**CMPS 350 Project Phase 2 – Report**

**Education Platform**

**(10% of the course grade)**

**The report must be submitted in Word format only**

|  |  |
| --- | --- |
| **Group Members** | Khalid Mahmoud (202207845)  Islam Al-Absi (202209546)  **Emails:** km2207845@qu.edu.qa; ia2209546@qu.edu.qa; |
| **GitHub link** | https://github.com/ahmedlaqtaf/webdevproject.git |

**Grades :**

**The student fills only the “Implementation Percentage”: Please specify either: *Working (completed x%)*, *Not Working (completed x%)* or *Not done*.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **%** | **Functionality**\* | **Quality of the implementation** | **Grade** |
| Design and implement the Data Model. | 10 |  |  |  |
| Init DB: populate the database with the data from the json files in seed.js | 5 |  |  |  |
| Server actions, APIs and Repository Implementation to read/write data from the database | 25 |  |  |  |
| Statistics use-case with NextJS | 40 |  |  |  |
| **Documentation**  - Data Model diagram.  - UI Design with screenshots and description.  - Database queries.  - Conducted tests and evidence.  - **Contribution** of each team member [-10pts if not done] | 20 |  |  |  |
| **Total** | 100 |  |  |  |
| Copying and/or plagiarism or not being able to explain or answer questions about the implementation. | -100 |  |  |  |

**Important remark: In case of copying and/or plagiarism or not being able to explain or answer questions about the implementation, you lose the whole grade.**

**\* Criteria for grading the functionality:**

- The functionality is working: you get 70% of the assigned grade.

- The functionality is not working: you lose 40% of assigned grade.

- The functionality is not implemented: you get 0.

- The remaining grade in all cases from above **is assigned to the quality of the implementation**,

- The grades are distributed on the various use cases, when the design/implementation is partial, you get only the grades of designed/implemented use cases.

Code quality criteria, include:

- Use of meaningful identifiers for variables and functions (e.g. using JavaScript naming conventions)

- Pages are responsive

- Clean code: simple and concise code, no redundancy

- Clean implementation without unnecessary files/code

- Use of comments where necessary

- Proper code formatting and indentation.

**You lose marks** for code duplication, poor/inefficient coding practices, poor naming of identifiers, unclean/untidy submission, and unnecessary complex/poor user interface design.

**Important Remark**:

**[Grades: 100-85]:** Will be given only to **fully functional application** with **all the quality criteria cited above met** and the project has excellent **design for the various functionalities**. **The report is professional**.

**[Grades: 85-80]:** Will be given only **to functional application** **with most of all the quality criteria cited above met** and the project has good design for the various functionalities. **The report is professional**.

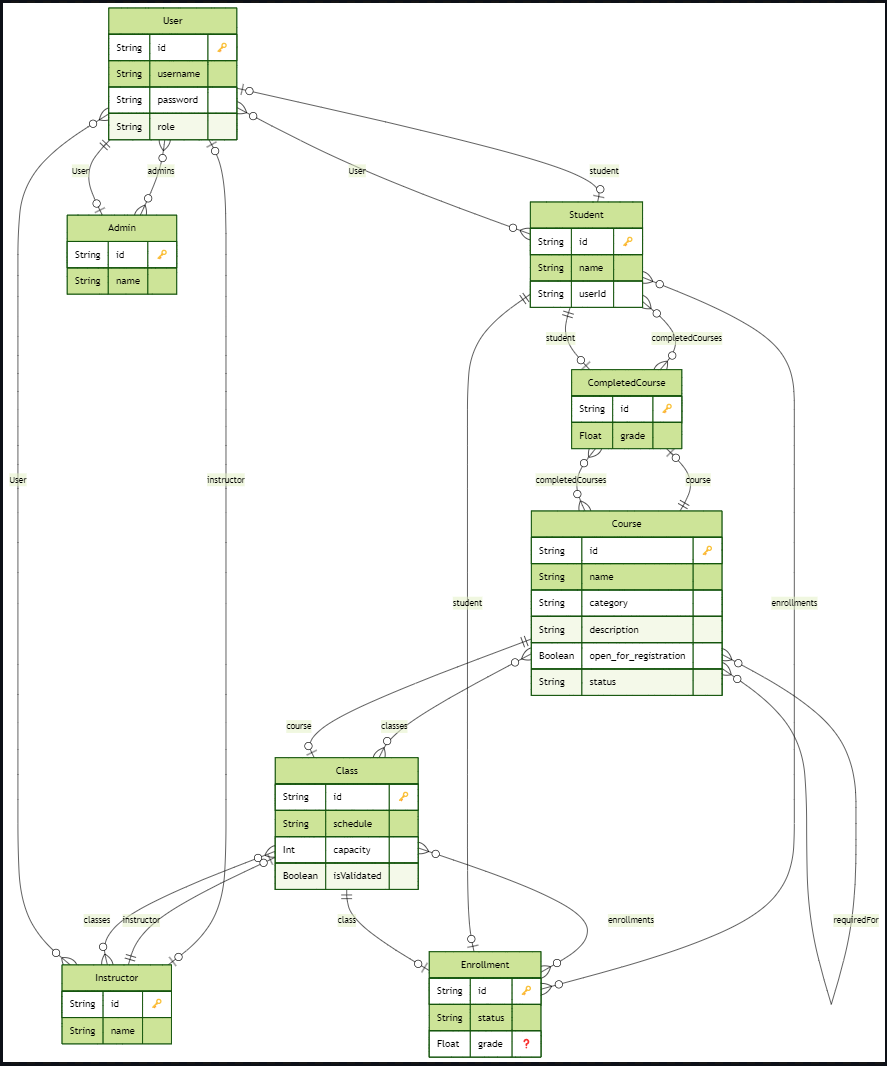
**[Grades: 80-75]:** 80% of the application functionalities are functional. The project respects partially the quality criteria. **The report is professional** but misses some information.

