**CSCE 5350 - Term Project (Option 1)**

**Team members:**

* **Anjana Priya Bachina**
* **Nikitha Dadi**

We are assigned project 19 to design and develop a database for a college.

**Introduction:**

The College Database Management System is one comprehensive application that is intended to handle important college information such as departments, instructors, courses, and students. It uses a state-of-the-art technology solution that is highly scalable and efficient, as well as easy to use. The architecture is divided into three core layers:

* Database Layer: The open-source relational database system PostgreSQL guarantees reliability and offers robust query functionality for managing relationships between departments, instructors, courses, students, and their enrollments.
* Backend Layer: A Node.js server using Express.js controls every CRUD functionality through RESTful APIs that connect the database to the front end.
* Frontend Layer: The web application was built in React.js and offers administrators a user-friendly panel inclusive of different data visualizations and easy navigation.

**Key Features:**

* User Interface: Simplified forms and tables that use React.js to work with the data as well as to visualize it.
* API Endpoints: Traditional RESTful APIs developed using Node.js and Express.js to perform data manipulation safely and efficiently.
* PostgreSQL Database: Standardized schema architecture for maintaining the relational consistency of the DBMS along with the enhanced execution of complex query language.
* Responsive Design: The design should be responsive to different devices so that it can be accessed and used easily across them.
* Extensibility: Clean code so that future changes and added functionality can be easily done.

**System Design Documentation**

This section provides general information about the structural and functional organization of the University Management System, about the architectural components and design choices.

**1. System Architecture**

The University Management System is designed using a three-tier architecture consisting of the following layers:

**Frontend (Presentation Layer): React.js**

* Enables the management of students, instructors, courses, and departments with a friendly user interface.
* Can handle the routing of components with the help of React Router.
* Interacts with the backend using the API only.

**Backend (Application Layer): Node.js**

* It works at the middle level as a connector between the front end and the database.
* Uses CRUD convention that stands for Create, Read, Update, and Delete, for RESTful APIs.
* Deals with business tasks and validates results.

**Database (Data Layer): PostgreSQL**

* Maintains structured information associated with students, instructors, courses, and departments.
* Establish relationships between entities with the aid of the primary and foreign keys.

**2. Data Flow**

**Frontend:**

* The UI is used by the user to carry out activities such as enrolling in a student or viewing information concerning the courses available.
* These actions result in API calls to the backend.

**Backend:**

* It takes the requests from the other components, processes them, and communicates with the database to read or write some data.
* Data to be processed is returned to the frontend via JSON.

**Database:**

* Performs SQL queries from the backend to retrieve or modify data.
* Keeps the relation between two tables intact.

**ER Diagram:**

1. **Entities and Attributes**:

* **Courses**
  + Attributes: course\_id (Primary Key), course\_name, department\_id (Foreign Key), instructor\_id (Foreign Key).
* **Students**
  + Attributes: student\_id (Primary Key), student\_name, email, phone\_number.
* **Enrollments**
  + Attributes: student\_id (Foreign Key), course\_id (Foreign Key).
* **Instructors**
  + Attributes: instructor\_id (Primary Key), instructor\_name, department\_id (Foreign Key).
* **Departments**
  + Attributes: department\_id (Primary Key), department\_name, head\_id (Foreign Key to instructor\_id).

1. **Relationships**:

* **Courses and Departments**
  + Each course belongs to one department (department\_id as a foreign key in the courses table).
* **Courses and Instructors**
  + Each course is taught by one instructor (instructor\_id as a foreign key in the courses table).
* **Students and Enrollments**
  + The enrollments table acts as a junction table, linking students and courses with many-to-many relationships.
* **Instructors and Departments**
  + Each instructor works in one department (department\_id as a foreign key in the instructor's table).
* **Departments and Head of Department**
  + Each department has a head, referenced by head\_id, which is a foreign key to the instructor's table.

1. **Indexes**:

* Primary keys (e.g., student\_id, course\_id) ensure unique entity.
* Foreign keys ensure referential integrity across related tables.

