**Online College Admission Portal**

**Overview:**

Traditional college admission processes are often slow, error-prone, and rely heavily on paperwork. This leads to frustration for both applicants and administrative staff. A modern solution is needed to create a user-friendly online admission portal. This portal should simplify application procedures, automate document handling, and improve communication between applicants and administrators. By doing so, it aims to make the admission process smoother, more transparent, and secure for everyone involved.

**Users of the System:**

**1.    Students**:

**Description**: Individuals seeking admission to the educational institution.

**Role**:

·      Create and manage their student accounts.

·      Check the number of seats in course and then apply.

·      Fill out and submit admission application forms.

·      They can pay the fees online.

2.    **Administrators**:

**Description**: College staff responsible for managing and overseeing the admission process.

**Roles**:

·      Access and navigate the admin dashboard for an overview of admission statistics.

·      Manage the seats per course.

·      Review and evaluate submitted applications.

·      View Student Payment History

**Functional Requirements:**

**·**Build a portal that enables students can get College Admission online.

·      The students can add/edit/view/delete admission.

·      The students can view institute details.

·      The students can pay their fees

·      The admin can view student payment history

·      The admin can manage the number of seats per course

·      The admin can add/edit/delete/view courses.

·      The admin can add/edit/delete/view institutes.

**Non-Functional Requirements:**

1. Security: The system must implement robust security measures to protect user data, including user authentication, secure data storage, and encrypted data transmission.
2. Scalability: The system should be designed to handle an increasing number of course listings, students, and users.
3. Usability: The user interface should be intuitive and user-friendly, with responsive design for mobile and desktop users.
4. Availability: The system should be available 24/7 with minimal downtime for maintenance.
5. Logging and Auditing: Support logging and auditing of system activities for monitoring and troubleshooting.

**Application Flow:**

**Student side:**

The application flow for the online admission portal begins with user registration, where prospective students create accounts by providing personal information. Upon logging in, users access the user dashboard and view the available institute and course details. Student check the number of seats available in course and then apply. Student cannot apply for the course that has no seats. Student can select a course and proceed to fill out and submit application forms, which include personal and academic details. Throughout the process, applicants can track the real-time status of their applications, viewing updates such as " Application submitted," “Pending”, " Accepted," or " Rejected."

**Admin side:**

The administrative flow within the online admission portal begins with administrators accessing the admin dashboard, providing a comprehensive overview of institutes, courses. From the dashboard, administrators proceed to the application review module, where they evaluate and assess submitted applications.

Administrator will manage the number of seats in the course. Administrators have the ability to mark applications as accepted, rejected, or pending based on their evaluations.

Admins can also edit/delete institute, courses. Once the admission process is complete, applicants receive their final admission status.

**Abstract**

The Student Admission Management System is a comprehensive web-based application designed to streamline and enhance the admission process for educational institutions. This innovative platform caters to two main user roles: Students and Administrators.

**Modules of the Application:**

**ADMIN:**

* Register
* Login
* Dashboard
  + Institutes
  + Courses available with available seat details
  + Payment details

**STUDENT:**

* Register
* Login
* Dashboard
  + Institutes menu
  + Courses menu
  + Seats available
  + Payment History
  + Make payment

**Technology Stack**

**Front End**

React, HTML, CSS, JavaScript

**Back End**

Java, Spring Boot, MySQL Database

**Authentication**

JWT for User Authentication

**Application assumptions**:

1.    The login page should be the first page rendered when the application loads.

2.    Manual routing should be restricted by using Auth Guard by implementing the canActivate interface..

3.    Unless logged into the system, the user cannot navigate to any other pages.

4.    Logging out must again redirect to the login page.

5.    To navigate to the admin side, you can store a user type as admin in the database with a username and password as admin.

6.    Design forgot password and forgot email buttons in login page.

**Validations**:

1. Basic email validation should be performed.

      2. Basic mobile number validation should be performed.

      3. Basic password should be performed.

**Client-Side Validation:**

Implement client-side validation using HTML5 attributes and JavaScript to validate user input before making API requests.

Provide immediate feedback to users for invalid input, such as displaying error messages near the input fields.

**Server-Side Validation:**

Implement server-side validation in the controllers to ensure data integrity.

Validate user input and API responses to prevent unexpected or malicious data from affecting the application.

Return appropriate validation error messages to the user interface for any validation failures.

**Exception Handling**

Implement exception handling mechanisms in the controllers to gracefully handle errors and exceptions.

Define custom exception classes for different error scenarios, such as API communication errors or database errors.

Log exceptions for debugging purposes while presenting user-friendly error messages to users. Record all the exceptions and errors handled store in separate table “ErrorLogs”.

**Error Pages:**

**Create custom error pages for different HTTP status codes (e.g., 404** Not Found, **500** Internal Server Error) to provide a consistent and user-friendly error experience.

Ensure that error pages contain helpful information and guidance for users.

Thus, create a reliable and user-friendly web application that not only meets user expectations but also provides a robust and secure experience, even when faced with unexpected situations.