The grades are not negotiable. We expect that only a small portion (around 15%) of the class will be able to meet the criteria for the grades **[100-85]. You should work hard to and demonstrate the merits of your application to earn those grades.+**

# Description of your proposed platform

The proposed platform is a web-based student registration system focused on the backend development of its database and API layer. In this phase, the main goal is to design and implement a relational database using Prisma and SQLite, and to build RESTful API endpoints that interact with student, course, enrollment, and other data. These APIs support core functionalities like retrieving course statistics, tracking student progress, and managing course information.

# Data Model



# Web API, Server Actions and repository

**-** **Student Queries**:

* **findAll ()**: Retrieves all students from the database.
* **findById(id)**: Fetches a single student by their unique id.
* **create(studentData)**: Creates a new student record with the provided data.
* **update (id, studentData)**: Updates an existing student’s data based on their id.
* **delete(id)**: Deletes a student record by their id.
* **getTotalStudents()**: Returns the total number of students in the system.
* **getCompletedCourses(studentId)**: Retrieves the completed courses for a specific student.
* **getAverageStudentsGPA()**: Returns the average GPA of all students.
* **getAverageStudentsCoursesCount()**: Calculates the average number of courses completed by students.
* **getStudentLearningPath(studentId)**: Retrieves the learning path of a student, showing completed, in-progress, and pending courses.

**-** **Course Queries**:

* **findAll ()**: Retrieves all courses in the system.
* **findById(id)**: Fetches a course by its unique id.
* **create(courseData)**: Creates a new course.
* **update (id, courseData)**: Updates course details based on the provided id.
* **delete(id)**: Deletes a course by its id.
* **getCoursesByCategory(category)**: Fetches all courses in a specific category.
* **searchCoursesByName(name)**: Searches for courses by their name.

**-** **Instructor Queries**:

* **findAll ()**: Retrieves all instructors.
* **findById(id)**: Fetches a specific instructor by id.
* **getClassCountPerInstructor ()**: Retrieves the number of classes assigned to each instructor.

**- Web API**:  
The **Web API** is organized into multiple endpoints, each corresponding to a specific model:

* **/api/students**: Handles all student-related actions like fetching, creating, updating, and deleting students.
* **/api/courses**: Handles course-related actions, including querying courses by category, creating new courses, and updating existing ones.
* **/api/instructors**: Manages instructor data and their class assignments.
* **/api/statistics**: Provides statistics about students, courses, enrollments, and academic performance (e.g., top students by GPA, pass/fail rates)

### **Server Actions**

The **Server Actions** interact with the database using **Prisma**. Each action (e.g., **GET**, **POST**, **PUT**, **DELETE**) is designed to handle specific operations related to **courses**. These actions are responsible for managing course data such as fetching courses, calculating enrollments, and creating new courses.

#### **Implemented Server Actions for Courses**

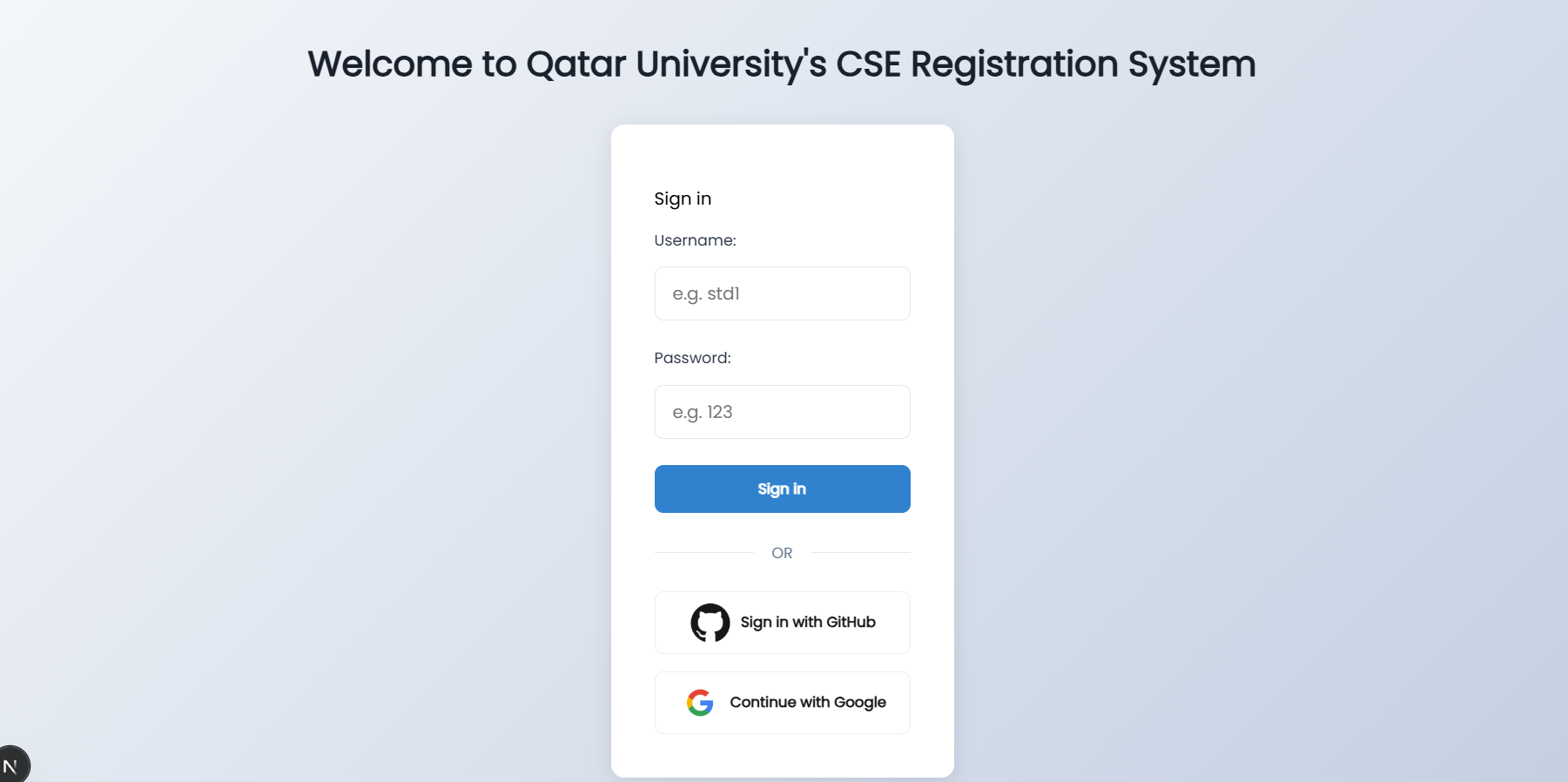
1. **Get Courses by Category**:
   * This action retrieves all courses that belong to a specific category. It returns the courses along with their associated classes.
2. **Get All Courses with Enrollment Count**:
   * This action fetches all courses and calculates the **total number of students enrolled** in each course by summing up the enrollments across all associated classes.
3. **Get Total Students Per Course**:
   * This action calculates the total number of students enrolled in each course, considering the number of enrollments in all of the course's classes.
4. **Create a New Course**:
   * This action allows the creation of a new course by accepting the necessary course data and storing it in the database.

