**CMPS 350 Project Phase 2 – Report**

**Education Platform**

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

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

|  |  |
| --- | --- |
| **Group Members** | Ghanim Mubarak Alkuwari (202208523)  Fahrel Azki Hidayat (202206836)  Mohammed Alam (202211429)  AbdulWasay Saqib (202211598)  **Emails:**  ga2208523@qu.edu.qa  fh2206836@qu.edu.qa  ma2211429@qu.edu.qa  aw2211598@qu.edu.qa |
| **GitHub link** | [https://github.com/fahrel-fh2 HYPERLINK "https://github.com/fahrel-fh2206836/student-management-app"2 HYPERLINK "https://github.com/fahrel-fh2206836/student-management-app"06836/student-management-app](https://github.com/fahrel-fh2206836/student-management-app) |

**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 | Working (completed 100%) |  |  |
| Init DB: populate the database with the data from the json files in seed.js | 5 | Working (completed 100%) |  |  |
| Server actions, APIs and Repository Implementation to read/write data from the database | 25 | Working (completed 100%) |  |  |
| Statistics use-case with NextJS | 40 | Working (completed 100%) |  |  |
| **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 | Working (completed 100%) |  |  |
| **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**

This platform is a full-featured, role-based academic management system developed to facilitate streamlined interactions among Administrators, Instructors, and Students in an educational environment. It is structured as a web application with clean separation of responsibilities and interfaces customized for each user type:

Admin Interface: Administrators have access to tools that allow them to manage the academic structure of the institution. This includes adding new courses and sections, editing existing ones, and overseeing all data through a centralized admin dashboard. This layer ensures that the course offerings remain organized and up-to-date.

Instructor Interface: Instructors are provided with an interface to manage student performance. Through their dashboard, they can allocate grades to students enrolled in their sections and access relevant academic information. This ensures academic transparency and simplifies grade submission processes.

Student Interface: Students can register for courses, browse course listings, and monitor their academic progress via a personalized dashboard. They can also follow a “learning path” that visualizes their progress through required and elective courses, aiding in informed academic planning.