A diagram of a course

Description automatically generated with medium confidence

**Database Schema:**

* **Table: students**

CREATE TABLE Students (

student\_id INT PRIMARY KEY AUTO\_INCREMENT,

student\_name VARCHAR(100) NOT NULL,

email VARCHAR(255) NOT NULL,

phone\_number VARCHAR(15) NOT NULL

);

* **Table: Departments**

CREATE TABLE Departments (

department\_id INT PRIMARY KEY AUTO\_INCREMENT,

department\_name VARCHAR(100) NOT NULL,

head\_id INT,

FOREIGN KEY (head\_id) REFERENCES Instructors(instructor\_id)

);

* **Table: Instructors**

CREATE TABLE Instructors (

instructor\_id INT PRIMARY KEY AUTO\_INCREMENT,

instructor\_name VARCHAR(100) NOT NULL,

department\_id INT NOT NULL

);

ALTER TABLE Instructors

ADD CONSTRAINT fk\_department

FOREIGN KEY (department\_id) REFERENCES Departments(department\_id);

* **Table: Courses**

CREATE TABLE Courses (

course\_id INT PRIMARY KEY AUTO\_INCREMENT,

course\_name VARCHAR(100) NOT NULL,

department\_id INT NOT NULL,

instructor\_id INT NOT NULL,

FOREIGN KEY (department\_id) REFERENCES Departments(department\_id),

FOREIGN KEY (instructor\_id) REFERENCES Instructors(instructor\_id)

);

* **Table: Enrollments**

CREATE TABLE Enrollments (

student\_id INT NOT NULL,

course\_id INT NOT NULL,

PRIMARY KEY (student\_id, course\_id),

FOREIGN KEY (student\_id) REFERENCES Students(student\_id),

FOREIGN KEY (course\_id) REFERENCES Courses(course\_id)

);

**Backend API Design:**

The back-end part of the University Management System is developed with Node.js, which is divided into modular points for controlling basics of the entities. Every API adheres to the RESTful principles to maintain the readability and scalability of the APIs.

**Students API**

* + GET /api/students: Get the list of all students.
  + POST /api/students: Enroll a new student into the system.
  + PUT /api/students/:id: Alter information of a certain student based on his identification number.
  + DELETE /api/students/: Delete a student from the system.

**Instructors API**

* + GET /api/instructors: Retrieve all instructors.
  + POST /api/instructors: This form is used to add a new instructor to the system.
  + PUT /api/instructors/:id: Change with the instructor information (future improvement).

**Courses API**

* + GET /api/courses: Search for all the courses with department information and instructor information.
  + POST /api/courses: New course by a particular department.

**Departments API**

* + GET /api/departments: Retrieve all departments.
  + POST /api/departments: Add a new department.
  + PUT /api/departments/:id: modify the department’s information.

**Enrollments API**

* + POST /api/enrollments: Register a student in a course with a student over the course.
  + DELETE /api/enrollments/:userid: Remove a student from a course by the enrollment ID of the course.

**Frontend Design :**

Based on the provided information, the frontend for the College Database Management System should be designed with the following features and components:

**Core Features**

**User-Friendly Interface:**

* Read only simplified forms for managing students, instructors, courses, departments and enrollments.
* Accompanied by the organizational layout for easy and quick movement around different areas.

**Responsive Design:** A responsive approach, with pre-designed CSS frameworks.

**Pages and Components**

**Dashboard:** Description of college statistics (number of students and instructors, courses offered, etc.). Also have search functionalities for students list, courses list, and departments list.