**Project Tasks:**

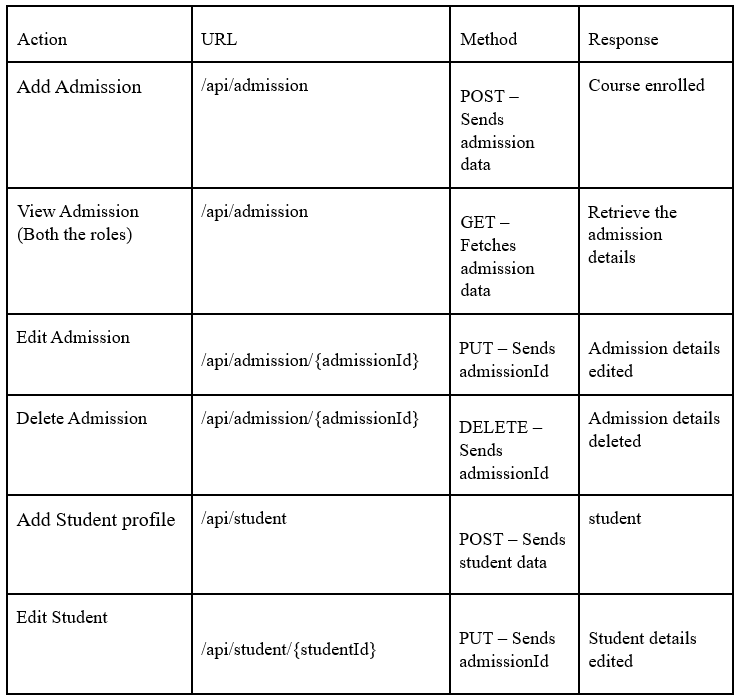
1. API Endpoints:

**Admin Role:**

* **Accessible API:** "/api/admin"
* **Description:** This API endpoint is reserved for administrators and provides access to functionalities such as the admin dashboard, application review. Only users with **ADMIN** privileges should be able to access this endpoint.

**Admin Side:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **URL** | **Method** | **Response** |
| User Login (Both Roles) | /auth/login | POST – Send emailId and password | Return Token userId,userRole and username |
| User Register (Both Roles) | /auth/register | POST – Send User Data | Returns true/false |
| Add Institute | /api/admin/institutes/add | POST – Send Institute Data | Institute added |
| View Institutes  (Both the Roles) | /api/viewinstitutes | GET – Fetch Institute data | Retrieve all institute data |
| Edit Institute | /api/admin/institute/{instituteid} | PUT -Sends Instituteid | Institute edited |
| Delete Institute | /api/admin/institute/{instituteid} | DELETE – Sends Instituteid | Institute deleted |
| Add Courses | /api/admin/{instituteid}/course | POST – Send Course Data | Course Added |
| View Courses  (Both the Roles) | /api/admin/courses | GET – Fetch all Courses | Retrieve all Courses |
| View Courses By  Instituteid  (Both the courses) | /api/admin/{instituteid}/course | GET -Fetch all Courses by Instituteid | Retrieve all the course based on instituteid |
| Edit Courses | /api/admin/institute/{instituteid}/course/{courseid} | PUT- Send courseid and instituteid | Course Edited |
| Delete Course | /api/admin/institute/{instituteid}/course/{courseid} | DELETE - Send courseid and instituteid | Course Deleted based on courseid |

**Student Side:   
  
**

**Backend Requirements:**

**Model Class:**

**|**Ø **User**:

This class stores the user type (admin or the customer) and all user information.

§ email: String

§ userId: Long

§  password: String

§  username: String

§  mobileNumber: String

§  userRole: String

Ø **Courses**:

This class stores the details of the course

§ courseId: Long

§ courseName: String

§ courseDescription: String

§ courseDuration: String

§ fess: Double

§ noOfSeats: int

§ Institute institute; (ManyToOne)

Ø **Institute**:

This class stores the details of the Institute or College

§ instituteId: Long

§ instituteName: String

§ instituteDescription: String

§ instituteAddress: String

§ mobile: String

§ email: String

§ noOfCoursesAvailable: Long

§ List<Courses> course : (OneToMany)

Ø **Student**:

This class stores the details of the students.

§ studentId: Long

§ User: user( OneToOne)

§ name: String

§ dob: Date

§ gender: String

§ motherName: String

§ fatherName: string

§ nationality: String

§ age: Long

§ address: String

§ mobile: String

§ marksSSLC: int

§ marksHSC: int

§ marksDiploma: int

§ eligibility: string

Ø **Admission**:

This class stores the details of the admission.

§ admissionId: long

§ List<Course>: course

§ status: String

§ Student: student

Ø **Payment**:

This class stores the details of the admission.

§ paymentId: long

§ status: String

§ amountPaid: Double

§ paymentDate: Date

§ modeOfPayment: String

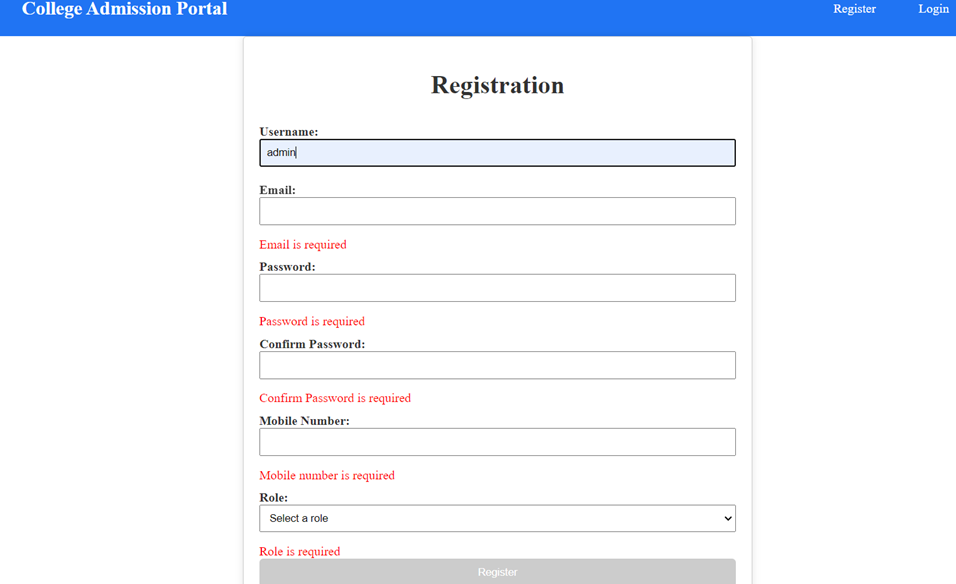
§ admission: Admission (OneToOne)

**Note**:.

Create a folder “components” inside src to store all the components.