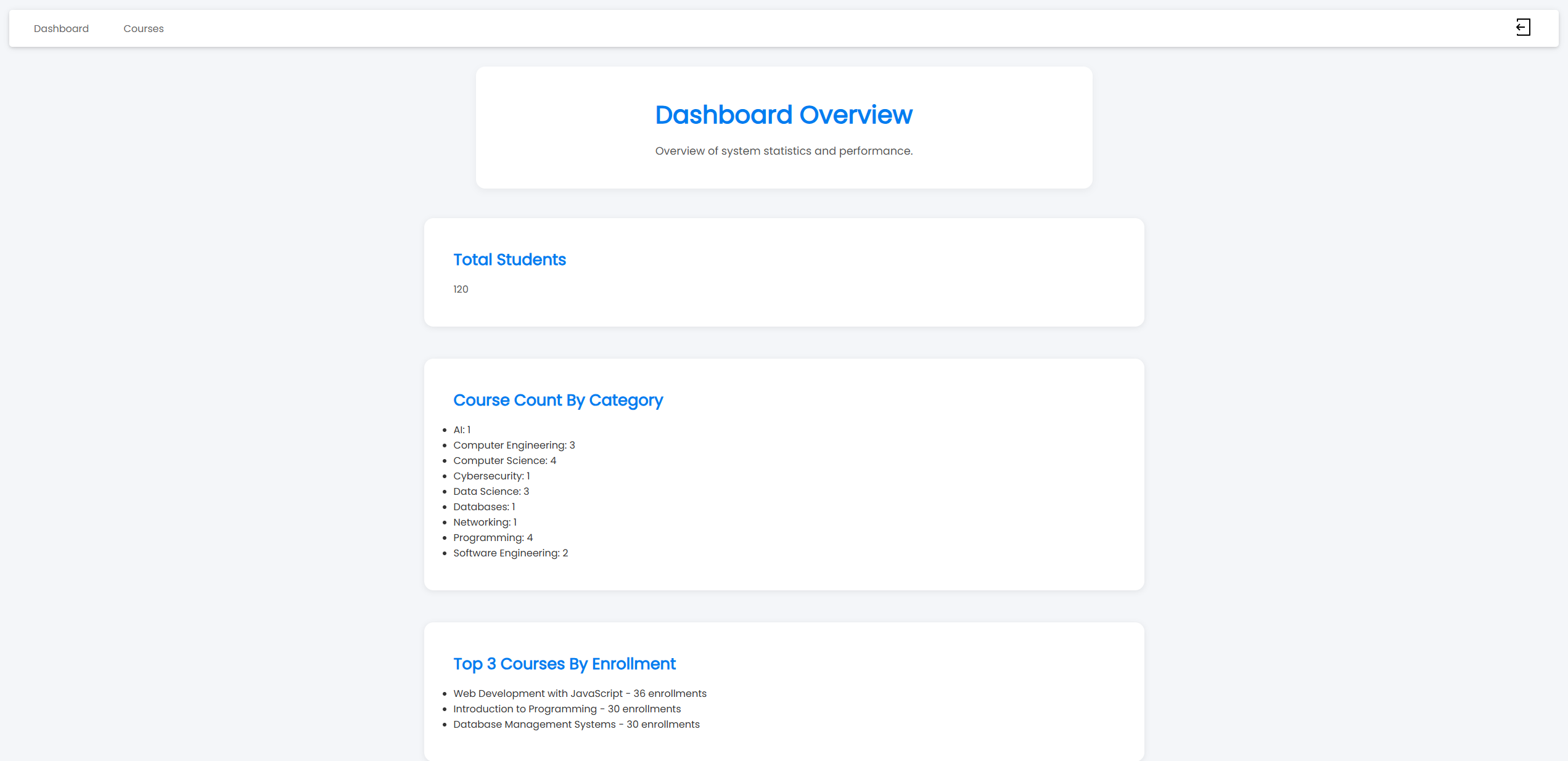
**Repository Layer**:  
The methods in the repository are responsible for querying and manipulating the database using Prisma's query methods:

