**CMPS 350 Project Phase 2 – Report**

**Education Platform**

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

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

|  |  |
| --- | --- |
| **Group Members** | Abdul Rahim Alsheikh Mahmoud (202210090)  Abdalrehman Daud (202203615)  Anas Elrahmani (202203650)  Emails: am2210090@student.qu.edu.qa; ad202203615@student.qu.edu.qa; ae2203650@student.qu.edu.qa; |
| **GitHub link** | Give a public link to you code (It is not acceptable to send codes by email) |

**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

## 1. Introduction

## This platform is a full-stack web application built with Next.js, designed to streamline university course management and enrollment for students, instructors, and administrators. It leverages modern technologies including React for dynamic client interfaces, server actions for secure backend logic, and Prisma ORM with a SQLite database for efficient data handling and persistence.

## 2. Technology Stack

|  |  |  |
| --- | --- | --- |
| Layer | Libraries / Services | Responsibility |
| Frontend | Next.js 14, React, CSS, React, plotly | Dynamic UI, routing, charts, form validation |
| Backend | Server Actions, Prisma ORM, Zod | Business logic, DB access, JWT handling |
| Database | SQLite | Persist courses, sections, users, enrollments |
| Auth | JWT credentials + Next‑Auth (Google / GitHub) | Secure sign‑in, role‑based cookies |
| Hosting | Node server | Edge‑optimised pages, API routes |

## 3. Core Use‑Cases

|  |  |
| --- | --- |
| Role | Capabilities |
| Student | Search/filter sections, register, withdraw, view schedule, GPA bar |
| Instructor | View own sections, Assign grades for students |
| Admin | Create/edit courses & sections, manage staff, approve or delete pending sections |
| Public | Sign in with Google/GitHub, view app statistics |

## 4. High‑Level Flow

Client components call server actions that run on the server, decode the JWT in the token cookie, perform Prisma queries, and return JSON/props back to React for rendering.

