## YOGA Academy Admission Portal

**Overview:**

This application is built to help customers to get YOGA Academy Admission online.

## Users of the System:

1. Admin
2. User

## Functional Requirements:

* Build a portal that enables customers can get YOGA Academy Admission online.
* The customers can add/edit/view/delete admission.
* The admin can add/edit/delete/view courses.
* The admin can add/edit/delete/view institutes.
* The admin can add/edit/delete/view students.
* Customer can provide reviews.

While the above ones are the basic functional features expected, the below ones can benice to have add-on features:

* Have appropriate filters for search.
* Email integration for intimate customers.
* Multi-factor authentication for the sign-in process
* Payment Gateway (if required)

## Output/ Post Condition:

* Records Persisted in Success & Failure Collections
* Standalone application / Deployed in an app Container

## Non-Functional Requirements:

|  |  |
| --- | --- |
| **Security** | * App Platform – Username/Password-Based Credentials * Sensitive data has to be categorized and stored in a securemanner * Secure connection for transmission of any data |
| **Performance** | * Peak Load Performance (during Festival days, National holidaysetc.) |

|  |  |
| --- | --- |
|  | * Admin application < 2 Sec * Non-Peak Load Performance * Appointment Application< 2 Sec * Admin Application < 2 Sec |
| **Availability** | * 99.99 % Availability |
| **Standard Features** | * Scalability * Maintainability * Usability * Availability * Failover |
| **Logging &**  **Auditing** | * The system should support logging(app/web/DB) & auditing at   all levels |
| **Monitoring** | * Should be able to monitor via as-is enterprise monitoring tools |
| **Cloud** | * The Solution should be made Cloud-ready and should have a   minimum impact when moving away to Cloud infrastructure |
| **Browser Compatible** | * All latest browsers |

**Technology Stack**

|  |  |
| --- | --- |
| Front End | Angular 10  Material Design Bootstrap / Bulma |
| Server Side | Spring Boot |
| Database | MySQL or Oracle or MSSQL |

## Platform Prerequisites (Do’s and Don’ts):

1. The react app should run in port 8081.
2. Spring boot app should run in port 8080.

## 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. Strictly adhere to id mapping and attribute mapping. Failing to do may fail test cases.
3. Strictly adhere to the proper project scaffolding (Folder structure), coding conventions, method definitions and return types.
4. Adhere strictly to the endpoints given below.

## This is a basic SRS document, so understand them well and please feel free to explore and come with new ideas.

**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 can Activate interface. For example, if the user enters as http://localhost:8080/signup or http://localhost:8080/home the page should not navigate to the corresponding page instead it should redirect to the login page.
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. Use admin/admin as the username and password to navigate to the admin dashboard.

## Validations:

1. Basic email validation should be performed.
2. Basic mobile number validation should be performed.

## Project Tasks: API Endpoints: Admin Side:

|  |  |  |  |
| --- | --- | --- | --- |
| Action | URL | Method | Response |
| Admin Login | /admin/login | POST-Sends email ID and password | Return True/False |
| Admin SignUp | /admin/signup | POST-Sends Admin Model data | Admin added |
| Add Courses | /admin/addCourse | POST – Sends Course data | Courses added |
| View Courses | /admin/viewCourse | GET – Fetches course data | Retrieve all the courses |
| Edit Courses | /admin/editCourse/{cours  eId} | PUT – Send course Id | Course edited |
| Delete Courses | /admin/deleteCourse | DELETE – Send course Id | Course deleted |
| Add Institutes | /admin/addInstitute | POST – Sends Institute data | Institute added |
| View Institutes | /admin/viewInstitutes | GET – Fetches course data | Retrieve all the institute |
| Edit Institutes | /admin/editInstitute/{inst ituteId} | PUT – Sends institute Id | Institute edited |

|  |  |  |  |
| --- | --- | --- | --- |
| Delete Institutes | /admin/ deleteInstitutes | DELETE – Sends Institute Id | Institute deleted |
| Add Student | /admin/addStudent | POST – Sends student data | Student added |
| View Student | /admin/viewStudent | GET – Fetches student details | Retrieve all the student details |
| Edit Student | /admin/editStudent/{stud entId} | PUT – sends student id | Student details edited |
| Delete Student | /admin/deleteStudent/{st udentId} | DELETE – sends student id | Student details deleted |

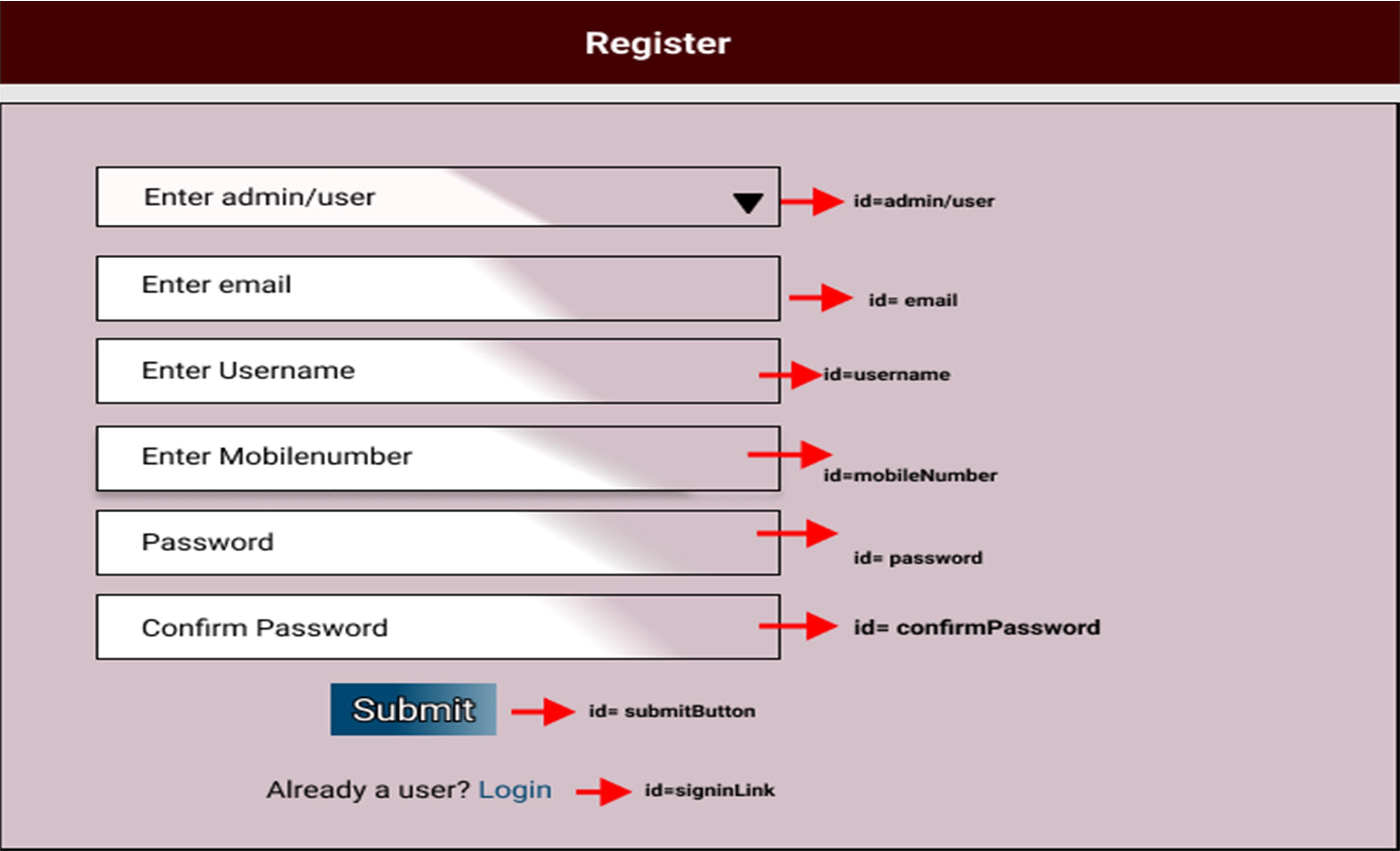
**User Side:**

|  |  |  |  |
| --- | --- | --- | --- |
| Action | URL | Method | Response |
| User Login | /user/login | POST-Sends email ID and  password | Return True/False |
| Admin SignUp | /user/signup | POST-Sends User Model  data | User added |
| Add Admission | /user/addAdmission | POST –  Sends admission data | Course enrolled |
| View Admission | /user/viewAdmission | GET –  Fetches admission  data | Retrieve the admission  details |
| Edit Admission | /user/editAdmission/{enrolI d} | PUT – Sends admissionId | Admission details edited |
| Delete Admission | /user/deleteAdmission/{enr ollId} | DELETE –  Sends admissionId | Admission details deleted |
| View Status | /user/viewStatus | GET – Fetches  Admission status | Admission Application Status |