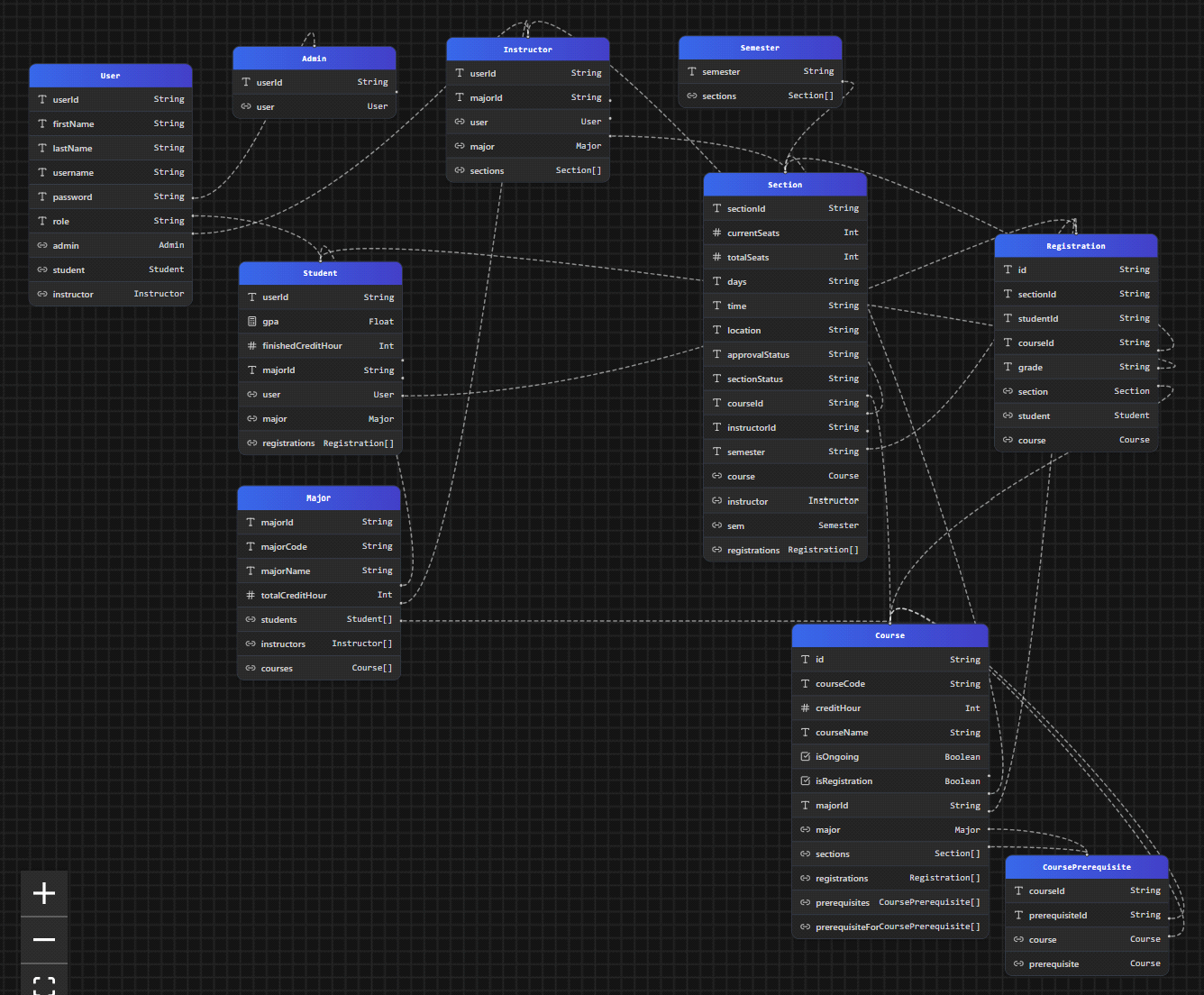
The platform integrates a Prisma ORM layer to manage all data operations through a SQLite database (dev.db). The schema is clearly defined using the schema.prisma file, outlining all models including students, users, courses, registrations, majors, and more. Initial data is populated using a seed.js script, allowing rapid development and testing with realistic data.

All interaction with the database is abstracted through a centralized repository module (app-repo.js) located in the repo directory. This promotes modularity, reusability, and clean separation of concerns across the codebase. Additionally, JSON files were initially used to simulate database functionality, providing an easy transition from static to dynamic data management.