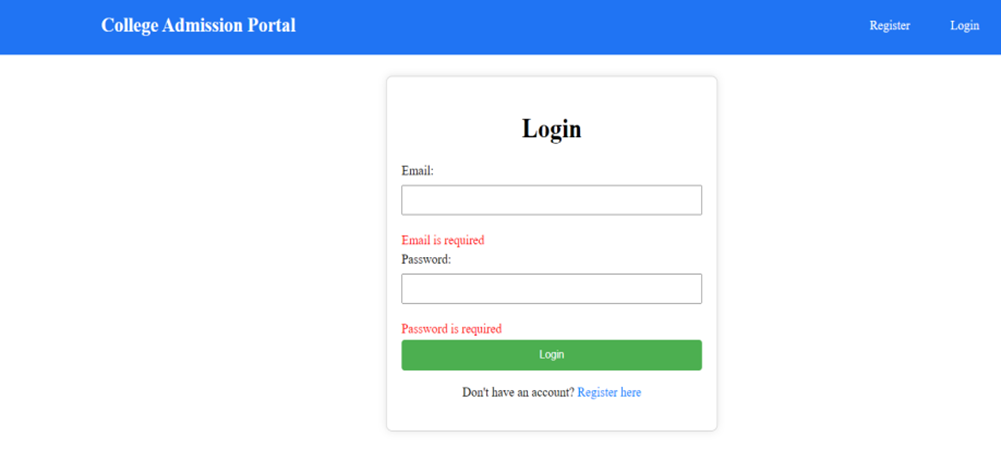
**FRONTEND SCREENSHOTS**

**STUDENT SIDE:**

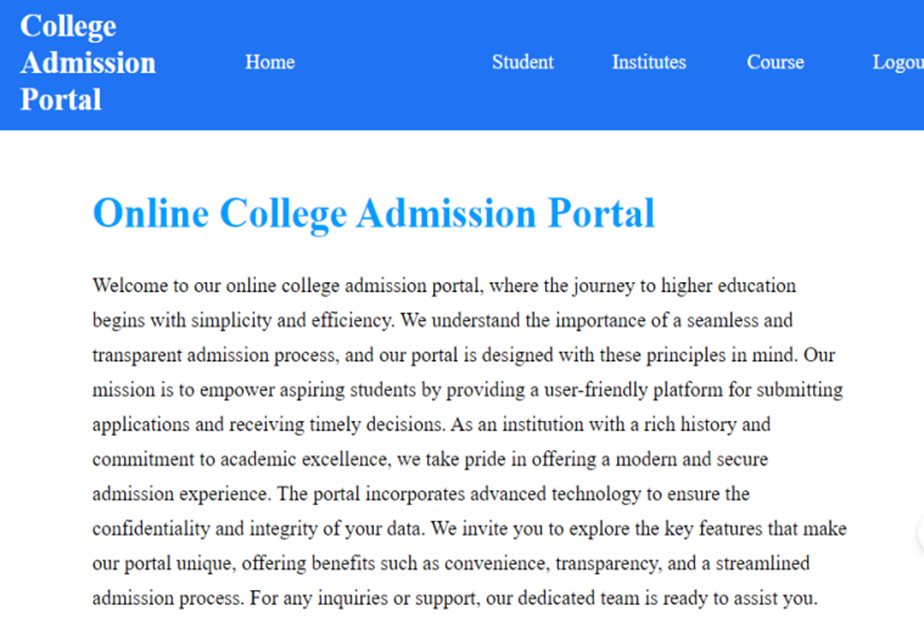
**Register Page  
  
**

1.    **Login**: Design a login page component inside the auth where the existing students can log in using the registered email id and password.

**Login Page**

****

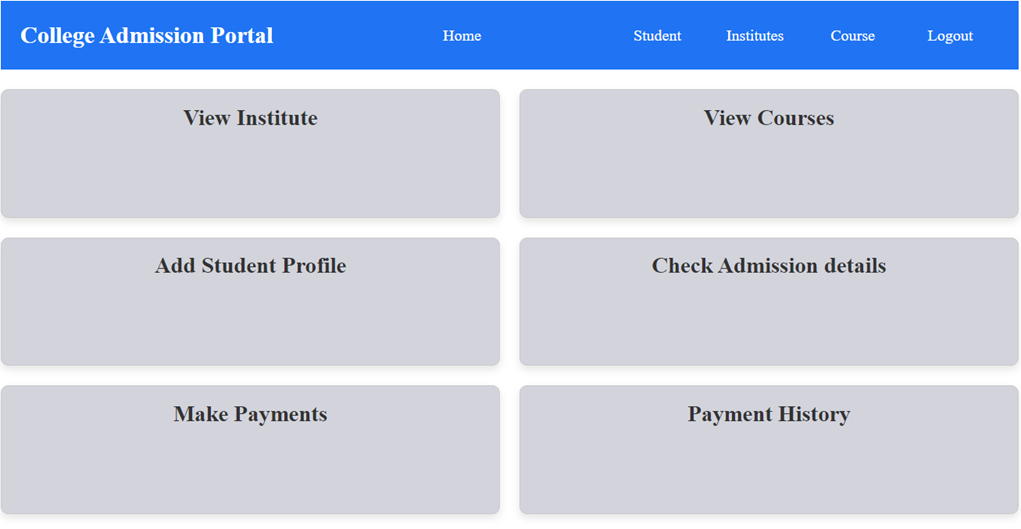
**2.       Home Component:** Design a component



