**CMPS 350 Project Phase 2 – Report**

**Education Platform**

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

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

|  |  |
| --- | --- |
| **Group Members** | Aly Deyab Abdelsalam Aly (202008905)  Mahmod Amr Mostafa Abdelmawgood (202105318)  **Emails:** aa2008905@student.qu.edu.qa; ma2105318@student.qu.edu.qa |
| **GitHub link** | https://github.com/Mahmod-ma2105318/WebProject |

**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 100% |  |  |
| Init DB: populate the database with the data from the json files in seed.js | 5 | Working 100% |  |  |
| Server actions, APIs and Repository Implementation to read/write data from the database | 25 | Working 100% | Server Action without Api |  |
| Statistics use-case with NextJS | 40 | Working 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 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

**Description of the Proposed Platform**

The proposed platform is a **web-based university course management system** designed to facilitate the registration, tracking, and administration of academic courses for students, instructors, and administrators. Developed using **Next.js**, **Prisma**, and a **SQLite database**, the platform enables users to:

* **Students**:
  + View available courses and sections
  + Register for courses
  + Track their registered and completed courses
  + View their grades and progress
* **Instructors**:
  + Manage the courses they teach
  + Monitor student enrollments and performance
  + View statistical insights such as course completion rates and failure rates
* **Administrators**:
  + Add and manage users, courses, and categories
  + Monitor system-wide analytics like enrollment rates and category performance

The platform supports **user authentication and role-based access**, ensuring that only authorized users can access specific features. It also incorporates **data visualization** to enhance usability for instructors and admins.

This system is designed to be scalable, modular, and easily extendable — with future capabilities such as third-party authentication (e.g., via GitHub) and advanced analytics in mind.

# Data Model

A diagram of a username

AI-generated content may be incorrect.

# Web API, Server Actions and repository

1. **🔧 Implemented Server Action Methods**

**✅ Authentication & Session Management**

These handle user authentication and session flow:

* logout() – Logs out the currently logged-in user and redirects to /
* handleLogin(formData) – Authenticates a user and redirects based on role
* getLoggedInUser() *(called within methods)* – Fetches the current user (from repo)

**📚 Course Registration & Grading**

These modify student enrollment or grading:

* registerCourse(sectionId) – Registers a student in a section
* gradeStudent(sectionId, studentId, grade) – Assigns a grade to a student

**🛠️ Course Management**

Handles creation/editing of courses:

* addOrEditCourseAction(formData) – Adds or edits a course using submitted data

**🔍 Search Functionality**

Handles course search operations:

* searchCoursesAction(prevState, formData) – Searches courses by name/category or general match  
  *(calls internal helper functions like searchForCoursesByName, searchForCoursesByCategory, searchForCourses – assumed to be defined in repo or elsewhere)*

**✅ Registration Request Handling**

Instructor/Administrator actions to approve or reject pending registration:

* approveRegReq(userId, sectionId) – Approves registration request
* declineRegReq(userId, sectionId) – Declines registration request

**✔️ Section Validation**

Admin-level control to validate/invalidate a course section before it opens:

* validateSection(sectionId) – Marks section as valid
* invalidateSection(sectionId) – Marks section as invalid

1. **📦 Organized as Server Actions**

All the above functions are:

* **Defined in a single module** (likely inside app/repo/lib.js or a similar location)
* **Marked with 'use server'** to explicitly run on the server
* **Used for data mutation or fetching**, especially where DB interaction via Prisma or internal APIs is required
* **Often end in a redirect()** if used after a mutation (e.g., addOrEditCourseAction, logout, handleLogin)

# Implemented statistics use case

# User Interface

A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

# A screenshot of a computer AI-generated content may be incorrect.

# Implemented queries

The backend logic is implemented using **Prisma ORM** in a plain JavaScript file (lib/stats.js). A total of 10 functions were written to extract meaningful statistics from the database. Examples include:

* Total students per course and per category.
* Top 3 most taken courses.
* Failure rate per course and per category.
* Most failed course and highest success rate course.

Each function uses efficient findMany, count, and aggregation-like logic to prepare structured JSON-ready results.

# Data used in the statics

The data is based on the following models from the Prisma schema:

* Enrollment: used to track students, grades, and statuses.
* Course: used for names, categories, and structure.
* Section: links enrollments to specific courses and instructors.
* Student: connects users to their enrollments.

Grades are interpreted as follows:

* "F" = **failure**
* "A", "B", "C", "D" = **success**
* null = **not completed**

# Conducted tests

The functionality was verified by:

* Checking the UI output for each stat manually.
* Ensuring the course names appeared correctly after fixing nested Prisma selects.
* Handling empty data cases (null, no failures, no completions).
* Verifying totals matched expected counts in seed data.

Test cases include:

* Courses with no enrollments.
* Students with only successes or only failures.
* Categories with mixed outcomes.

# Implemented queries

Here is a summary of the 10 queries implemented in stats.js:

1. getTotalStudentsPerCourseCategory()
2. getTotalStudentsPerCourse()
3. getTopMostTakenCourses()
4. getFailedStudentsPerCourse()
5. getFailureRatePerCategory()
6. getTotalEnrollments()
7. getAverageCourseCompletionRate()
8. getStudentsNeverFailed()
9. getMostFailedCourse()
10. getHighestSuccessRateCourse()

All were called server-side from the JSX page and returned data used directly in rendering.

# Discussion of the project contribution of each team member

|  |  |
| --- | --- |
| **Student name** | **Student contributions** |
| Aly Aly | Repo, use case (statistics) |
| Mahmod Abdelmawgood | Schema DB, server actions |
| Pages, components and done by both of us | |
|  |  |