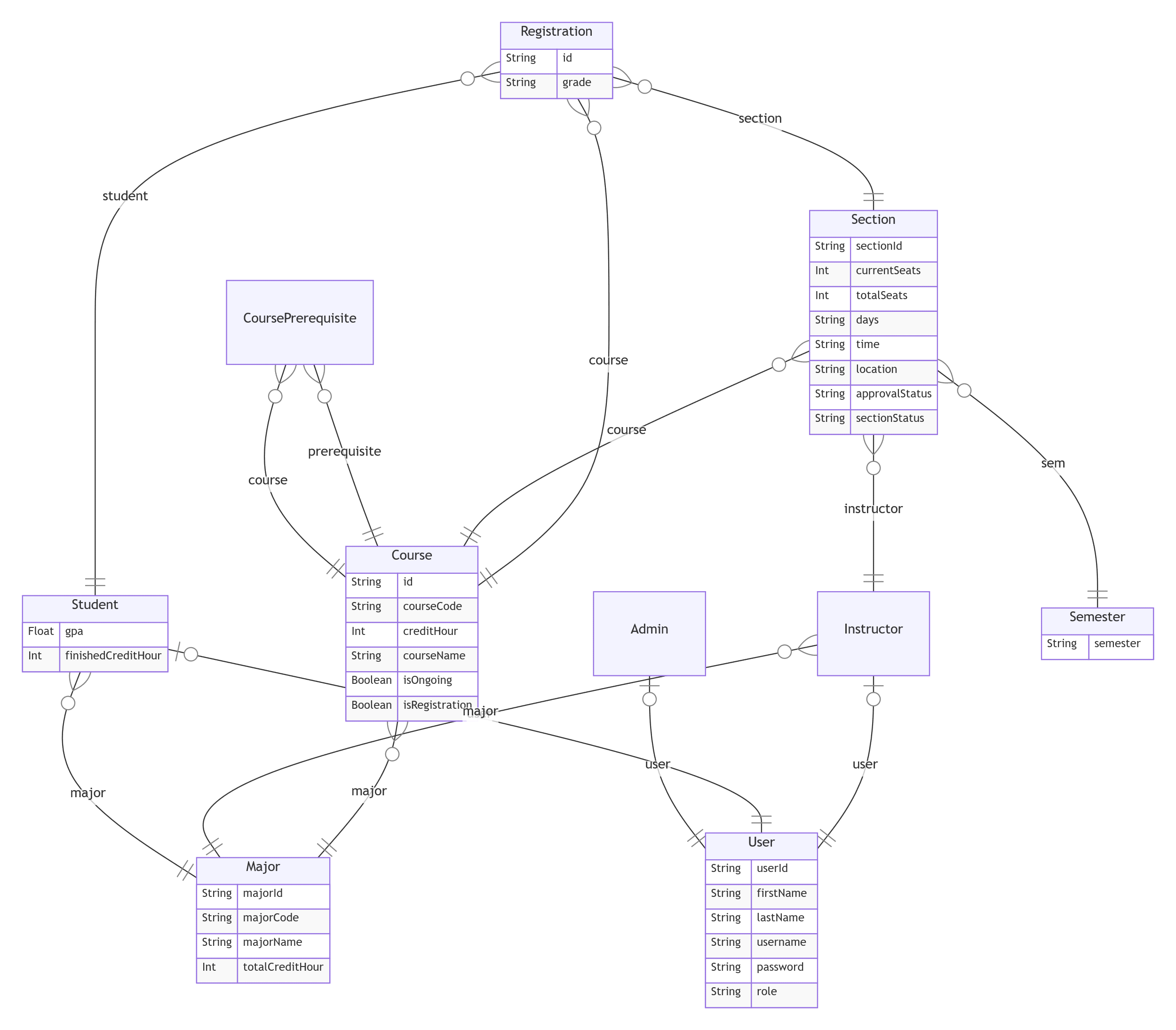
This architecture ensures scalability and maintainability, making the platform ideal for deployment in academic institutions or as a prototype for further development.

* **Data Model**

UML Diagram:



ERD Diagram:



* **Web API, Server Actions and repository**

List all your implemented methods (functions) to query your data,

Show how you organized them in WebAPI and Server actions

We have multiple methods in the appRepo.js under the repo folder. All of these methods work directly with prisma and return data from the database created through prisma. For each sub-section we have multiple get functions that allows us to be more specific in what we want to retrieve from the database. Our methods are as follows:

**User Methods**

* getUsers()
* getUser(username, password)
* getUserByEmail(username)
* getInstructors()
* updateStudent(studentId, updatedStudent)

**Section Methods**

* getSections(instructorId, sem, notSem = false, courseId, approvalStatus, sectionStatus, notApproval)
* getSectionById(sectionId)
* getInstructorTotalStudentSem(instructorId, sem)
* addSection(section)
* updateSection(sectionId, updatedSection)

**Major Methods**

* getMajors()
* getMajorById(majorId) *(you defined this twice by accident)*
* updateMajor(majorId, updatedMajor)
* getMajorByCode(majorCode)

**Course Methods**

* getCourses()
* getCourseByMajorStatus(majorId, status)
* getCourseByMajorId(majorId)
* getCourseById(id)
* getCourseByCode(code)
* getCourseByName(name)
* getCourseByNameAndMajor(name, majorId)
* addCourse(course)
* getCoursePrerequisites(id)
* updateCourse(id, updatedStatus)

**Registration Methods**

* getRegistrations(studentId, semester, sectionStatus)
* deleteRegistrations(sectionId, studentId)
* getStudentCompletedCourses(studentId, courseId)
* hasStudentCompletedCourse(studentId, courseId)
* addRegistration(reg)
* searchRegistrations(sectionId, search)
* getRegistration(studentId, sectionId)
* updateRegistration(regId, updatedReg)