**Student Management:**

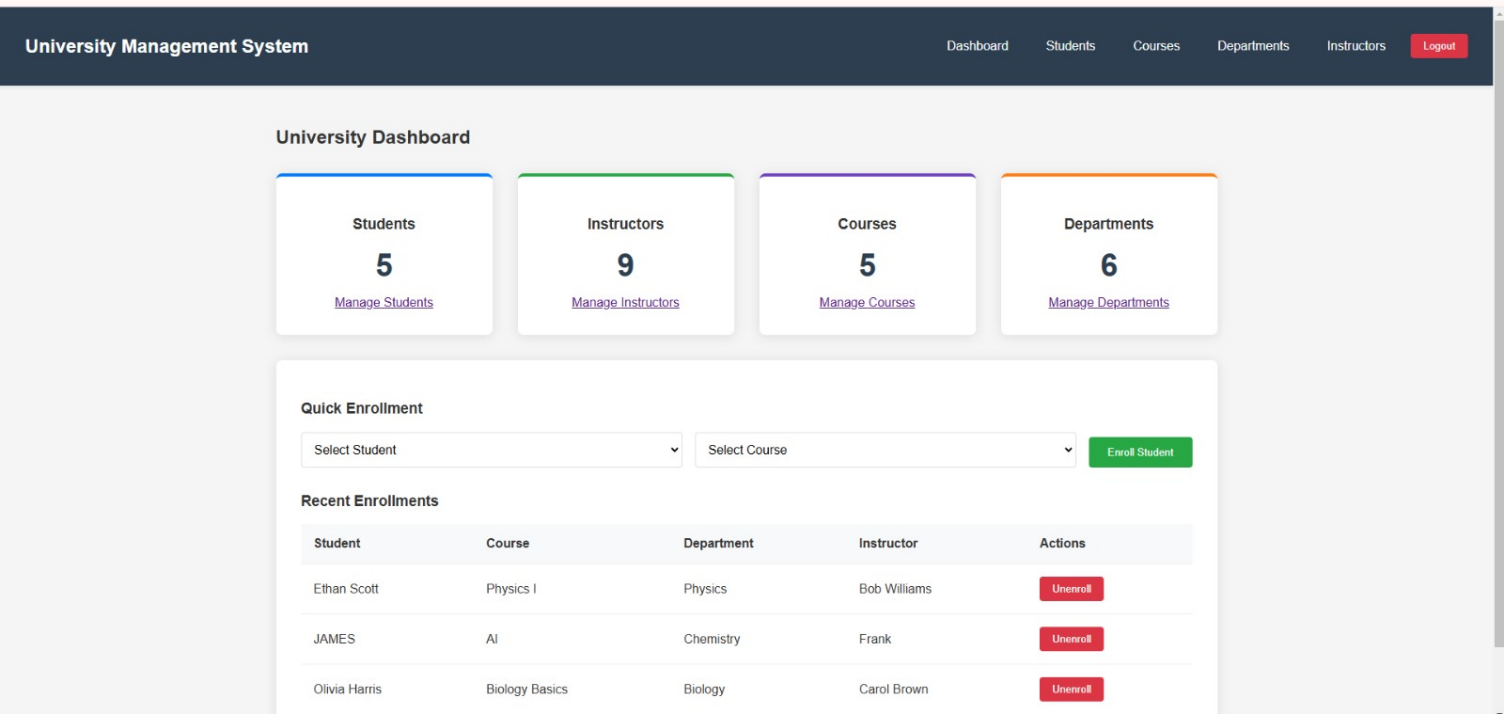
* List View: The numerical table shows the student details inclusive of the view, edit and delete options buttons.
* Add/Edit Form: This is a form for creating or updating students’ profiles with validation.

