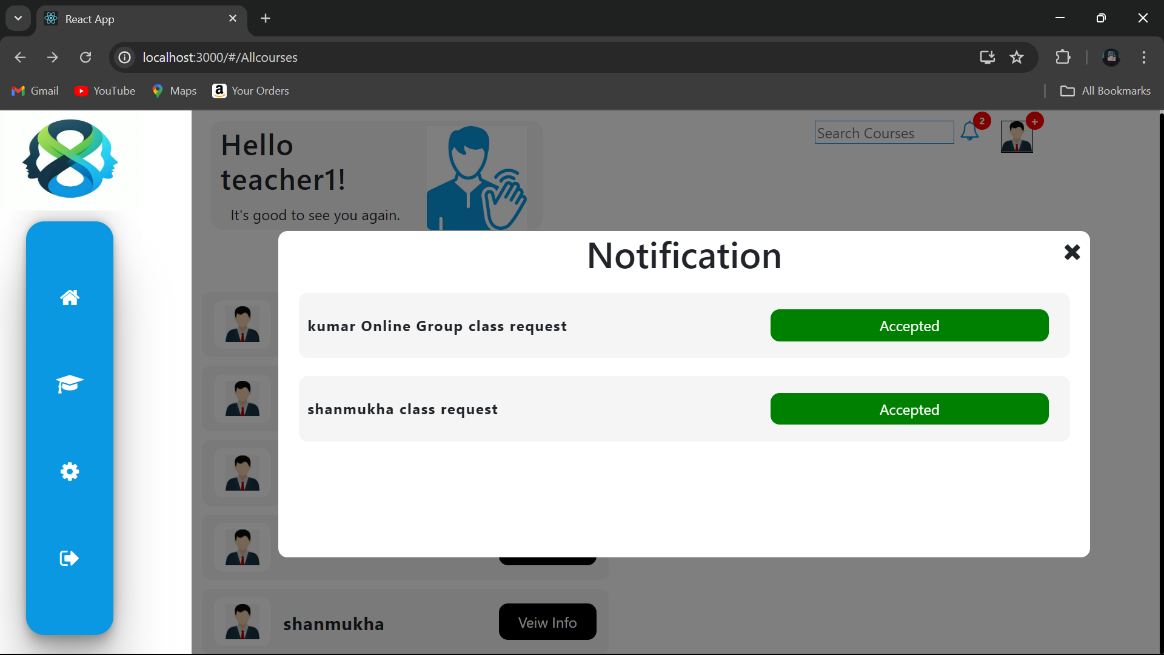


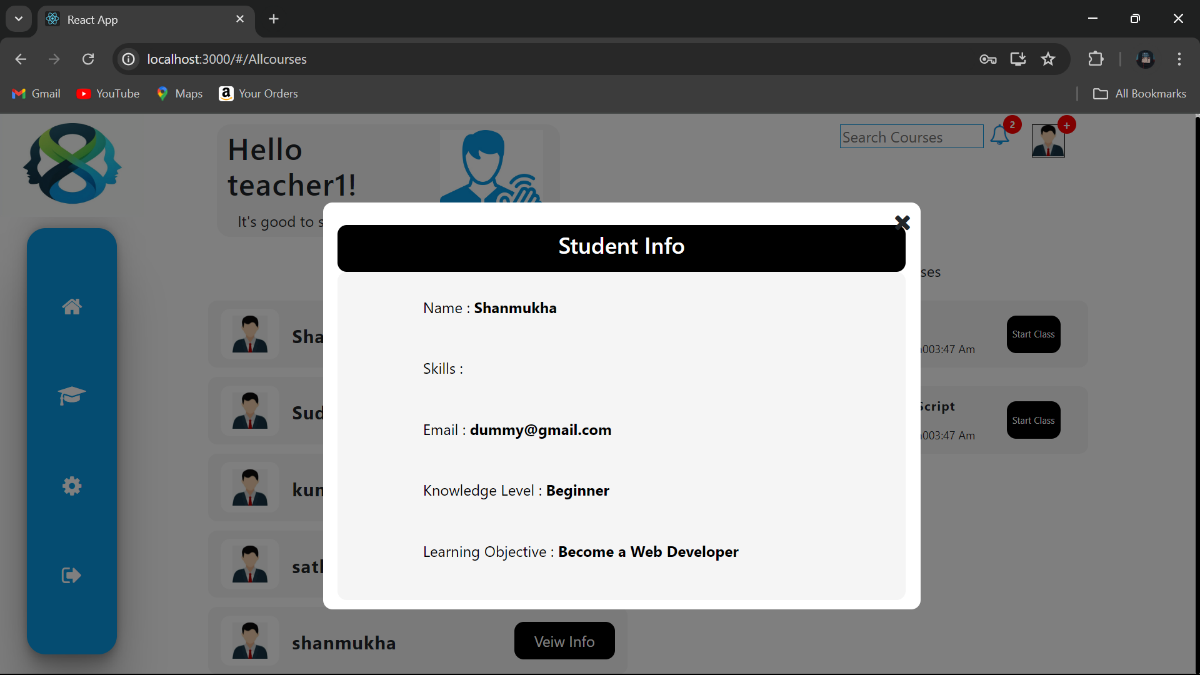


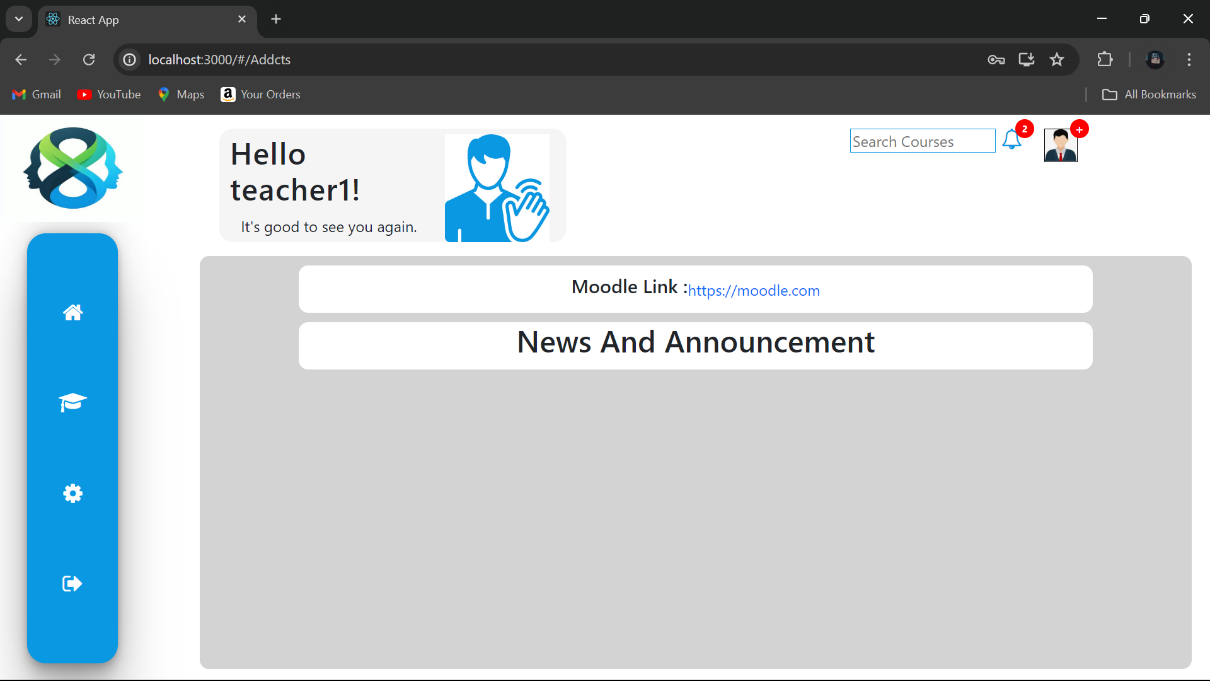


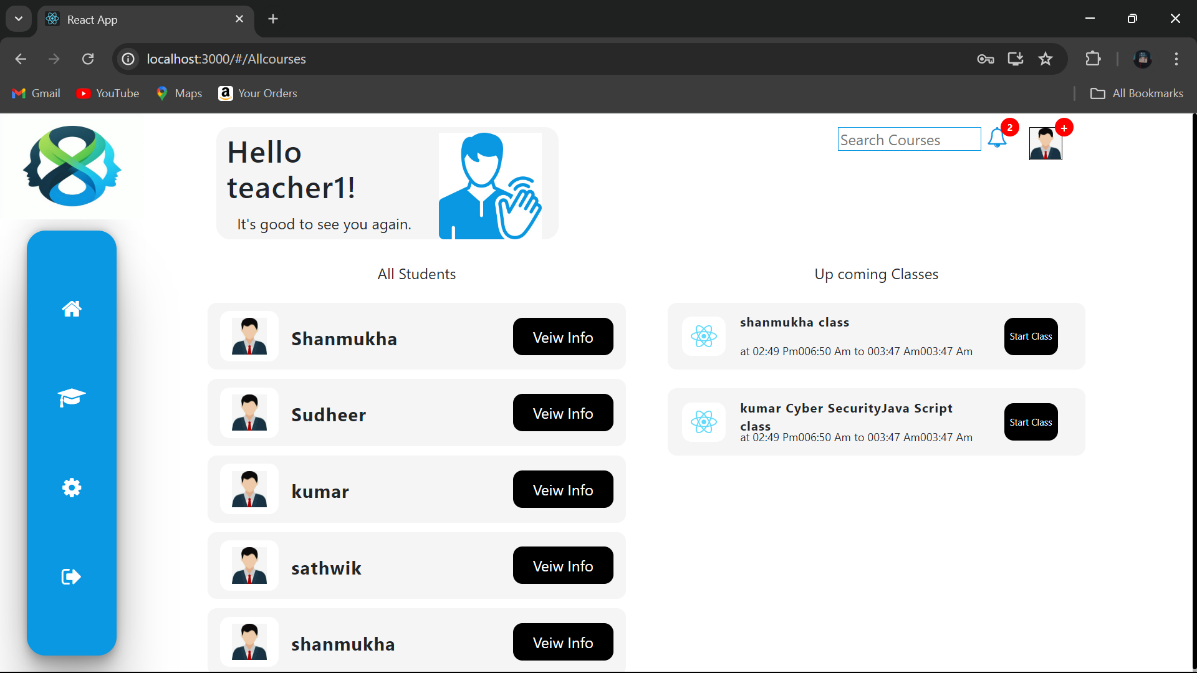


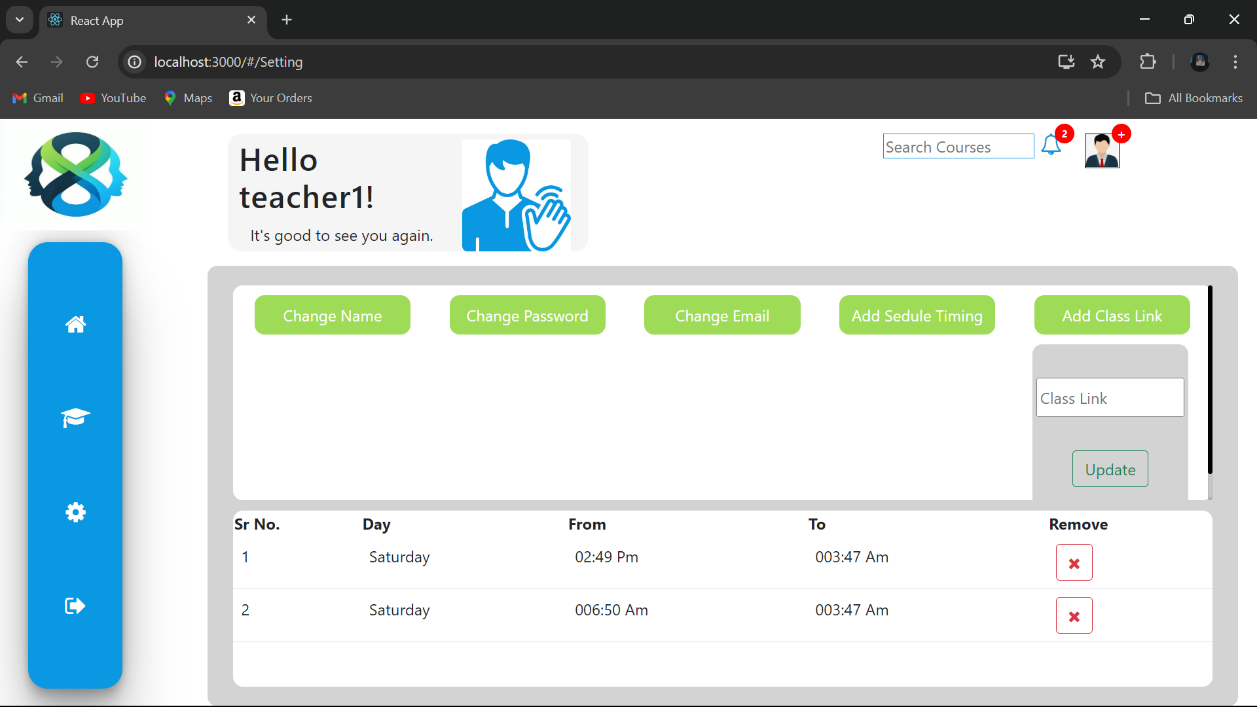












**Key Components**

1. **App.js**
   * Responsible for routing and main application layout.
2. **/admin**
   * **AdminDashboard.js**: Dashboard for admin to manage the platform.
   * **ManageCourses.js**: Component for adding, updating, and deleting courses.
   * **ManageUsers.js**: Component for overseeing users and their roles.
   * **ViewEnrollments.js**: Component to view and manage student enrollments.
   * **CourseStatistics.js**: Component to view course performance and statistics.
   * **adminapicall.js**: Handles API calls related to admin operations.
3. **/auth**
   * **AdminRoutes.js**: Handles protected routes specifically for admin.
   * **PrivateRoutes.js**: Handles protected routes for students and teachers.
   * **index.js**: Authentication helpers
4. **/core**
   * **CourseCard.js**: Component for displaying course details in a card format.
   * **CourseList.js**: Component to display a list of courses.
   * **SearchFilter.js**: Component for filtering and searching courses.
   * **CourseProgress.js**: Component to track and display student progress.
   * **Cart.js**: Component for managing the cart
   * **ImageHelper.js**: Handles image-related operations.
   * **coreapicalls.js**: General API call handlers for core functionalities.