**Semester Methods**

* getSemesters(removeSem)

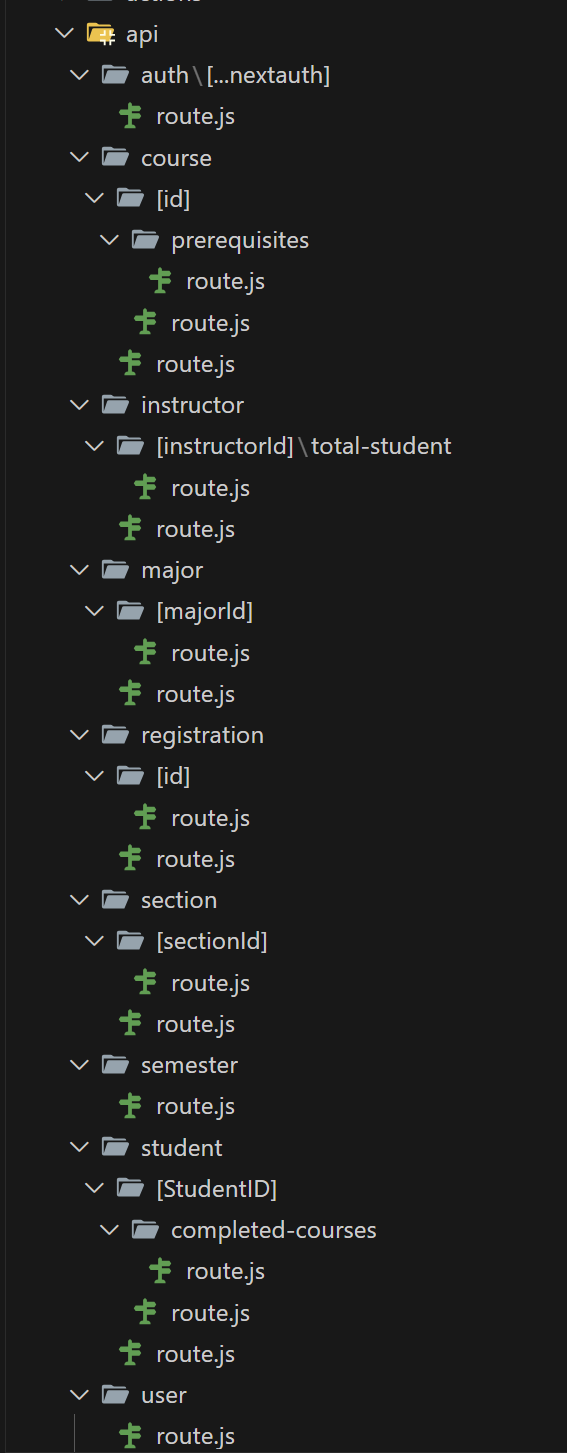
**Learning Path Methods**

* getStudentById(userId)
* getRegistrationsByStudentId(studentId)
* getCoursesByMajor(majorId) *(repeated name again but different from the previous Course method)*
* getLearningPathData(userId)

**Student Statistics Methods**

* getTotalStudentsPerMajor()
* getTop3MostEnrolledCourses()
* getPassFailRatePerCourse()
* getAverageGPAperMajor()
* getAverageGradePerCourse()
* getCompletionRateByCourse()
* getHighestFailRateCourses()
* getStudentsPerSemester()
* getTop3StudentsByGPA()
* getAverageCreditsPerMajor()

API structure:

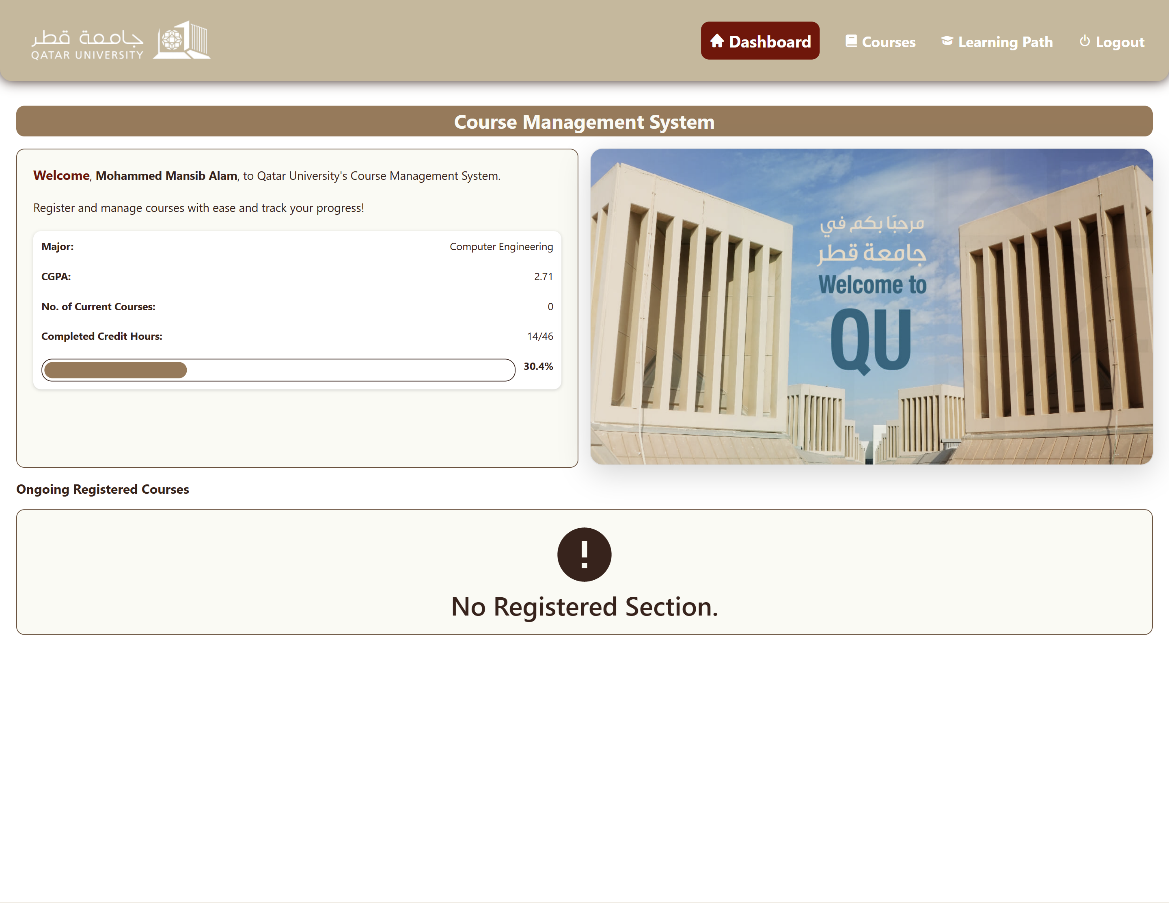


Inside the route.js file we have called the relevant method from the appRepo.js file to handle various API requests. Our API is built to follow a clear and organized structure, where each endpoint performs a specific database operation by directly utilizing the repository methods we created.

* **Implemented statistics use case**
* **User Interface**

**Student:**

Student Dashboard:



Student Learning Path:

A screenshot of a computer

AI-generated content may be incorrect.

Search for Courses:

A screenshot of a computer screen

AI-generated content may be incorrect.

Registration Screen:

A screenshot of a computer program

AI-generated content may be incorrect.

* **Implemented queries**
* **Data used in the statics**
* **Conducted tests**
* **Discussion of the project contribution of each team member**

|  |  |
| --- | --- |
| **Student name** | **Student contributions** |
| Ghanim Mubarak Alkuwari | Converted learningPath using react, next.js and prisma. I also added the 10 statistical methods in the repo |
| Mohammed Alam | Converted the course search section to fetch data from API, created multiple API and repo methods, worked on navigation, worked on the registration screen. Tested student screens |
|  |  |
|  |  |
|  |  |