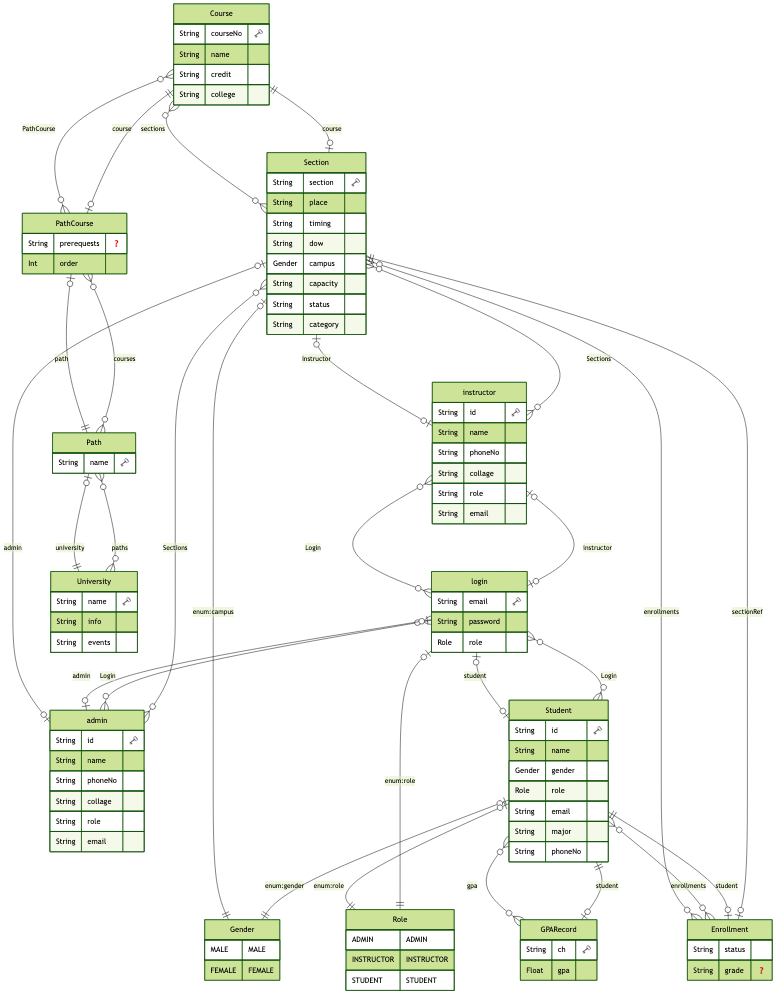
## 5. Key Pages & Components

|  |  |  |
| --- | --- | --- |
| Route | Component(s) | Highlights |
| /login | LoginForm, OAuthButtons | Credential and OAuth login |
| /student/registration | RegisterationPage, RegisterationTable | Search & register sections |
| /student/schedule | ScheduleTable | Student timetable |
| /student/summary | RegisterationTable (summary mode) | Withdraw/completion overview |
| /student/path | path | Show student progress in his/her major path |
| /settings | Settings | Profile info, change password |
| /home | Callender, graph | Show the university information and calendar with upcoming events |
| /admin/registration | RegisterationPage, RegisterationTable,  addcourse | Adding courses, approve/reject sections |
| /courses |  | Show courses for every user |
| /statistics | StatisticsPage | Aggregated metrics |
| /help&support |  | Contact information and university location |

## 6. Security & Validation

# • Passwords are bcrypt‑hashed. • JWT stored in HTTP‑only SameSite=Lax cookie. • Prisma prevents SQL injection; FK constraints enforce integrity.

# Data Model



2.2 schema:

generator client {

provider = "prisma-client-js"

}

generator erd {

provider = "prisma-erd-generator"

output = "./diagram.png"

format = "png"

}

datasource db {

provider = "sqlite"

url = env("DATABASE\_URL")

}

model Course {

courseNo String @id

name String

credit String

college String

sections Section[]

PathCourse PathCourse[]

}

model Section {

courseNo String

section String

place String

timing String

dow String

campus Gender

capacity String

status String

category String

instructorId String?

adminId String?

course Course @relation(fields: [courseNo], references: [courseNo])

instructor instructor? @relation(fields: [instructorId], references: [id])

admin admin? @relation(fields: [adminId], references: [id])

enrollments Enrollment[]

@@id([courseNo, section])

}

model Enrollment {

studentId String

courseNo String

section String

status String

grade String?

student Student @relation(fields: [studentId], references: [id])

sectionRef Section @relation(fields: [courseNo, section], references: [courseNo, section])

@@id([studentId, courseNo, section])

}

model GPARecord {

ch String

gpa Float

studentId String

student Student @relation(fields: [studentId], references: [id])

@@id([studentId, ch])

}

model admin {

id String @id

name String

phoneNo String

collage String

role String

email String

Login Login?

Sections Section[]

}

model instructor {

id String @id

name String

phoneNo String

collage String

role String

email String

Login Login?

Sections Section[]

}

model Student {

id String @id

name String

gender Gender

role Role

email String

major String

phoneNo String

gpa GPARecord[]

enrollments Enrollment[]

Login Login?

}

enum Role {

ADMIN

INSTRUCTOR

STUDENT

}

enum Gender {

MALE

FEMALE

}

model Login {

email String @id

password String

role Role

adminId String? @unique

instructorId String? @unique

studentId String? @unique

admin admin? @relation(fields: [adminId], references: [id])

instructor instructor? @relation(fields: [instructorId], references: [id])

student Student? @relation(fields: [studentId], references: [id])

@@map("login")

}

model University {

name String @id

info String

events String

paths Path[]

}

model Path {

name String @id

universityName String

university University @relation(fields: [universityName], references: [name])

courses PathCourse[]

}

model PathCourse {

pathName String

courseNo String

prerequests String?