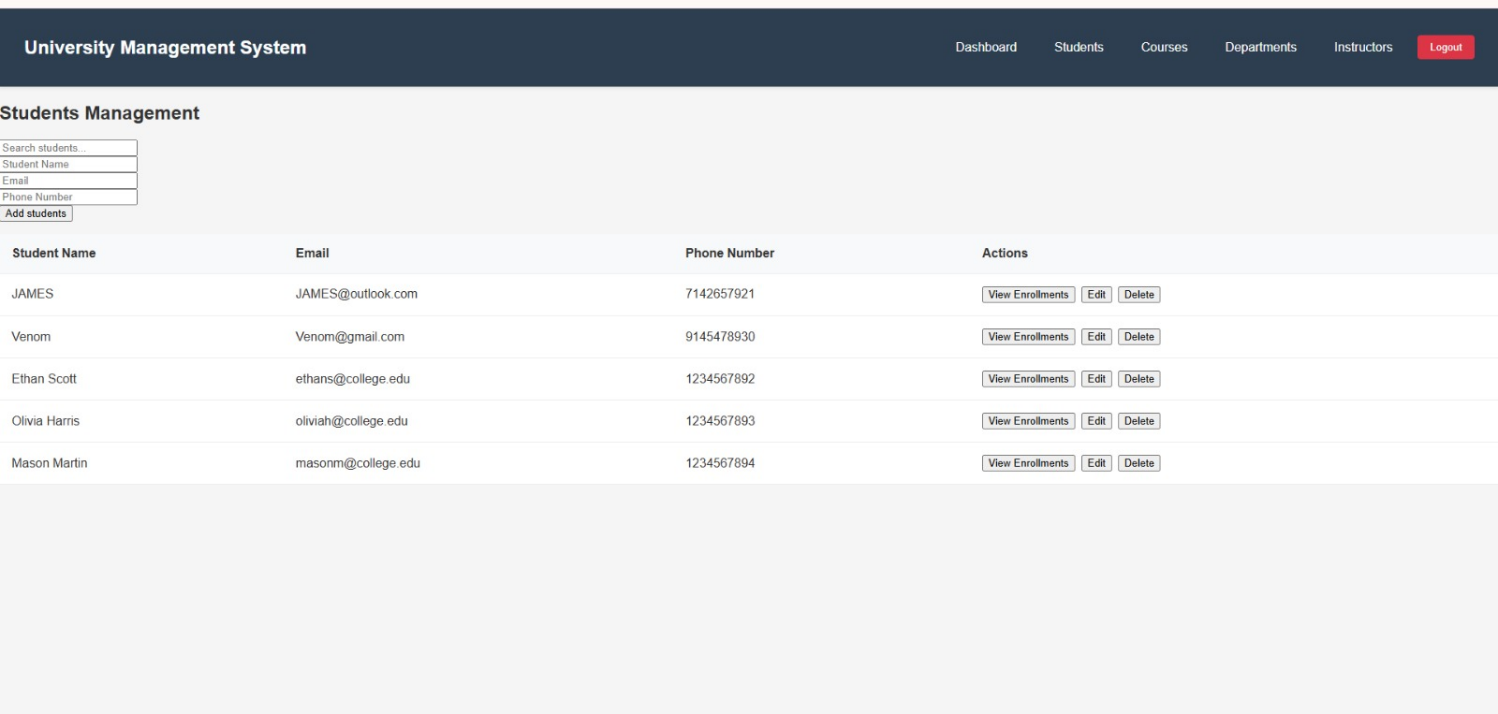
**Instructor Management:** As in the case of in Student Management program that contains many of the similar features as Instructor Management but is structured for the instructors’ data.

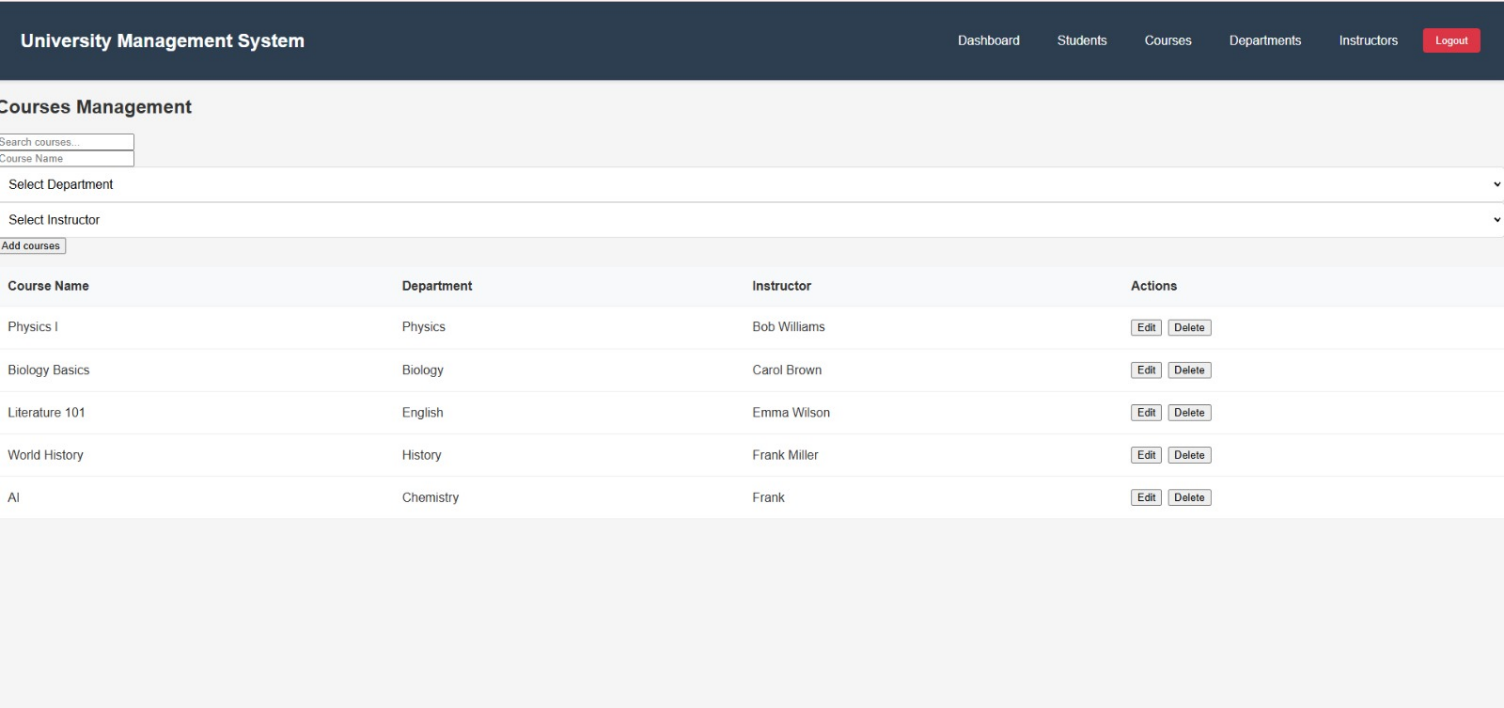
**Courses Management:** Create and List views for the management of courses and demonstrate related departments and teachers.