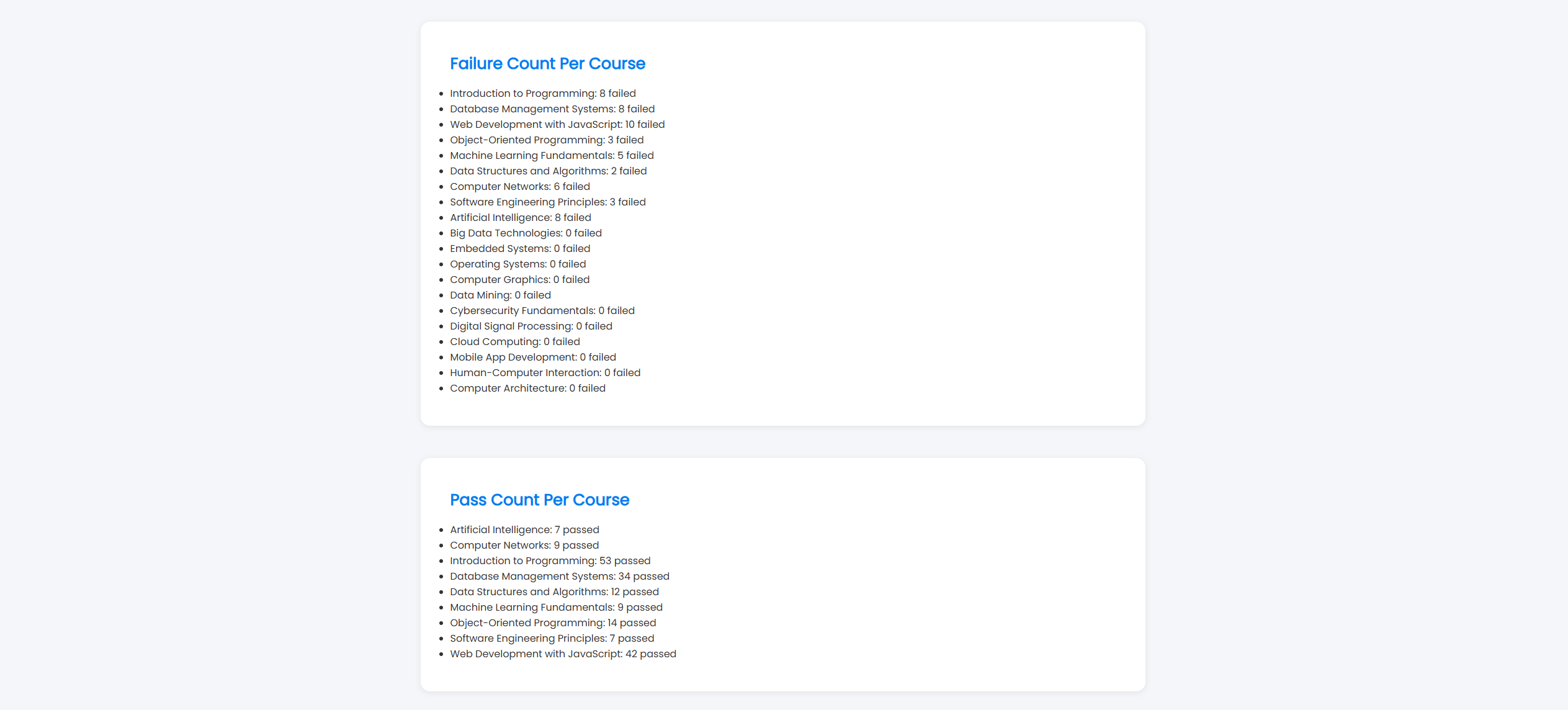
* **StudentRepo**: Handles student-related database operations like fetching, creating, and updating student records.
* **CourseRepo**: Handles course-related database operations such as retrieving course data, adding new courses, and updating them.
* **InstructorRepo**: Handles actions related to instructors and their assigned courses.