order Int

path Path @relation(fields: [pathName], references: [name])

course Course @relation(fields: [courseNo], references: [courseNo])

@@id([pathName, courseNo])

}

# 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

**Repository :**

**Admin**

* **getAdmins(): Fetch all admins**
* **getAdmin(id): Get a specific admin by ID**
* **updateAdmin(admin): Update admin information**

**Courses & Sections**

* **getCourses(pageType, campus): Get courses filtered by page type (pending, approved) and campus**
* **getCourse(courseNo): Get a specific course**
* **addCourse(course): Create or update a course**
* **addCourseWithSection(data): Create course with section in one step**
* **getSectionsByFilter({ status, campus, courseNo, college }): Filter sections dynamically**
* **updateCourse(course): Update course data**
* **getSectionById(courseNo, section): Get full section data with course, instructors, and enrollments**
* **updateSectionStatus(courseNo, section, newStatus): Update section's approval status**
* **deleteSection(courseNo, section): Delete a section**

**Student**

* **getStudents(): Fetch all students**
* **getStudent(id): Get a student by ID**
* **updateStudent(student): Update student data**
* **registerStudentInSection(studentId, courseNo, section): Register a student**
* **unregisterStudentFromSection(studentId, courseNo, section): Withdraw a student from a section**
* **getStudentSchedule(studentId): Get a student’s enrolled courses and schedule**
* **updateGrade(courseNo, section, studentId, grade): Update a student's grade**

**Instructor**

* **getInstructors(): Fetch all instructors**
* **getInstructor(id): Get a specific instructor**
* **updateinstructor(instructor): Update instructor data**

**Authentication**

* **getUsers(): Get all users from the login table**
* **getUser(email, password): Validate user and return corresponding profile**
* **changePass(email, currentPassword, newPassword): Change a user's password**
* **updateUser(user): Update user login info**

**University Info & Paths**

* **getUni(): Get university general information**
* **getPaths(): Get all academic paths with their courses**
* **getPath(name): Get detailed path info and ordered courses**

**Server Action:**

**Authentication**

* **loginAction(email, pass): Authenticate the user, generate JWT, and store it in cookies**
* **getUserFromToken(user): Retrieve full user details (admin, instructor, or student) based on decoded token**

**Home**

* **uniInfoAction(): Fetch university name, description, and events**

**User**

* **getCoursesAction(user): Return all course sections a user is enrolled in**
* **changePassAction(email, pass, newPass): Change password after verifying the current one**

**Courses & Sections**

* **getCoursesRegisterationAction({ status, campus, courseNo, college }): Fetch sections based on optional filters**
* **handelAddCourseAction(formData): Parse and sanitize form data to add a course and section**
* **updateSectionStatusAction(courseNo, section, newStatus): Update the approval status of a section**
* **deleteSectionAction(courseNo, section): Delete a section using its composite key**

**Enrollment**

* **registerStudentAction(studentId, courseNo, section): Register a student into a course section**
* **unregisterStudentAction(studentId, courseNo, section): Withdraw a student from a course section**

**Schedule**

* **getStudentScheduleAction(studentId): Return all enrollments for a student along with section and course info**

**Grades**

* **getSectionGrades(courseNo, sectionID): Get list of students and grades for a section**
* **updateGradeAction(courseNo, section, studentId, grade): Update a student's grade in a section**

**Paths**

* **getPathAction(name): Retrieve learning path and ordered courses**

**Statistics**

* **getStatistics(): Return usage and academic statistics from the platform**

# Implemented statistics use case

# User Interface

# Implemented queries

# Data used in the statics

# Conducted tests 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.

# Discussion of the project contribution of each team member

|  |  |
| --- | --- |
| **Student name** | **Student contributions** |
| Abdul Rahim Alsheikh Mahmoud | 202210090 |
| Abdalrehman Daud | 202203615 |
| Anas Elrahmani | 202203650 |