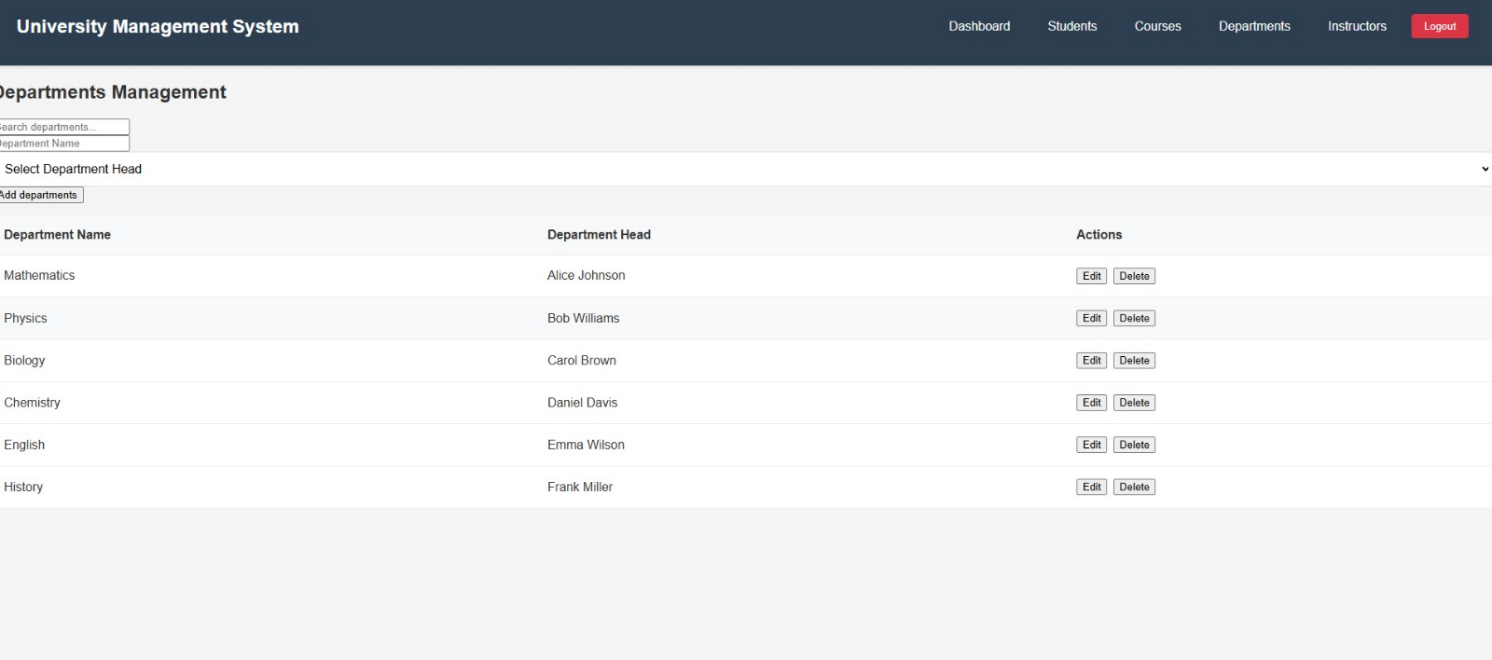
**Department Management:** Department list with head designation and the course affiliated with the department.