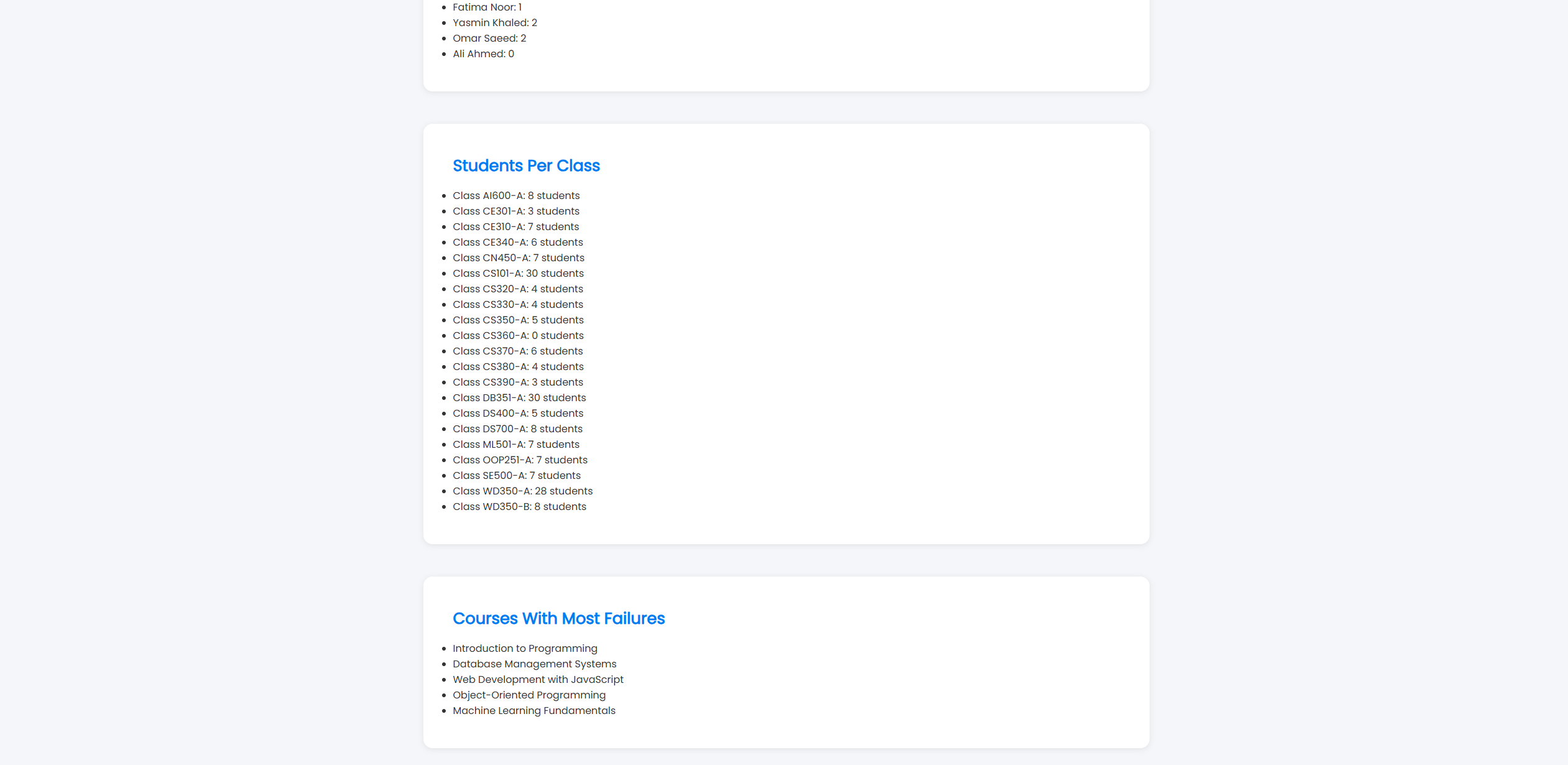
# Implemented statistics use case

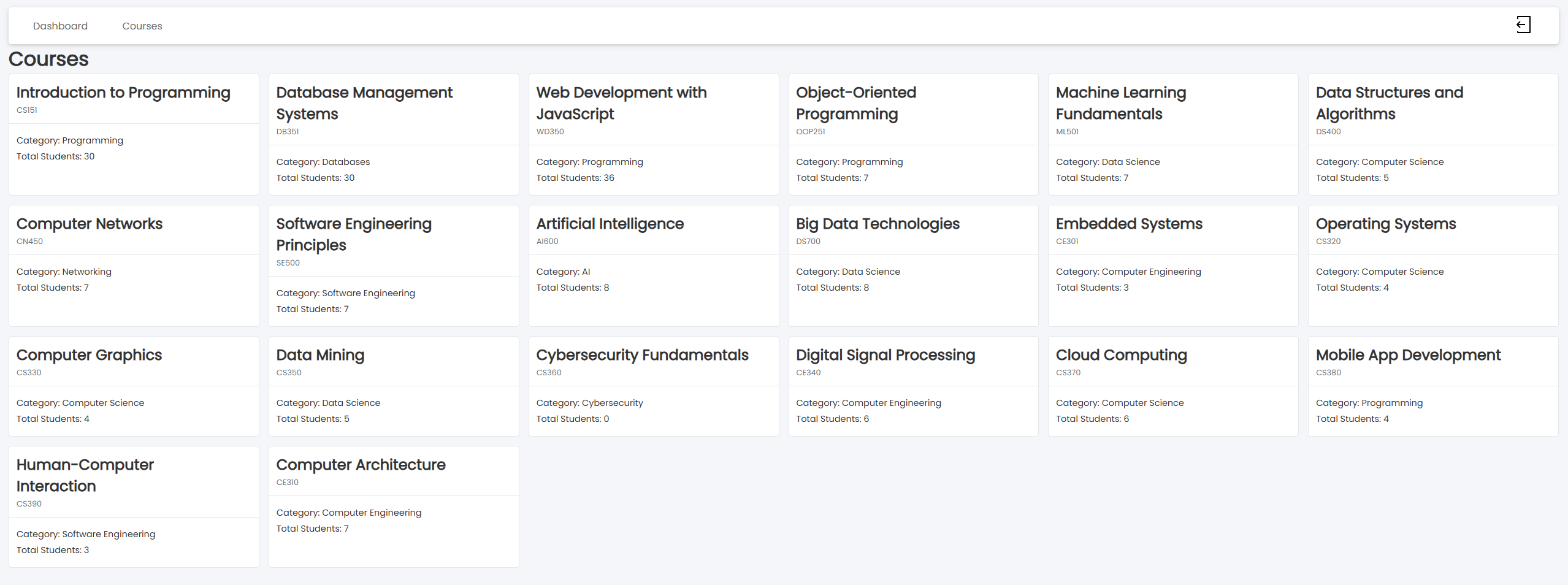
# User Interface











# Implemented queries

* + **Student Queries:**  
    Implemented queries to authenticate students using user credentials, retrieve individual student profiles, and manage their enrollments. Additionally, developed logic to fetch courses enrolled by a student and compute their GPA based on completed coursework.
  + **Course Queries:**  
    Designed queries to fetch detailed course information, manage student enrollments, and enforce prerequisite requirements. Also implemented logic to identify the most popular courses by analyzing enrollment trends across classes.
  + **Instructor Queries:**  
    Created queries to link instructors with their assigned classes and retrieve details of the courses they are teaching. This includes functionality to calculate the number of classes per instructor for workload tracking.
  + **Statistics Queries:**  
    Developed aggregation and analytical queries to generate insights related to academic performance. These include total student count, course enrollment statistics, pass/fail rates per course, class sizes, and GPA rankings for top-performing students.

# Data used in the statics

The implemented data for the statics:

**1- Total number of students**

Retrieves the number of all registered students.

**2-Course count per category**

Groups all courses by their category and returns the total count in each category.

**3-Top 3 courses by enrollment count**

Calculates the number of enrolled students in each course and returns the top three with the highest enrollment.

**4-Failure count per course (grade < 2.0)**

Identifies and counts how many students failed each course based on final grades.

**5-Pass count per course (grade ≥ 2.0)**

Groups completed courses by course ID and counts how many students passed.

**6-Class count per instructor**

Calculates how many classes are assigned to each instructor.

**7-Student count per class**

Returns the number of students enrolled in each class.

**8-Top 5 students by GPA**

Computes the GPA of each student and returns the top five students with the highest GPA.