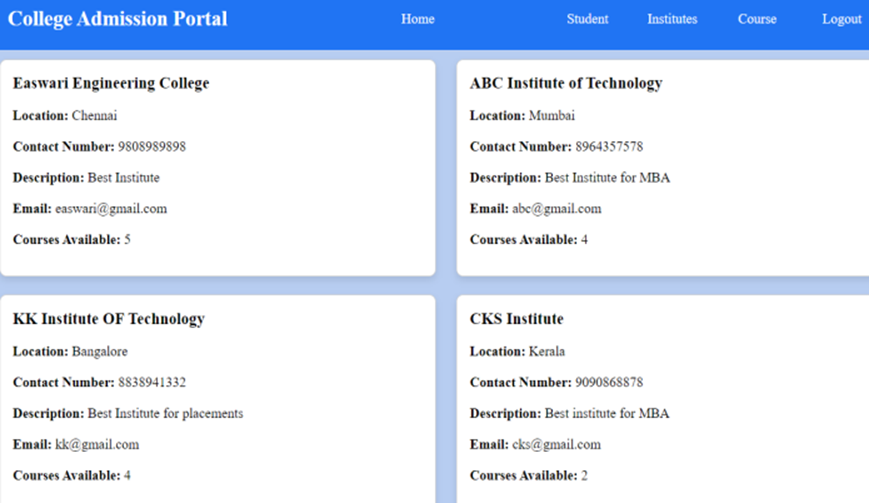
**STUDENT**

**3.       Student Dashboard:**

Design a page for student dashboard

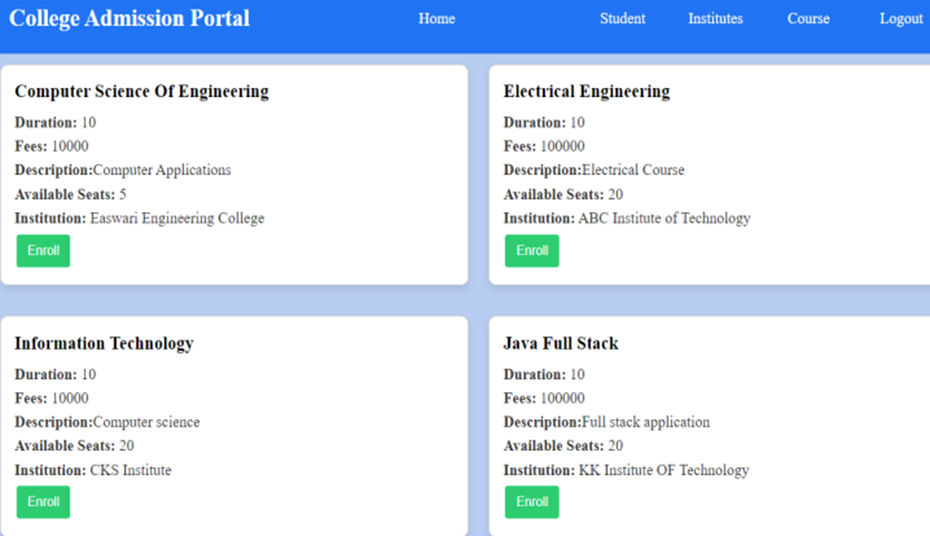


4.    **View Institute**: Design a component

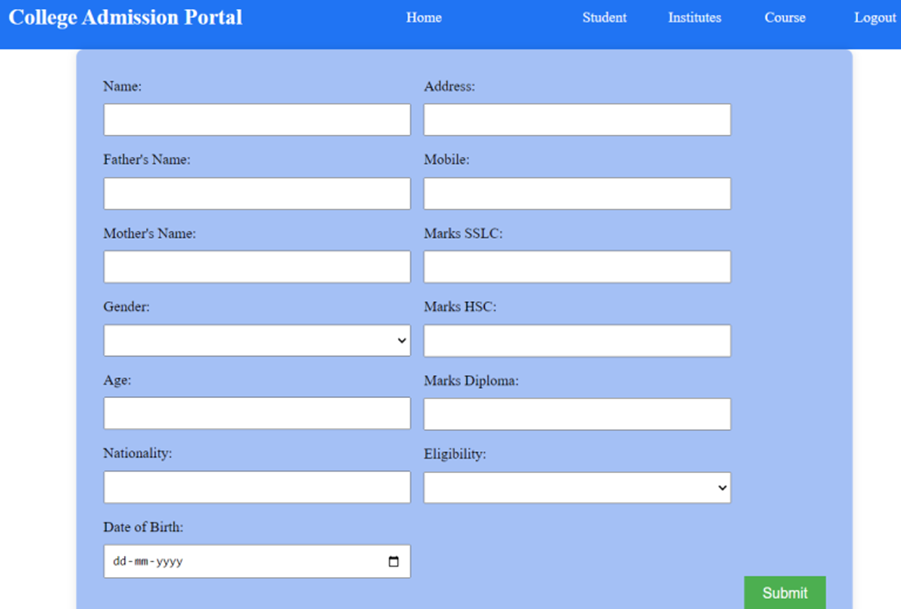


**5.     View Course**: Design a component

**List of Course Available Page:**

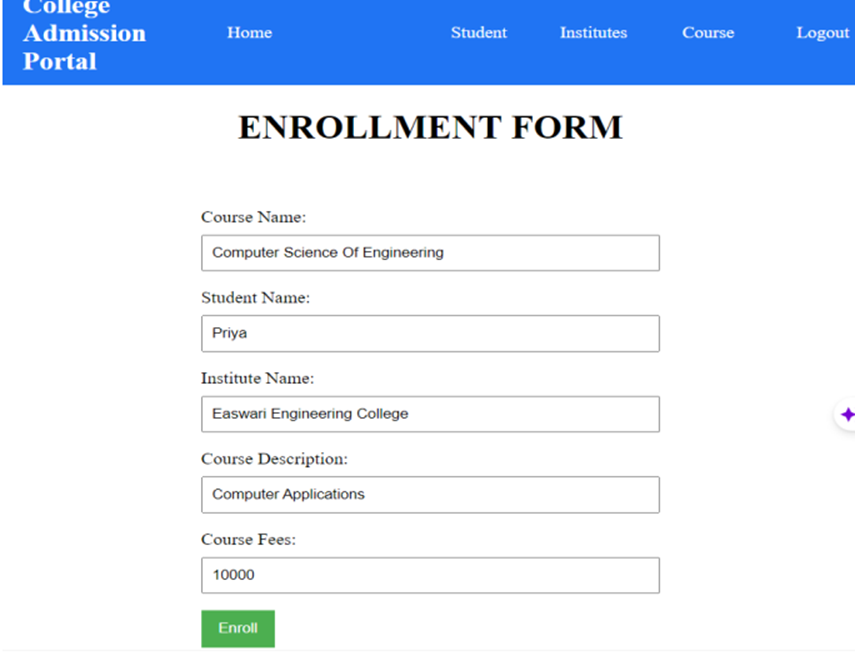


1. **Student profile form page:**  Create a component name add-student for student profile.