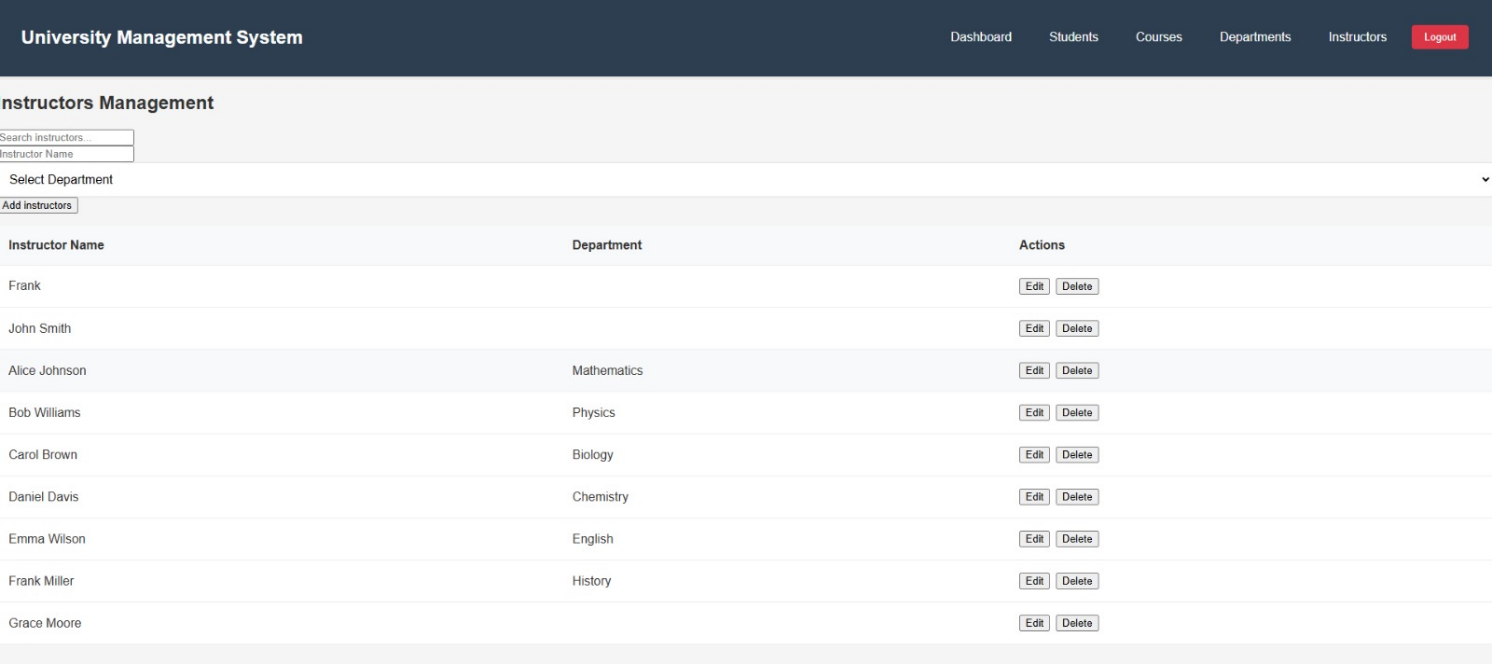
**Enrollments:** Show student-course enrollments with the possibility to add or remove enrollments at your discretion.

****

****

****

****

****