   * **paymentHelper.js**: Handles payment-related operations.
5. **/student**
   * **StudentDashboard.js**: Main dashboard for students to view their courses and progress.
   * **EnrollCourse.js**: Component for enrolling in a new course.
   * **MyCourses.js**: Component to view and manage enrolled courses.
   * **CertificateDownload.js**: Component to download course completion certificates.
   * **SearchFilter.js**: Component for filtering and searching courses.
   * **studentapicall.js**: Handles API calls related to student operations.
6. **/teacher**
   * **TeacherDashboard.js**: Main dashboard for teachers to manage their courses.
   * **AddCourse.js**: Component for adding a new course.
   * **ManageSections.js**: Component for managing sections within a course.
   * **UpdateCourse.js**: Component for updating existing course details.
   * **DeleteCourse.js**: Component for deleting a course if no students are enrolled.
   * **teacherapicall.js**: Handles API calls related to teacher operations.
7. **/shared**
   * **Header.js**: Component for the application header.
   * **Footer.js**: Component for the application footer.
   * **Notification.js**: Component for displaying notifications and alerts.
   * **LoadingSpinner.js**: Component to show a loading spinner during data fetches.
8. **/services**
   * **authService.js**: Functions for handling authentication-related API calls.
   * **courseService.js**: Functions for managing course-related API calls.