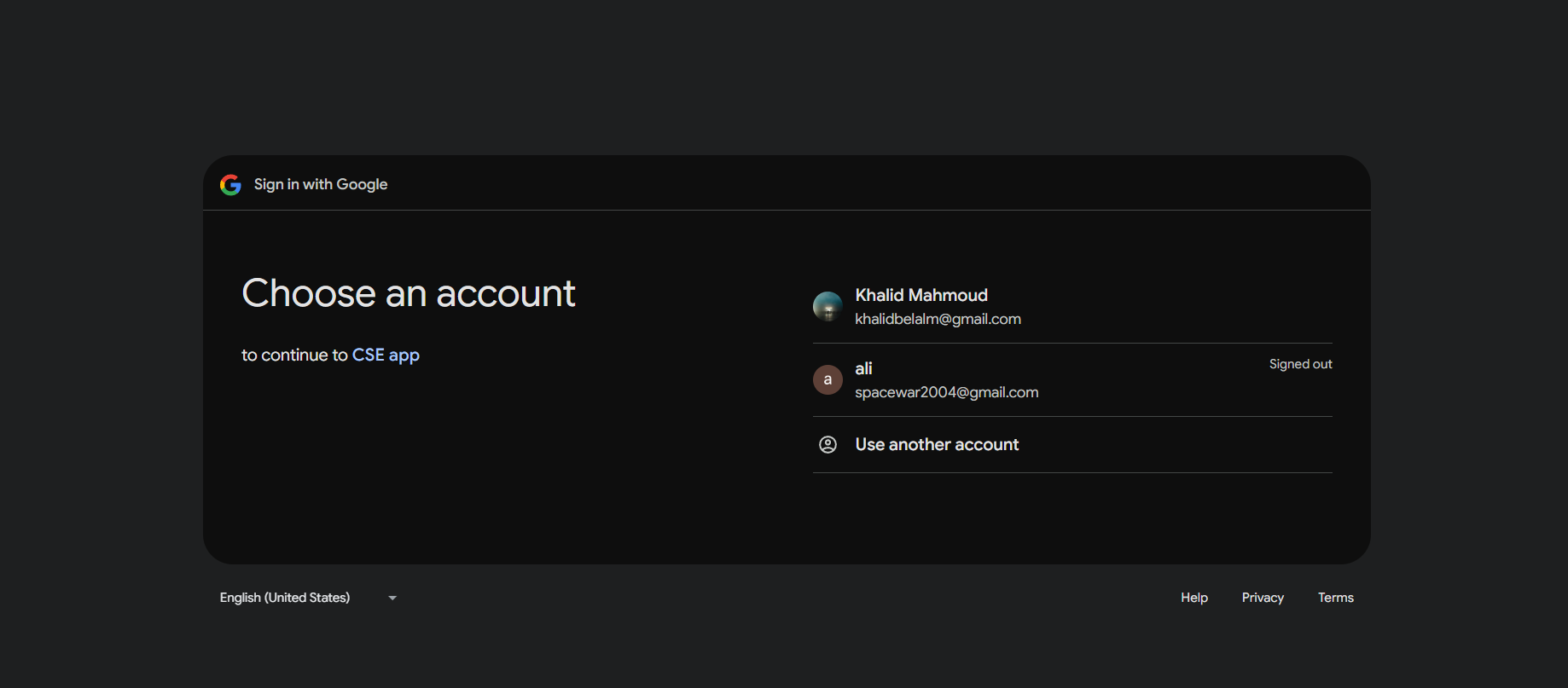
**9-Courses with most failures**

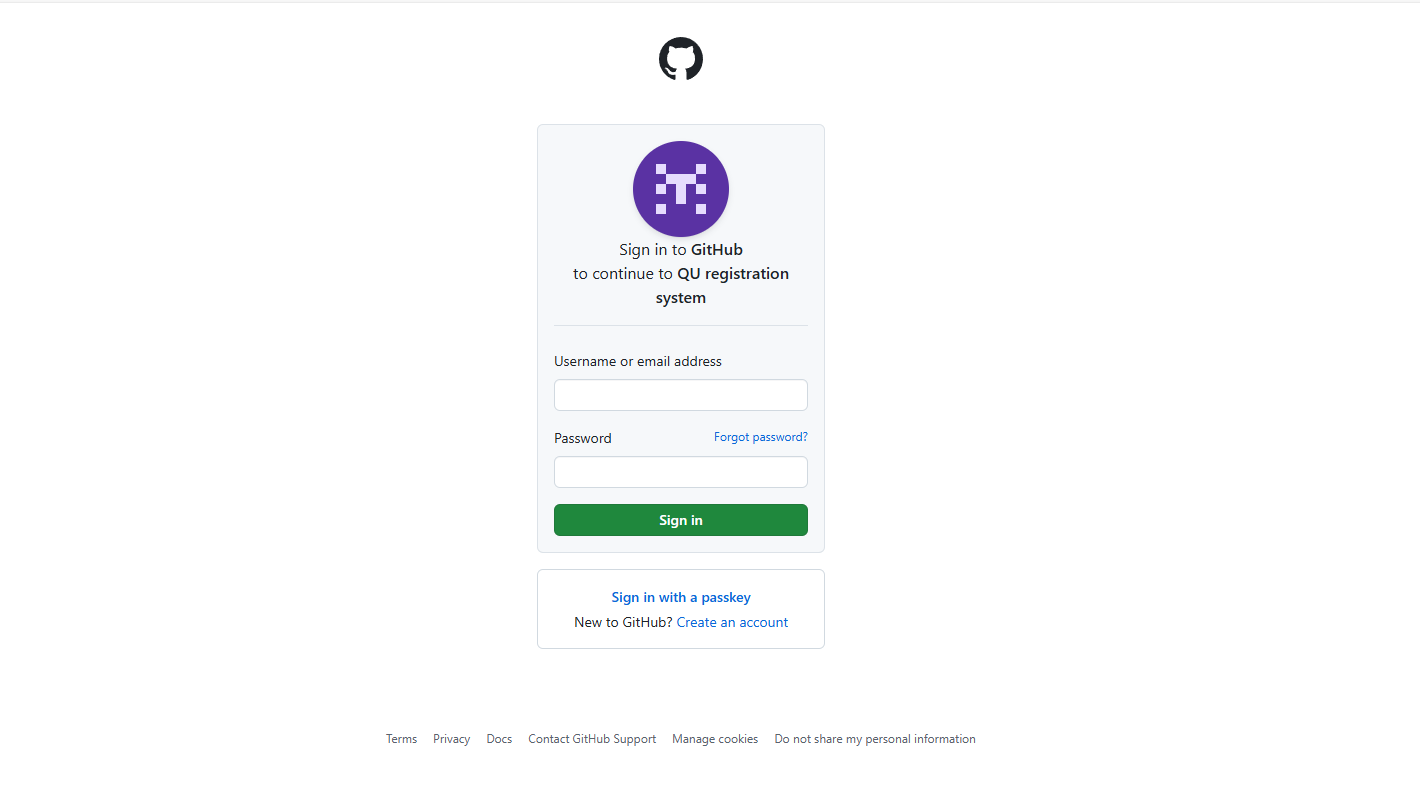
Analyzes each course to determine which has the highest failure rates.