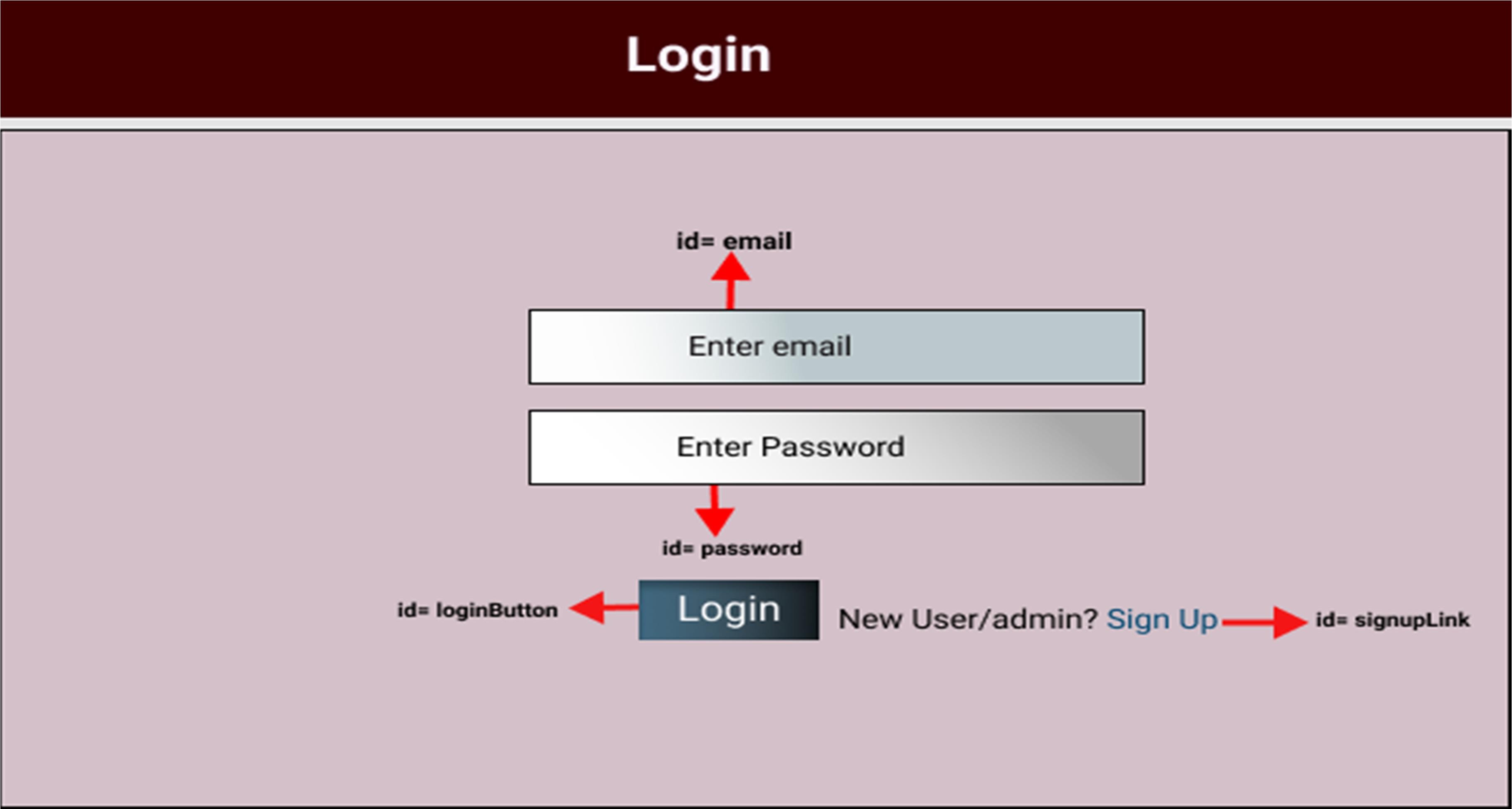
# Frontend:

## Customer:

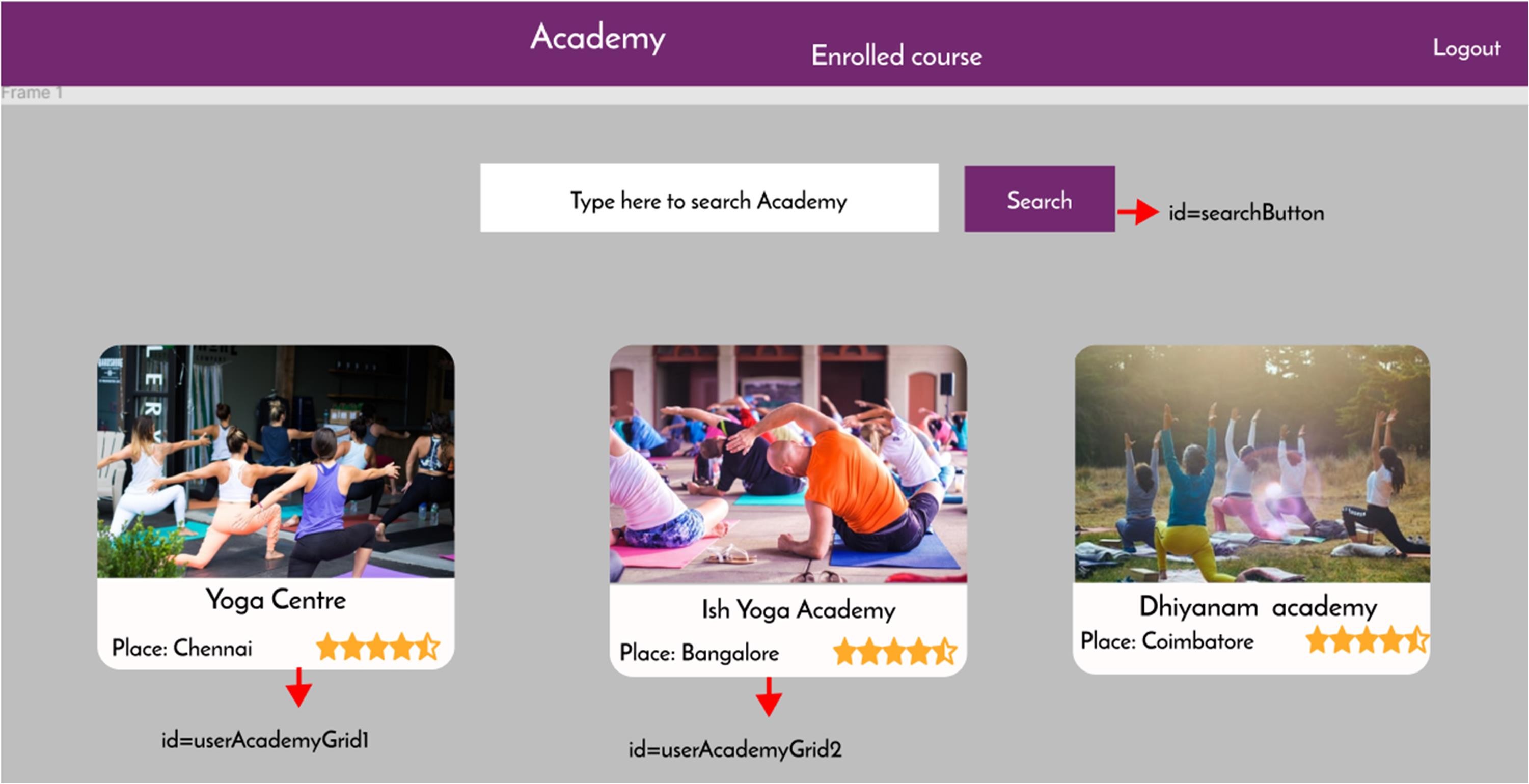
1. Auth: Design an auth component where the customer can authenticate login and signup credentials
2. Signup: Design a signup page component inside the auth where the new customer has options to sign up by providing their basic details.
   1. Ids: Refer to the screenshot below for the id details.
   2. Output screenshot:

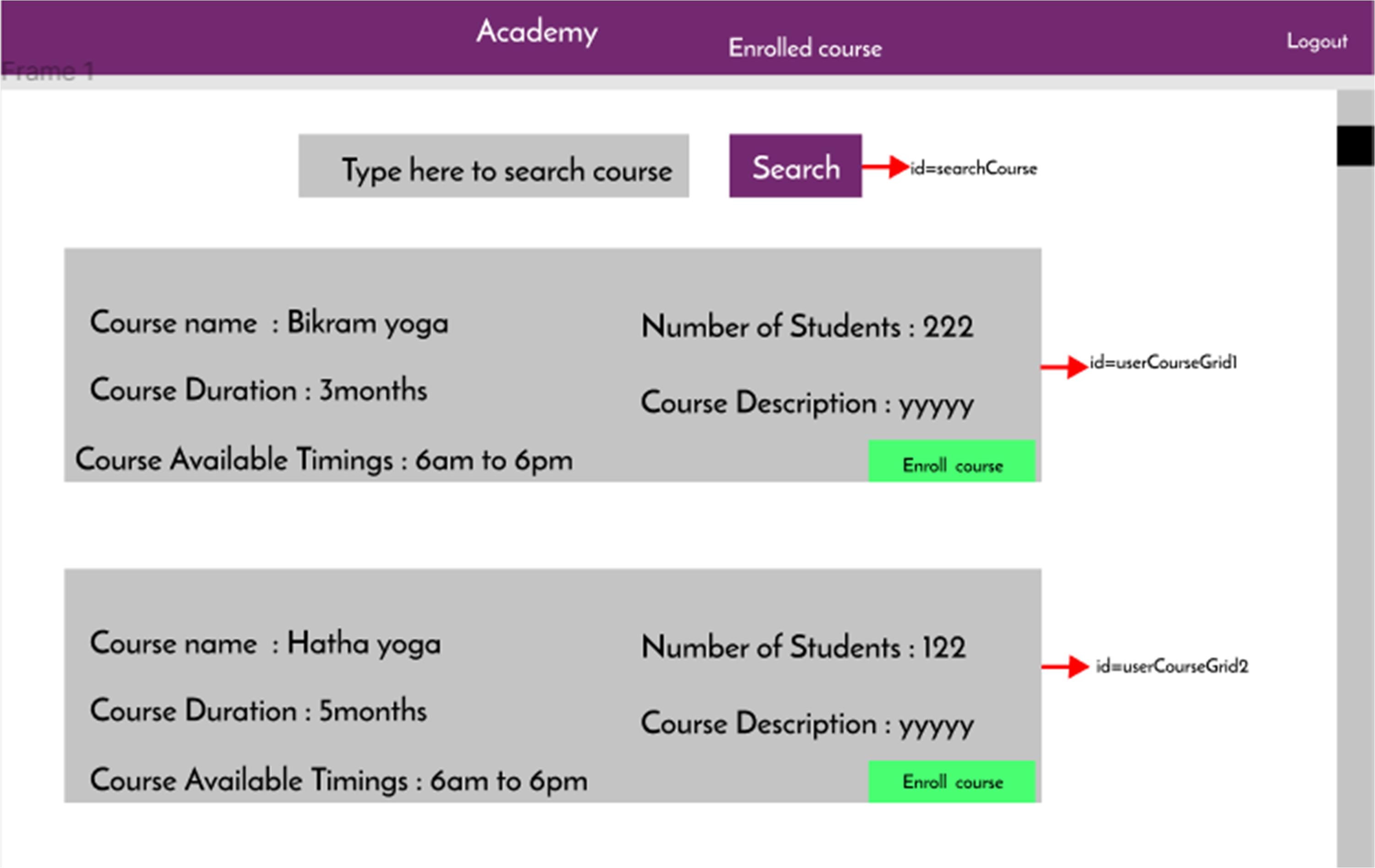


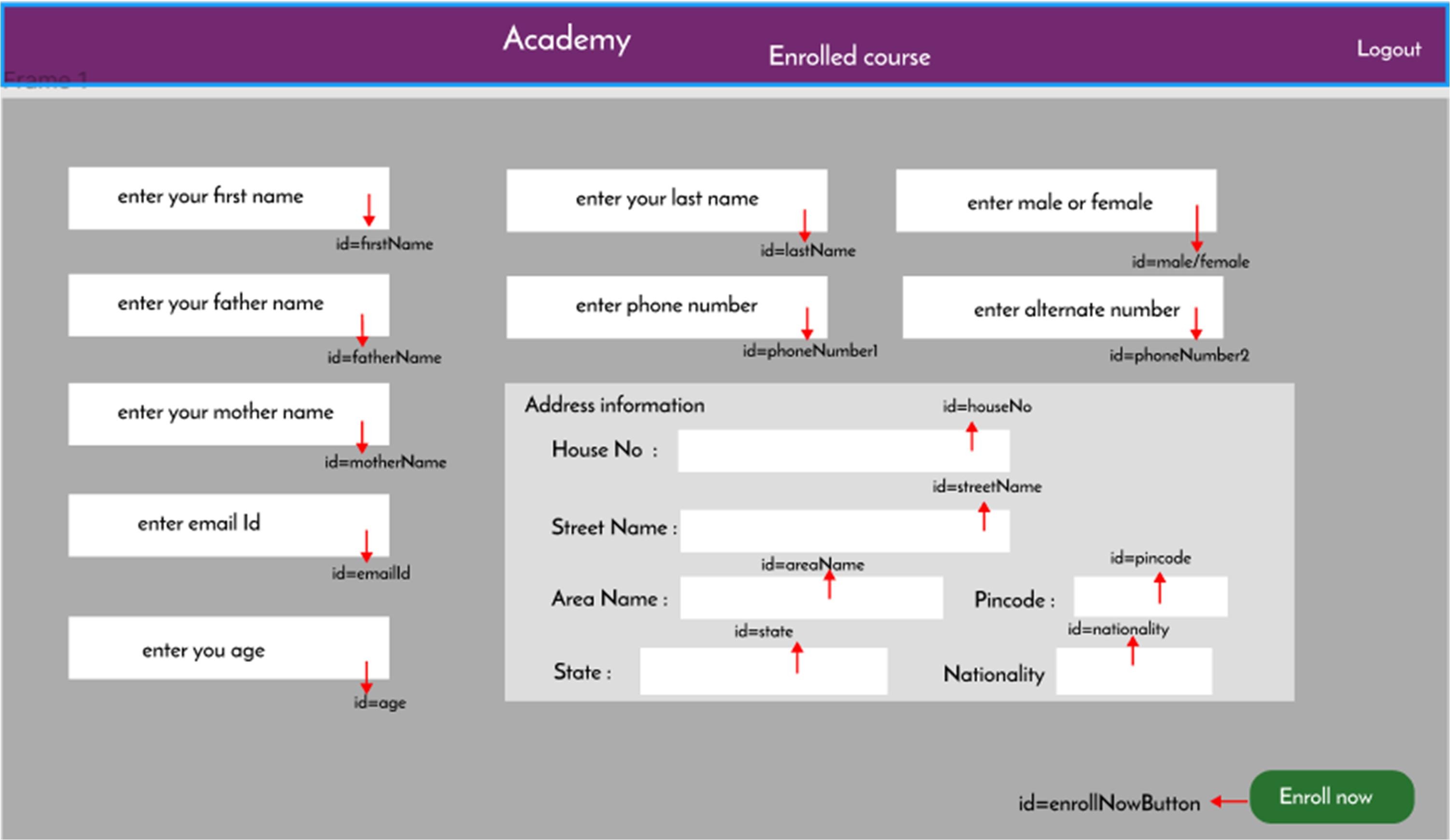
1. Login: Design a login page inside the auth where the existing customer can log in using the registered email id and password.
   1. Ids: Refer to the screenshot below for the id details.
   2. Output Screenshot:



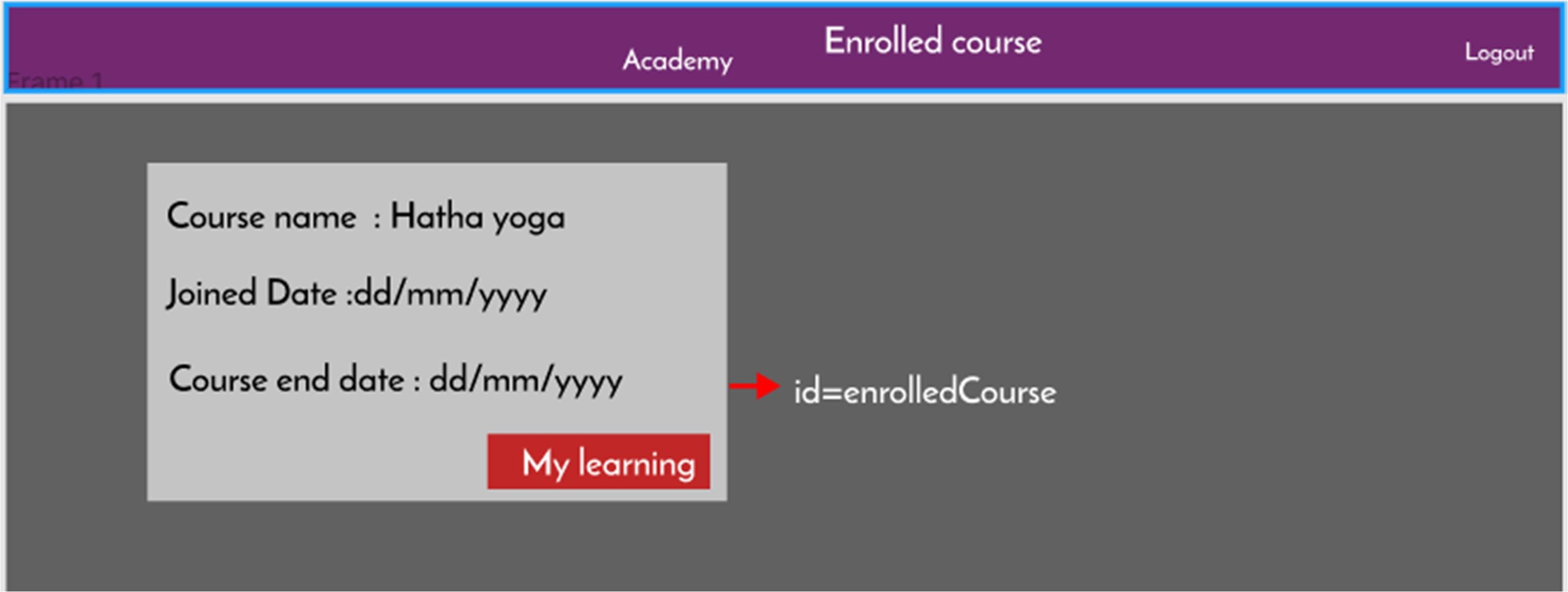
1. View Academy: Design a component
   1. Ids: Refer to the screenshot below for the id details.
   2. Output Screenshot:







1. Enrolled Course: Design a component
   1. Ids: Refer to the screenshot below for the id details.
   2. Output Screenshot:

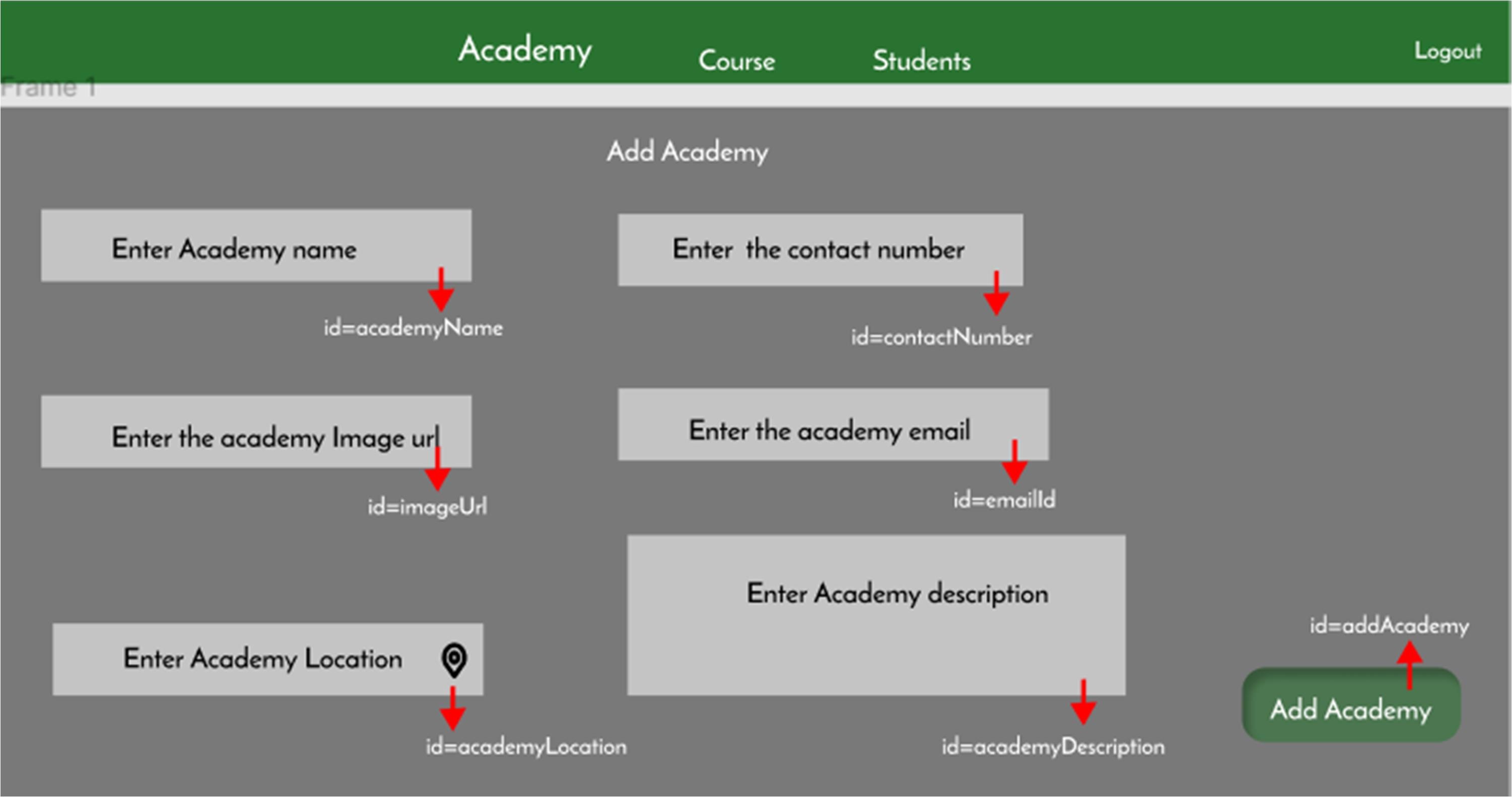


## Admin:

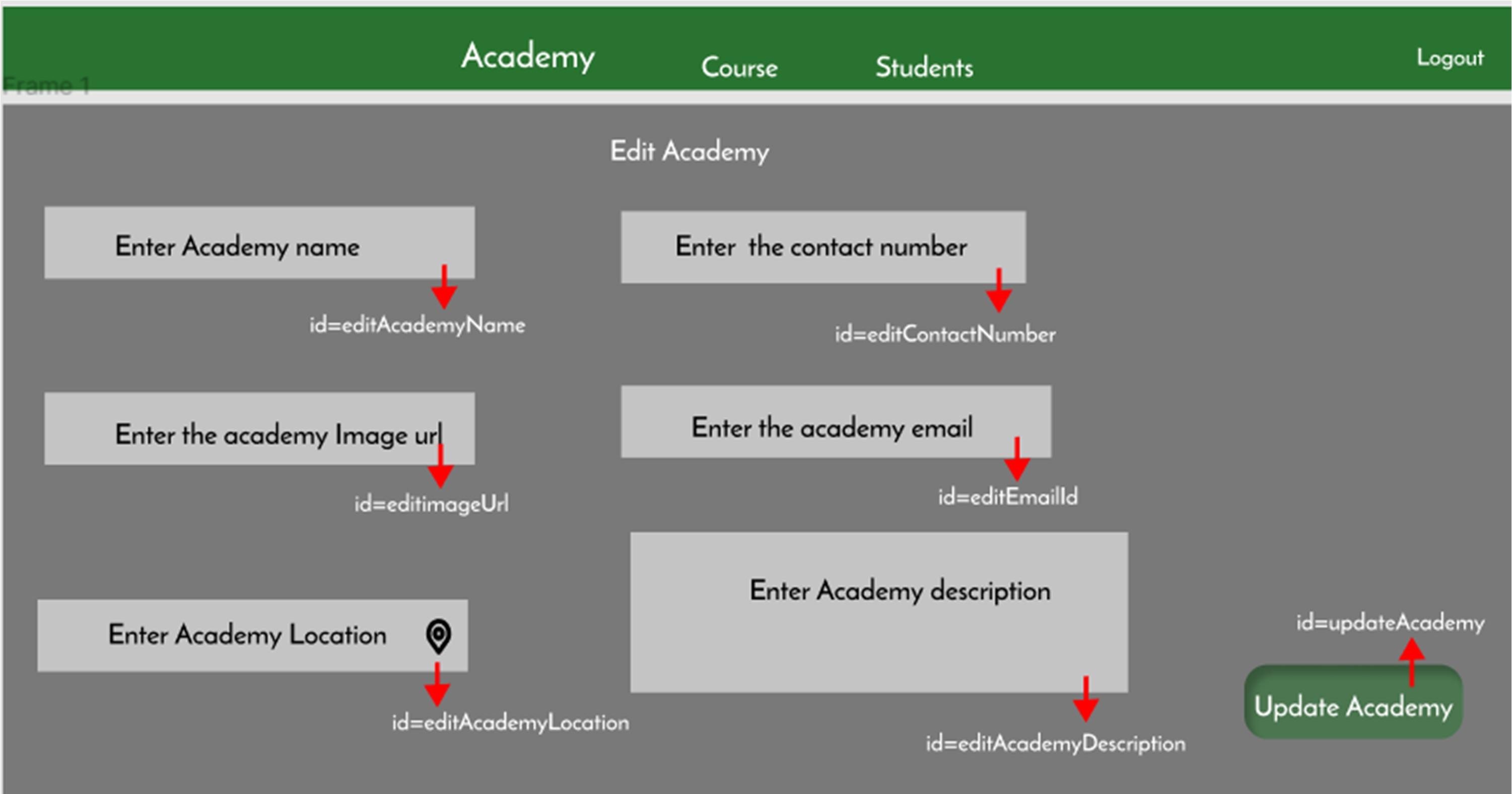
1. Admin Academy: Design a component .Admin can add the new academy details.
   1. Ids: Refer to the screenshot below for the id details.
   2. Output Screenshot:



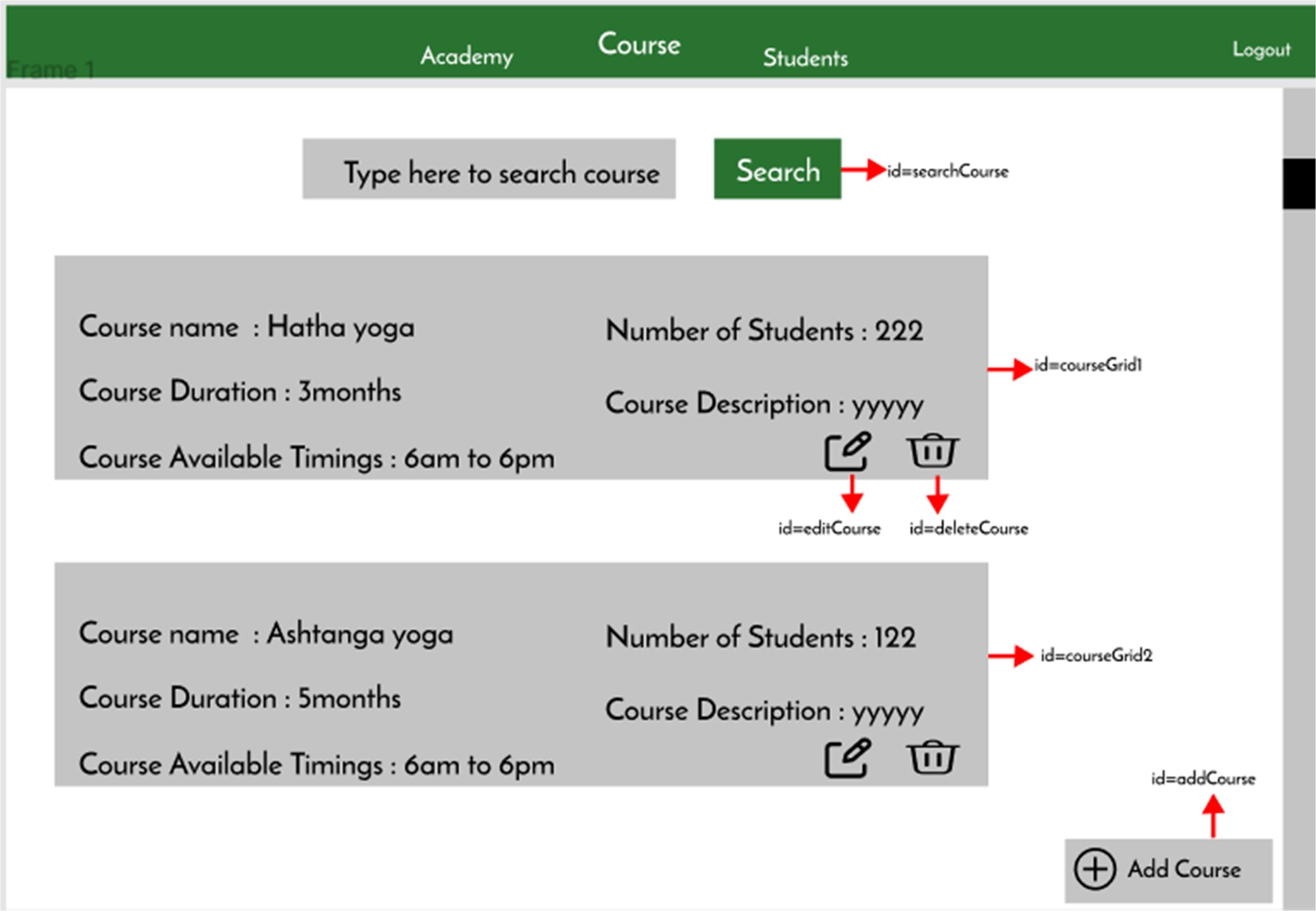
Add:



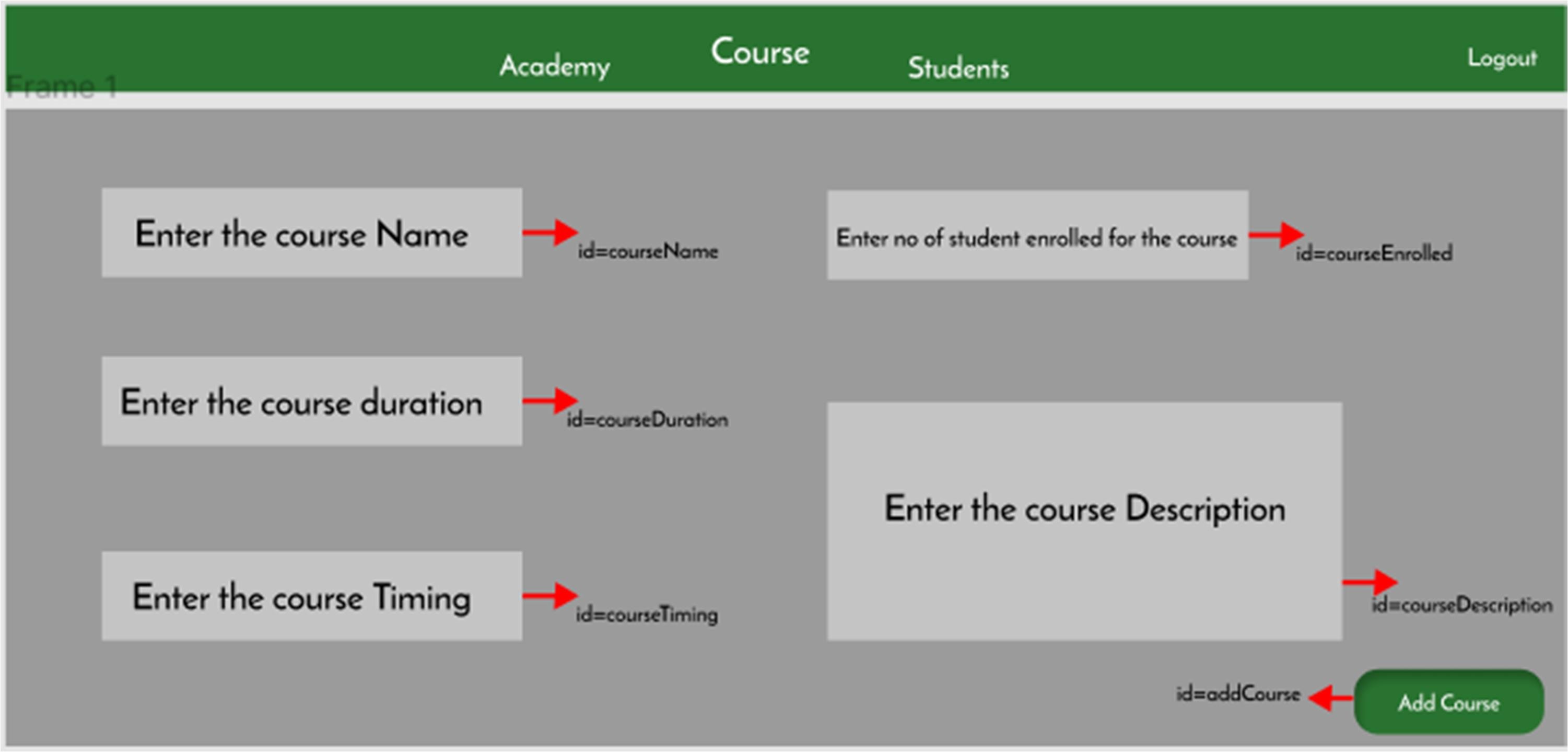
Edit:



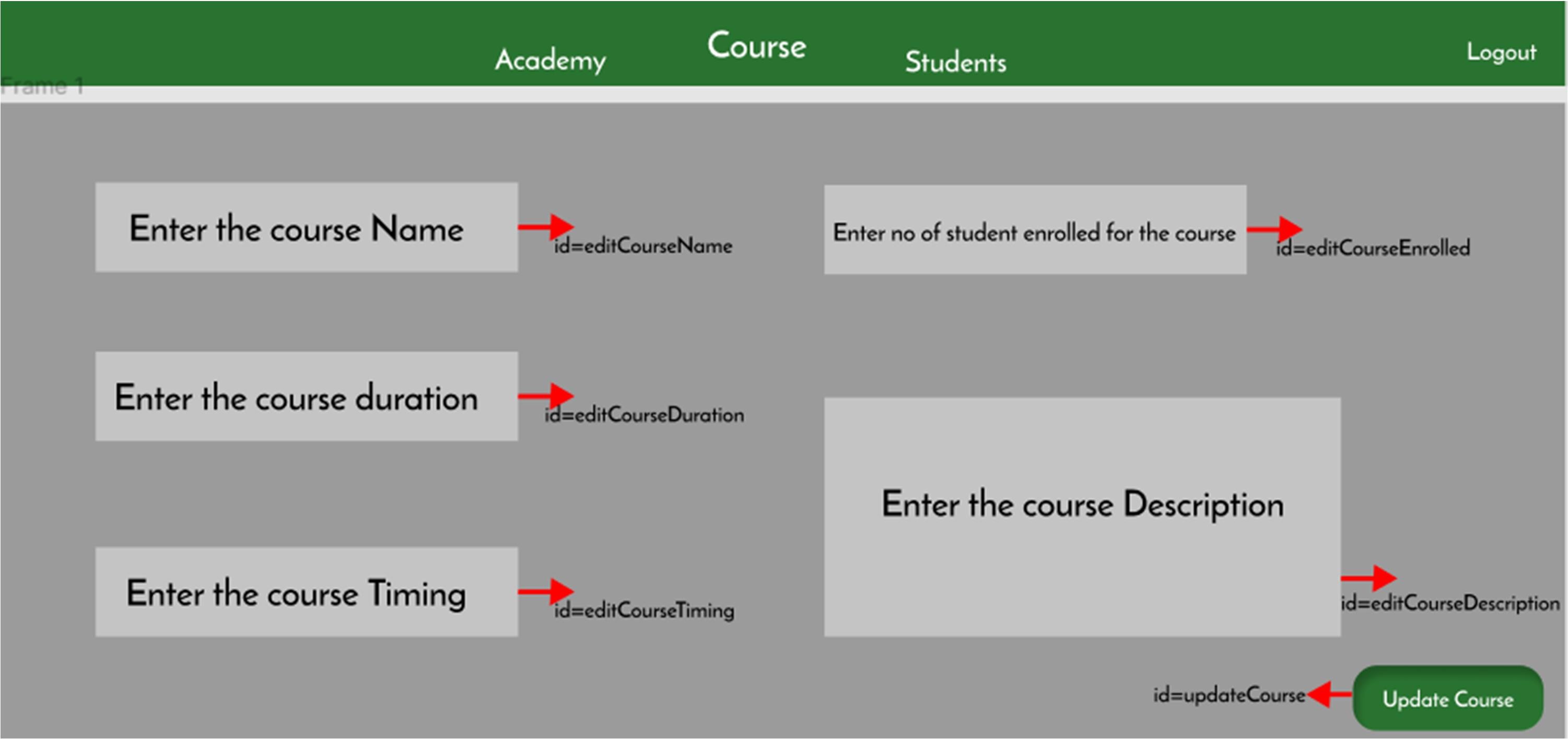
1. Admin Course: Design a component
   1. Ids: Refer to the screenshot below for the id details.
   2. Output Screenshot:



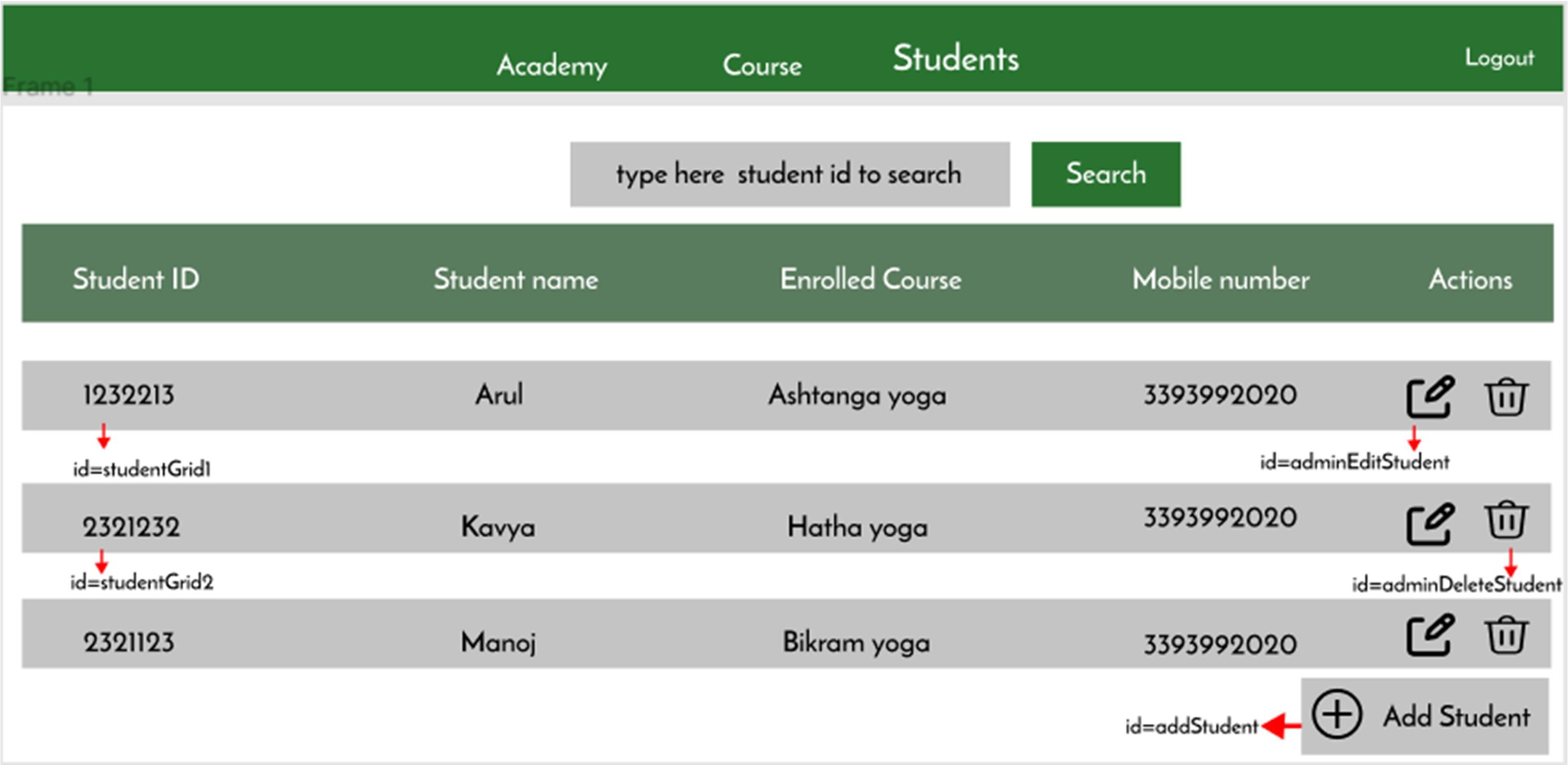
Add:



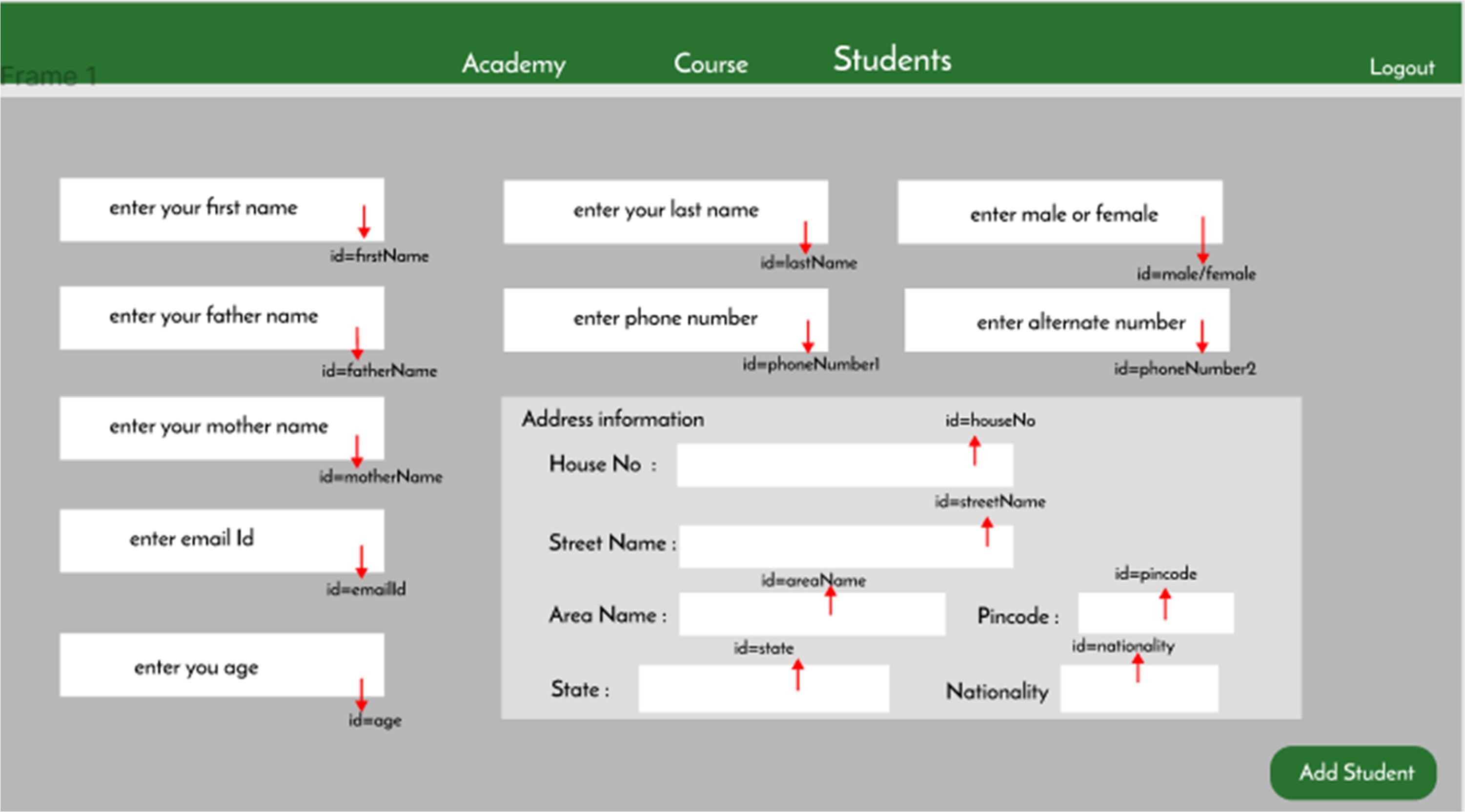
Edit:



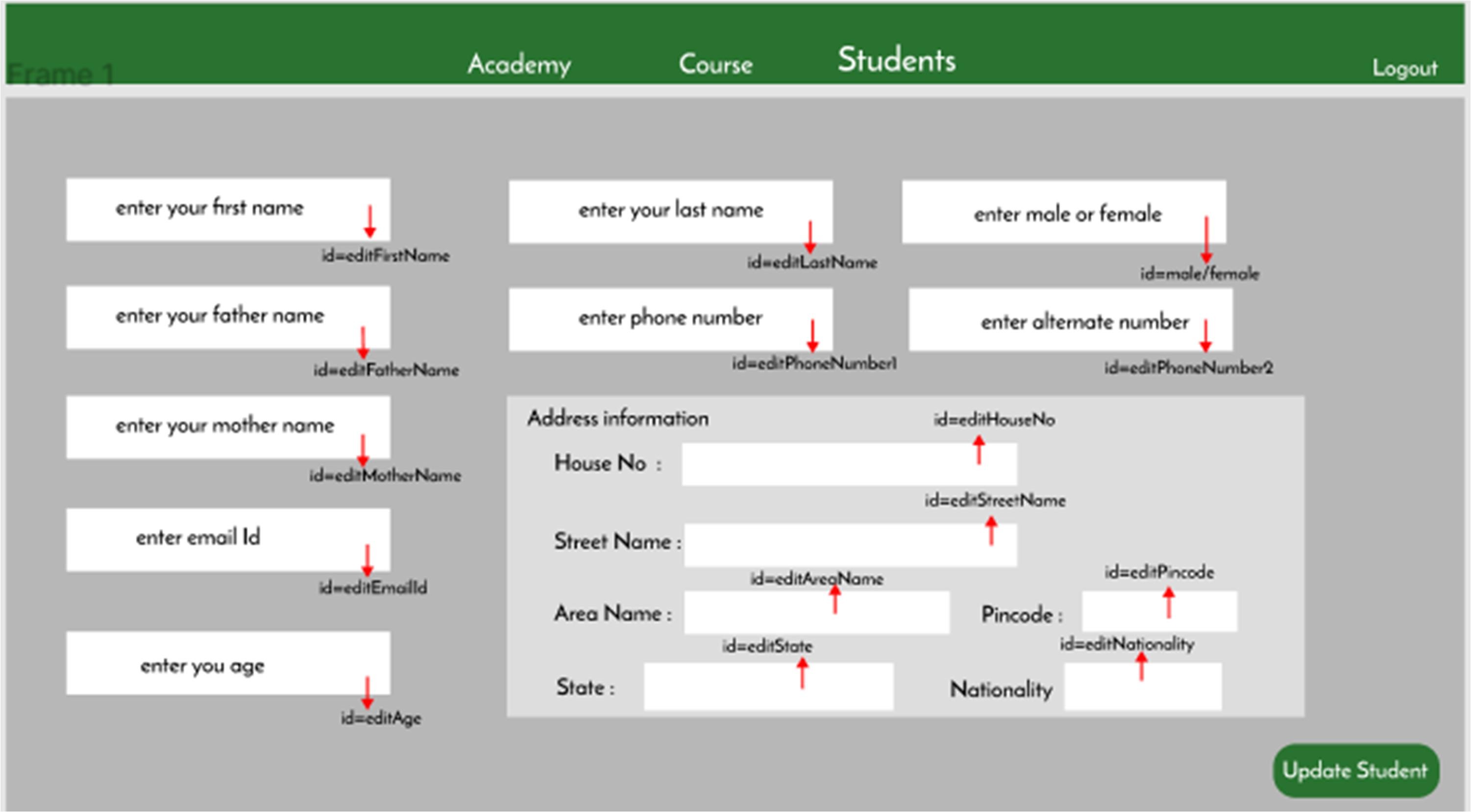
1. Admin Students: Design a component .In this component admin can edit and delete the students.
   1. Ids: Refer to the screenshot below for the id details.
   2. Output Screenshot:



Add:



Edit:



v

# Backend:

## Class and Method description:

**Model Layer:**

1. **UserModel**: This class stores the user type (admin or the customer) and all user information.
   1. Attributes:
      1. email: String
      2. password: String
      3. username: String
      4. mobileNumber: String
      5. userRole: String
2. **LoginModel**: This class contains the email and password of the user.
   1. Attributes:
      1. email: String
      2. password: String
3. **AdminModel**: This class stores the details of the admin.
   1. Attributes:
      1. email:String
      2. password:String
      3. mobileNumber:String
      4. userRole:String
4. CourseModel: This class stores the details of the course
   1. Attributes:
      1. courseId: int
      2. courseName: String
      3. courseDescription: String
      4. courseDuration: int
5. InstituteModel: This class stores the details of the Institute or College
   1. Attributes:
      1. instituteId: int
      2. instituteName: String
      3. instituteDescription: String
      4. instituteAddress: String
      5. mobile: String
      6. email: String
6. Student Model: This class stores the details of the students.
   1. Attributes:
      1. studentId: int
      2. studentName: String
      3. studentDOB: Date
      4. address: string
      5. mobile: String
      6. Age:int

## Controller Layer:

1. **AuthController**: This class control the user /admin signup and signin
   1. Methods:
      1. isUserPresent(LoginModel data): This method helps to check whether the user present or not and check the email and password are correct and return the boolean value.
      2. isAdminPresent(LoginModel data): This method helps to check whether the admin present or not and check the email and password are correct and return the boolean value.
      3. saveUser(UserModel user): This method helps to save the user data in the database.
      4. saveAdmin(UserModel user): This method helps to save the admin data in the database.
2. **UserController**: This class helps to add/edit/view/delete admission process.
   1. Methods:
      1. addAdmission(StudentModel student, int courseId, int instituteId):This method adds new admission.
      2. editAdmission(int admissionId): This method helps to edit admission details
      3. viewAdmission(int admissionId): This method helps to view the admission details
      4. deleteAdmission(int admissionId): This method helps to delete the admission
      5. ViewStatus(int admissionId): This method helps to view the status of the

admission.

1. AdminController: This class helps to add/edit/view/delete various details necessary with admission process.
   1. Methods:
      1. .addStudent(StudentModel student): This method helps to add student.
      2. viewStudent(int studentId): This method helps to view student.
      3. editStudent(int studentId): This method helps to edit student.
      4. deleteStudent(int studentId) This method helps to delete student.
      5. addCourse(CourseModel course): This method helps to add course.
      6. editCourse(int courseId): This method helps to edit course.
      7. deleteCourse(int courseId): This method helps to delete course.
      8. viewCourse(int courseId): This method helps to view course.
      9. addInstitute(int instituteId): This method helps to add institute.
      10. editInstitute(int instituteId): This method helps to edit institute.
      11. deleteInstitute(int instituteId): This method helps to delete the institute
      12. ViewInstitute(int instituteId): This method helps to view the institute details

How to run the Project

**Back End**

**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**'

**To Connect Database Open Terminal**

**Cmd:mysql -u root –protocol=tcp -p**

**Password: examly  
  
Front End**

**Step 1:**

Open the terminal

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