   * **studentService.js**: Functions for handling student-related API calls.
   * **teacherService.js**: Functions for managing teacher-related API calls.
9. **index.js**
   * Entry point for the React application, rendering the App component.
10. **Routes.js**
    * Defines application routing, including routes for different user roles and page components.
11. **styles.css**
    * Global CSS styles for the application.

**Routing**

Routing is managed using React Router. Here are the main routes:

* **Home Route**

**/home**: Landing page of the application.

* + **Path:** /
  + **Component:** Home.js
* **/signup**: User signup page.

**Signup Route**

* + **Path:** /signup
  + **Component:** Signup.js
* **/signin**: User signin page.

**Signin Route**

* + **Path:** /signin
  + **Component:** Signin.js
* **/admin signin**: admin signin page
* **/instructor signin**: instructor signin page
* **Student Routes**
  + **Student Dashboard Route**

/dashboard: Dashboard displaying user data and statistics.

* + - **Path:** /student/dashboard
    - **Component:** StudentDashboard.js
  + /profile: User profile management.
  + **Enroll Course Route**
    - **Path:** /student/enroll
    - **Component:** EnrollCourse.js
  + **My Courses Route**
    - **Path:** /student/courses
    - **Component:** MyCourses.js
  + **Course Progress Route**

/courses: Course listing and details.

* + - /course/: Individual course page.