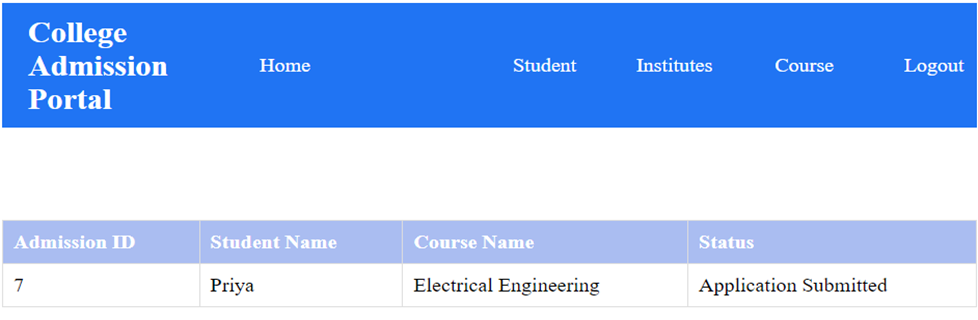
**7.       Add-Admission Page:**

Design a page for enrolment form

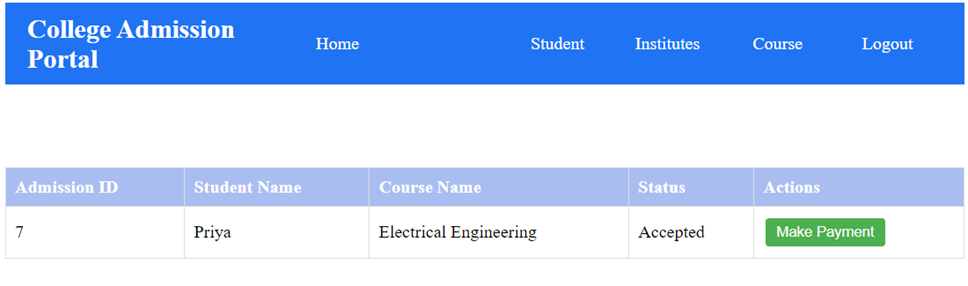


**8.       View-Admission Page:**

**Design a component for view Admission status**

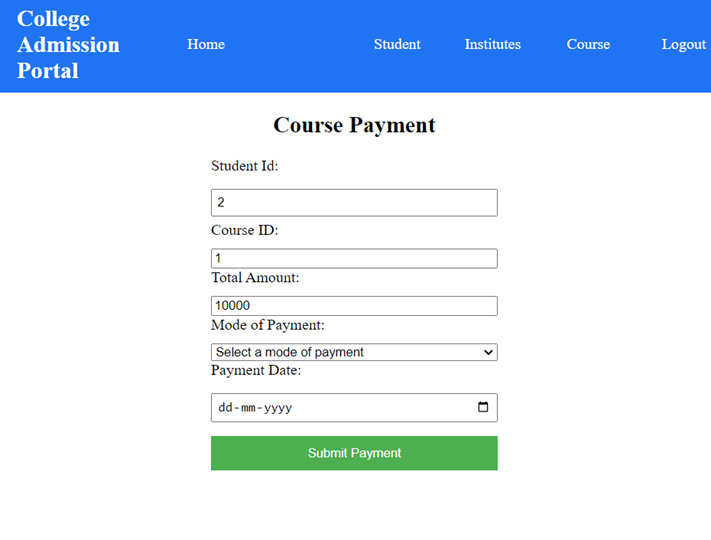


**Once the status is “Approved” by ADMIN you should get a make payment button and route it to payment page.**



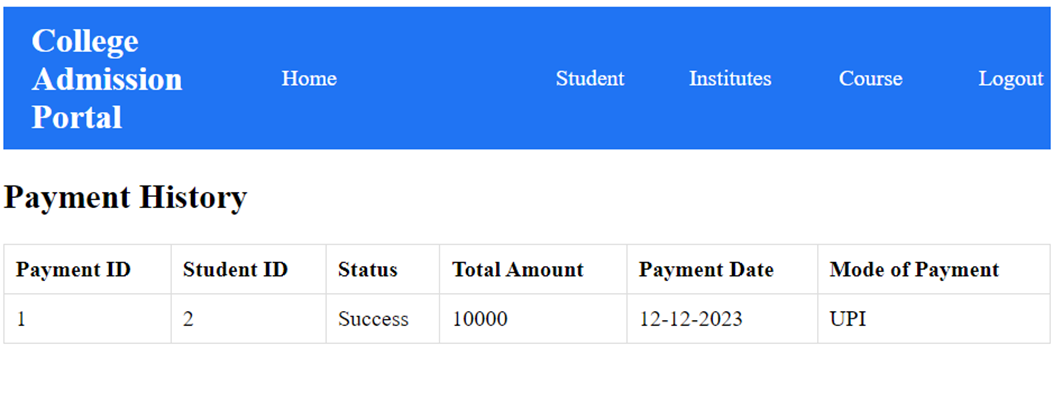