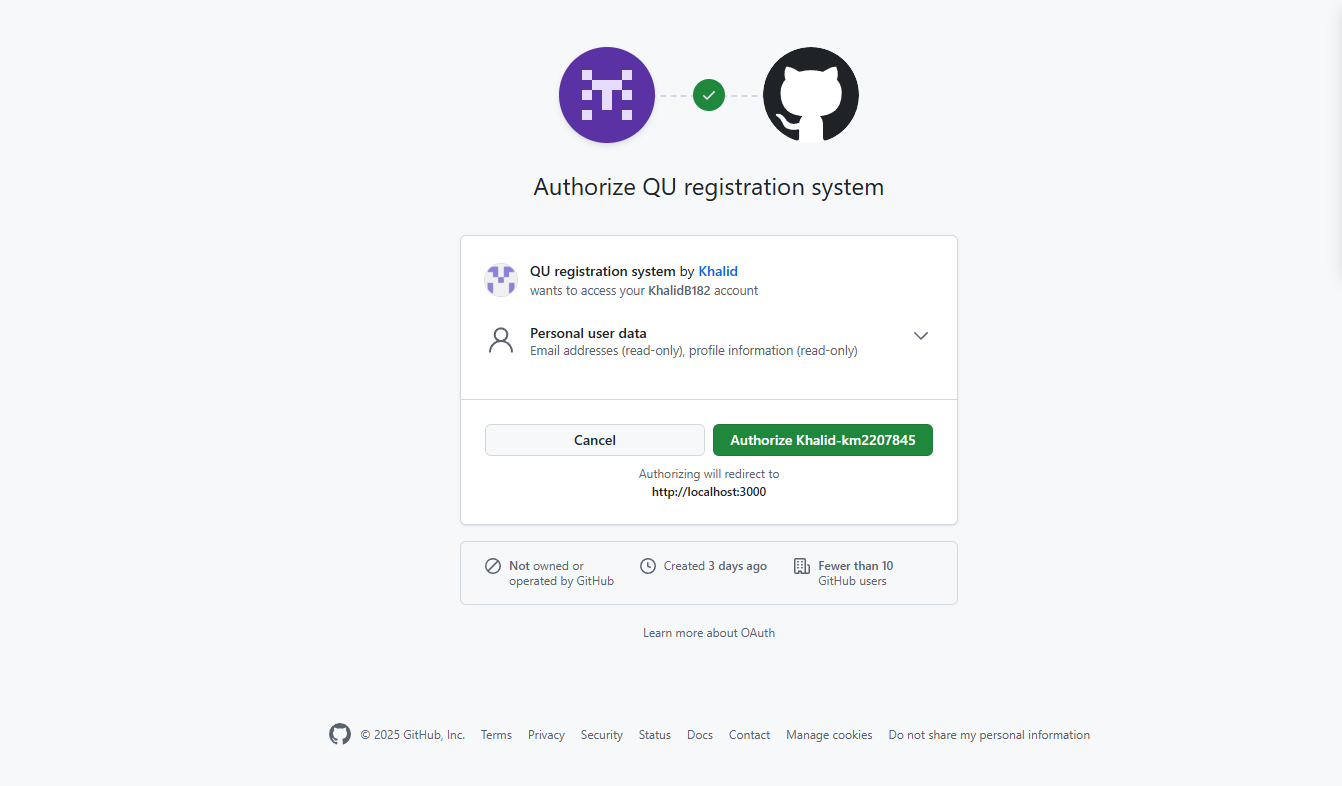
**10-Course status summary**

Groups courses by their status (e.g., active or cancelled) and counts them.

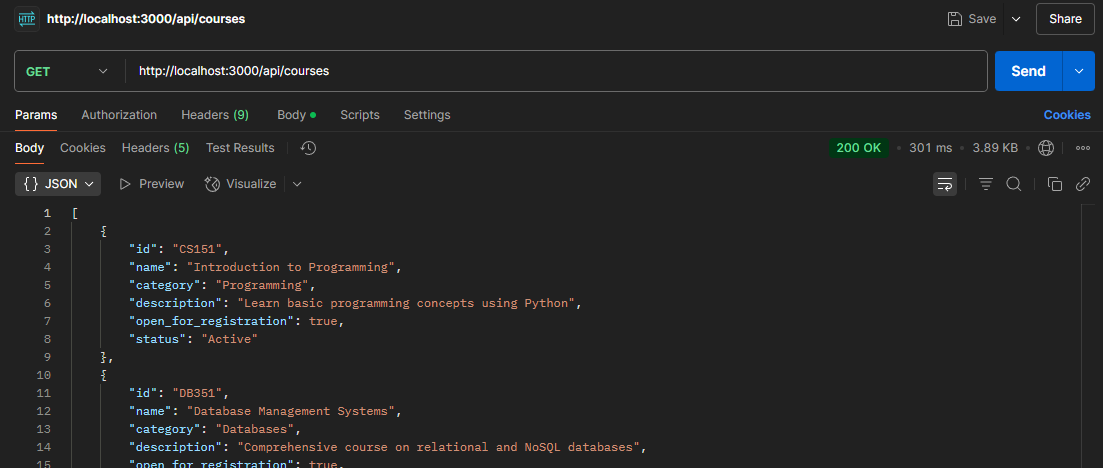
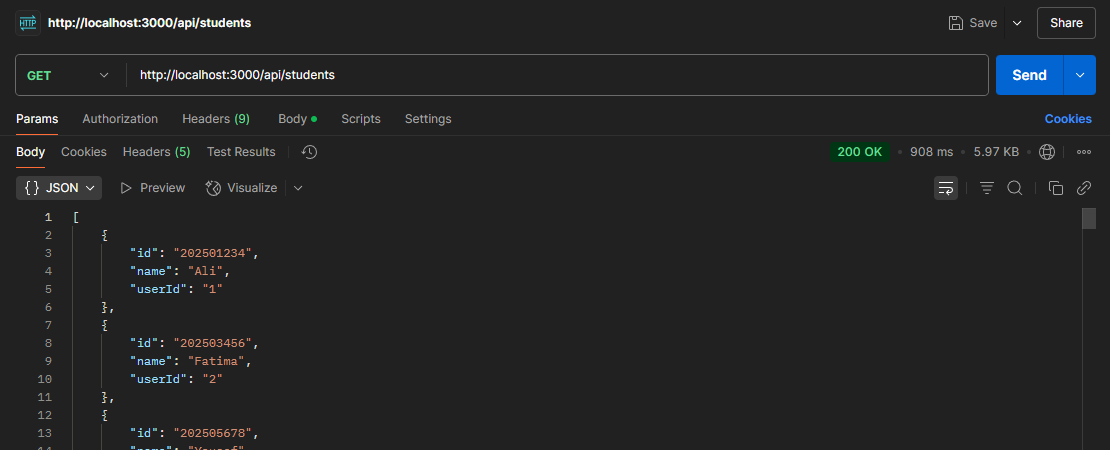
# Conducted tests







**Some PostMan testing:**



# Discussion of the project contribution of each team member

|  |  |
| --- | --- |
| **Student name** | **Student contributions** |
| Khalid Mahmoud |  I designed and implemented the database schema using Prisma.   I handled the login authorization using both credentials and GitHub and Google.   I developed the main API routes for students, courses, and enrollments etc…   I contributed to the frontend page design using CSS and React. |
| Islam Al-Absi | * Contributed to the API routes * Server Actions * Organized the Repositories * Worked on statisticsRepo |
|  |  |
|  |  |
|  |  |