    - **Path:** /student/progress/:courseId
    - **Component:** CourseProgress.js
  + **Certificate Download Route**
    - **Path:** /student/certificate/:courseId
    - **Component:** CertificateDownload.js
* **Teacher Routes**
  + **Teacher Dashboard Route**
    - **Path:** /teacher/dashboard
    - **Component:** TeacherDashboard.js
  + **Add Course Route**
    - **Path:** /teacher/create/course
    - **Component:** AddCourse.js
  + **Manage Sections Route**
    - **Path:** /teacher/manage/sections/:courseId
    - **Component:** ManageSections.js
  + **Update Course Route**
    - **Path:** /teacher/update/course/:courseId
    - **Component:** UpdateCourse.js
  + **Delete Course Route**
    - **Path:** /teacher/delete/course/:courseId
    - **Component:** DeleteCourse.js
* **Admin Routes**
  + **Admin Dashboard Route**
    - **Path:** /admin/dashboard
    - **Component:** AdminDashboard.js
  + **Add Course Route**
    - **Path:** /admin/create/course
    - **Component:** AddCourse.js
  + **Manage Courses Route**
    - **Path:** /admin/courses
    - **Component:** ManageCourses.js
  + **Manage Users Route**
    - **Path:** /admin/users
    - **Component:** ManageUsers.js
  + **View Enrollments Route**
    - **Path:** /admin/enrollments
    - **Component:** ViewEnrollments.js
  + **Course Statistics Route**
    - **Path:** /admin/statistics
    - **Component:** CourseStatistics.js
* **Shared Routes**
  + **Cart Route**
    - **Path:** /cart
    - **Component:** Cart.js