**9.       Add-payment page**

Design a page for payments



10.   **Payment-History page**

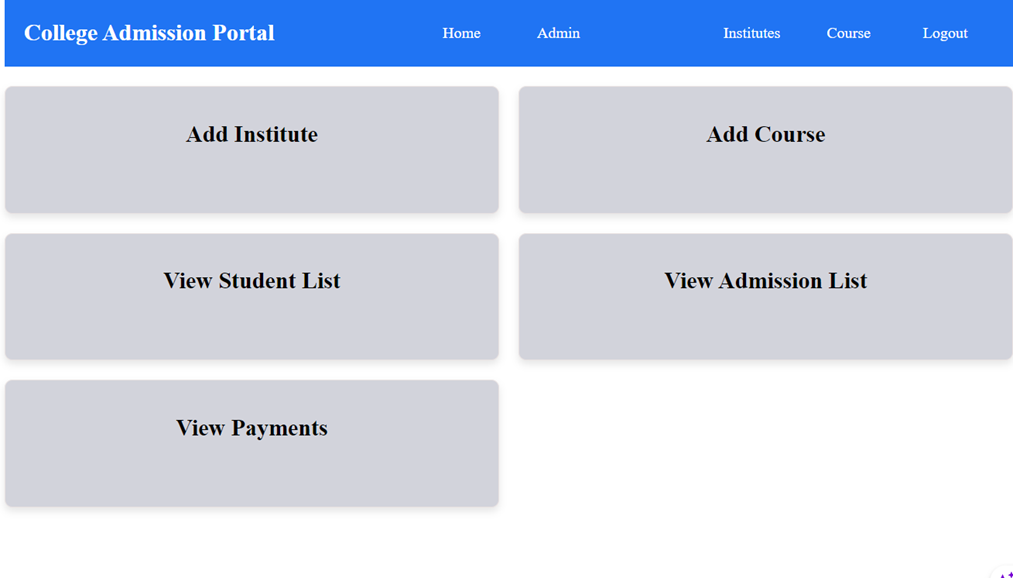
Design a page for view-payment



**Admin:**

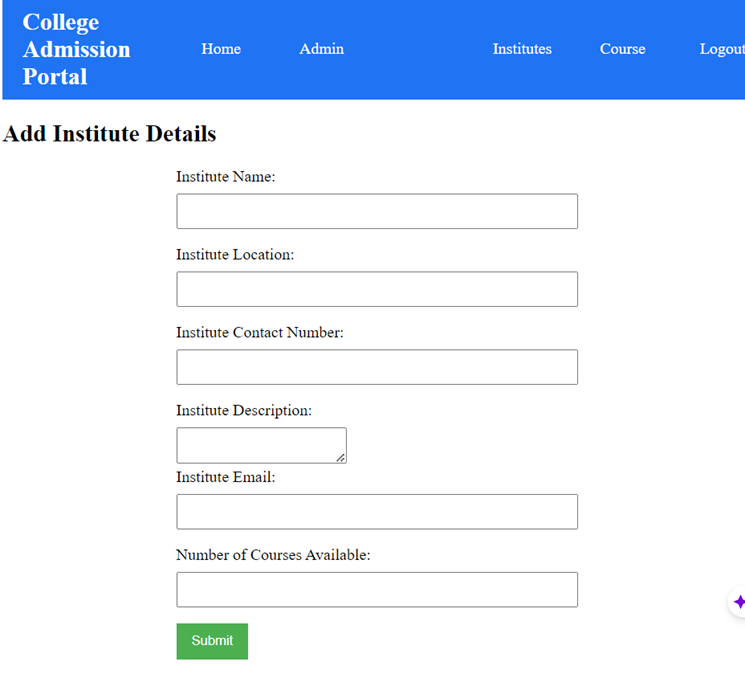
**11.   Admin Dashboard:**

Design a page for admin dashboard



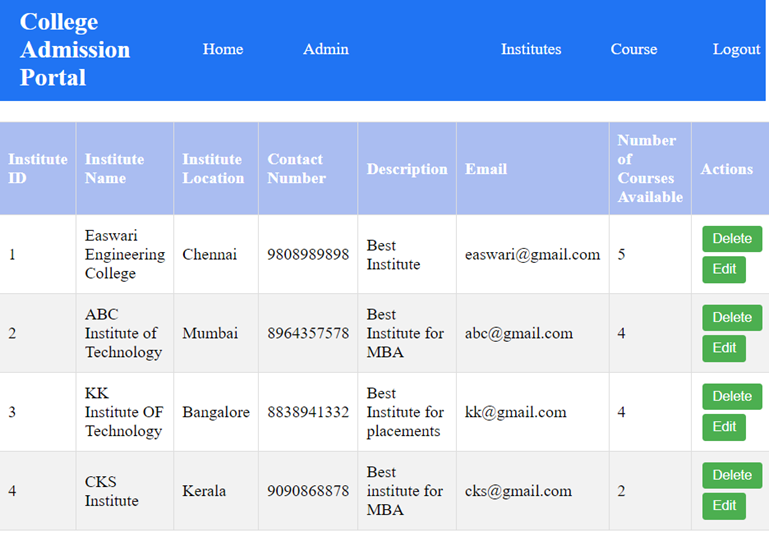
12. **Add-Institute**: Design a component