**State Management**

State management is achieved using React's built-in hooks such as useState and useEffect. If applicable, you can also mention additional state management libraries like Redux or Context API.

**State Management Approach:**

* **useState**: Used to create and manage state variables in functional components.
* **useEffect**: Used to handle side effects in functional components, such as fetching data from APIs or subscribing to events.

**Integration with Backend**

The frontend communicates with the backend APIs hosted on [backend URL]. Key endpoints include:

* **Courses**
  + **GET** /api/courses - Retrieves a list of all courses.
  + **GET** /api/course/{courseId} - Retrieves data for a specific course.
  + **POST** /api/course/create - Adds a new course.
  + **PUT** /api/course/{courseId} - Updates a specific course.
  + **DELETE** /api/course/{courseId} - Deletes a specific course if no students are enrolled.
* **Sections**
  + **GET** /api/section/{sectionId} - Retrieves data for a specific section.
  + **POST** /api/section/create - Adds a new section to a course.
  + **PUT** /api/section/{sectionId} - Updates a specific section.
  + **DELETE** /api/section/{sectionId} - Deletes a specific section.
* **Users**
  + **POST** /api/user/signup - Handles user signup.
  + **POST** /api/user/signin - Handles user authentication.
  + **GET** /api/user/{userId} - Retrieves data for a specific user.
  + **PUT** /api/user/{userId} - Updates a specific user's data.
  + **DELETE** /api/user/{userId} - Deletes a specific user account.
* **Student Enrollments**
  + **POST** /api/enroll - Enrolls a student in a course.
  + **GET** /api/enrollments/{studentId} - Retrieves a list of courses a student is enrolled in.
  + **GET** /api/enrollment/{enrollmentId} - Retrieves data for a specific enrollment.
  + **DELETE** /api/enrollment/{enrollmentId} - Unenrolls a student from a course.
* **Course Progress**
  + **GET** /api/progress/{enrollmentId} - Retrieves the progress of a student in a specific course.
  + **PUT** /api/progress/{enrollmentId} - Updates the progress of a student in a course.
* **Certificates**
  + **GET** /api/certificate/{enrollmentId} - Retrieves a course completion certificate for a student.
* **Admin Operations**
  + **GET** /api/admin/users - Retrieves a list of all users (for admins).
  + **GET** /api/admin/courses - Retrieves a list of all courses (for admins).
  + **GET** /api/admin/enrollments - Retrieves a list of all student enrollments (for admins).

**Notes:**

* **Authentication & Authorization**: Ensure that the endpoints for sensitive operations are protected with proper authentication and authorization mechanisms. Use JWT or similar methods to secure these endpoints.
* **Error Handling**: Implement proper error handling on both the frontend and backend to manage and display errors gracefully.
* **Data Validation**: Validate data on the backend to ensure that only correct and expected data is processed.

**User Interface (UI) Design**

The UI design follows the following principles:

* **Clarity**: The interface is designed to be clear and understandable. Each element is placed with purpose, ensuring that users can easily understand and navigate the application without confusion.
* **Simplicity**: The design avoids unnecessary complexity. By focusing on essential features and functionality, the interface remains clean and uncluttered.
* **Consistency**: Consistent design elements are used throughout the application. Colours, fonts, and button styles remain uniform.
* **Responsiveness**: The UI is designed to be fully responsive, ensuring that the application works seamlessly on various devices and screen sizes.

The application uses Bootstrap and Material UI as the primary UI frameworks. Their pre-styled components and responsive grid systems help ensure a consistent and mobile-friendly layout across different screen sizes and devices.

**Third-Party Integrations**

* **React Router**: Used for handling client-side routing.
* **Bootstrap**: Used for styling and layout.
* **Material UI**: Used for additional styling and component enhancements.