**Institite add page**



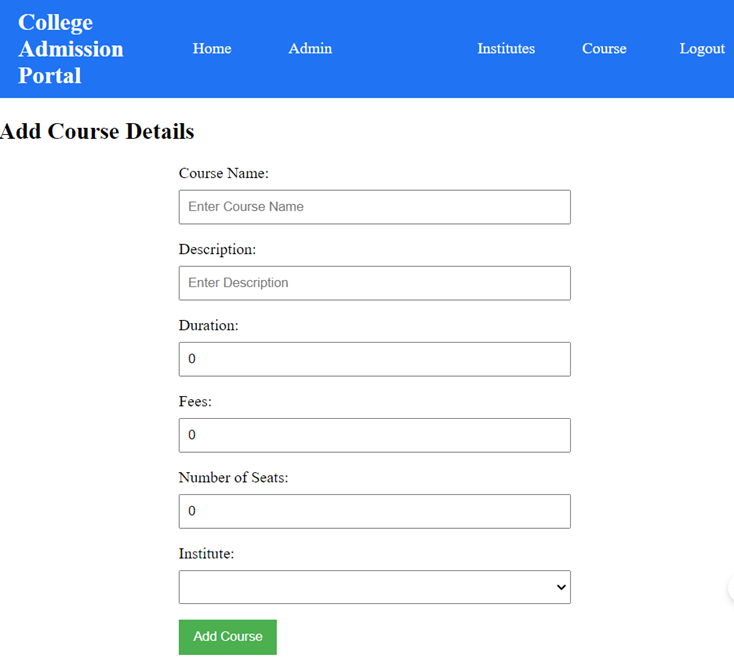
**13. View Institute Page**

**Design a page for view-institute**

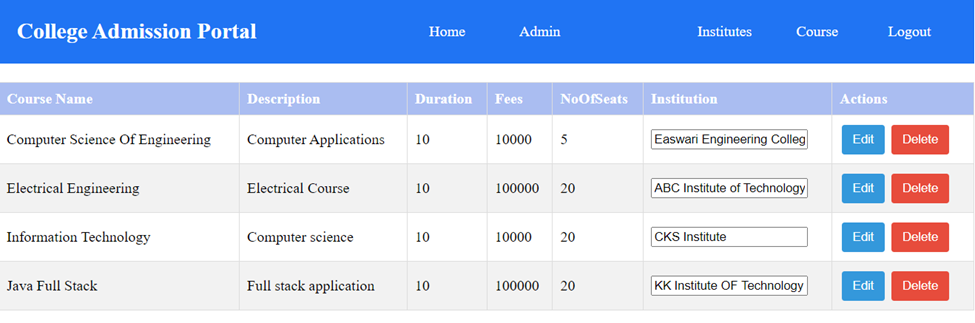
****

14. **Add Course**: Design a component

**AddCourse Page**

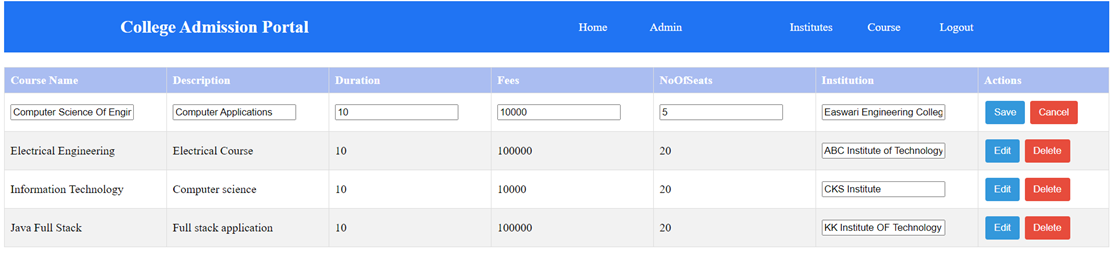


15.   **View Course Page:** Design a component for view-course page



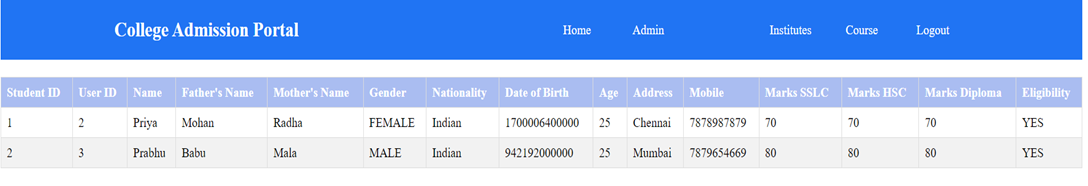
**Edit Menu**

**(**On clicking on the edit button, the columns must be editable except institute name)

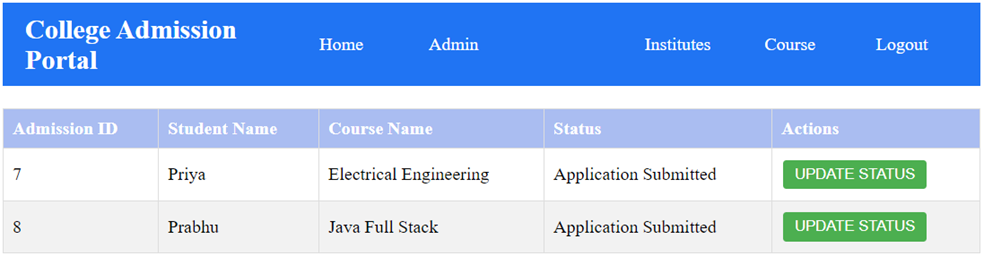


16. **View Students**: Design a component

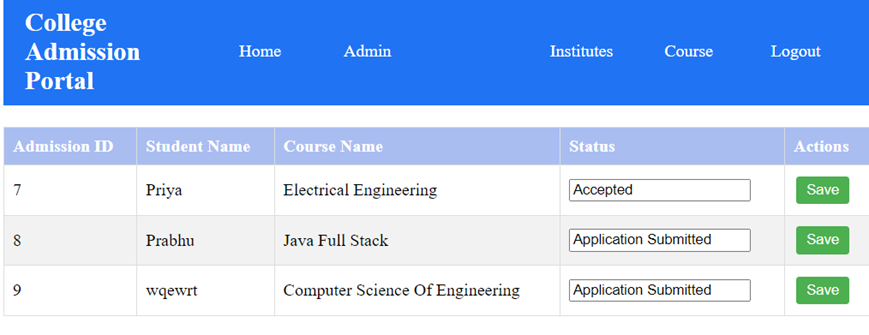
**Students List page**

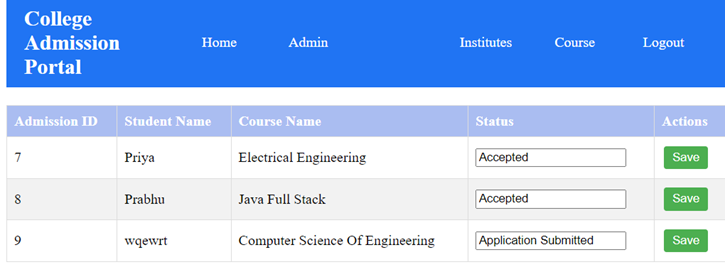


17.   **View Admission page:** Design a page for view admission page

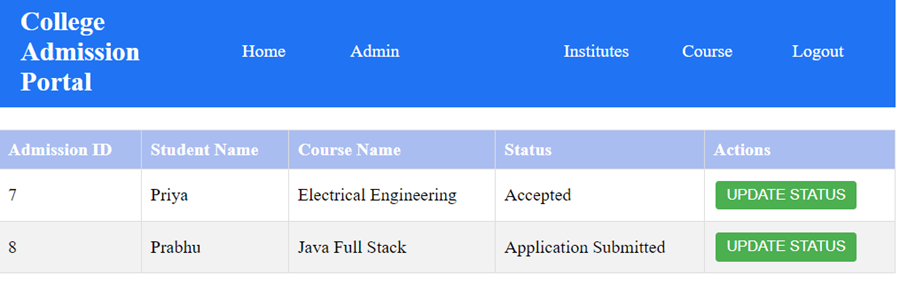
****

On clicking the update status button, the page should make the status field editable and change to action to save button

**  
On** Clicking the save button, the status should be changed



18.   **View-Payment page:** Design a page for payment history



**19.   Error Page** : Design a page for handling error (Name the component as error)

A blue and white screen with text

Description automatically generated**Platform Prerequisites (Do’s and Don’ts):**

**Ø** The react app should run in port 8081.

Ø The springboot app should run in port 8080.

**Other Important Key factors in the application:**

**•** Should use Custom Exceptions mandatory.

• Tables should have proper relationship and keys

• Frontend Application should be menu driven.

• Proper Menu / Navigation for corresponding role

• Client side validations and server side validations are mandatory

• Error should be handled.

• Follow best programmer practice while developing

• Provide proper Naming Conventions

**Key points to remember:**

**1.**The id (for frontend) and attributes(backend) mentioned in the SRS should not be modified at any cost. Failing to do may fail test cases.

2.    Remember to check the screenshots provided with the SRS.

3.    Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.

4.    Adhere strictly to the endpoints mentioned in API endpoints section.

5.    Don't delete any files in a project environment.

**HOW TO RUN THE PROJECT:**

**FRONTEND:**

**Step 1:**

Use “nvm use 14” command to change node version to 14

**Step 1:**

Use "cd reactapp" command to go inside the reactapp folder

Install Node Modules **- "**npm install**"**

**Step 2:**

Write the code inside src folder

Create the necessary components

**Step 3:**

Click the run test case button to run the test cases

**Note :**

* Click PORT 8081 to view the result / output
* If any error persists while running the app , delete the node modules and reinstall them

**BACKEND:**

**Do not modify the application.properties and pom.xml files. If you change there may be build failure and test case will fail. Follow the naming convention as specified in the image shown above.**

**API endpoint:**

8080

**Platform Guidelines:**

To run the command use **Terminal**in the platform.

**Spring Boot:**

Navigate to the springapp directory => **cd springapp**

To start/run the application '**mvn spring-boot:run**'

Click on the Run Test Case button to pass all the test cases