**CMPS 350 Project Phase 2 – Report**

**Education Platform**

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

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

|  |  |
| --- | --- |
| **Group Members** | Student1 full name (StudentId)  Student2 full name (StudentId)  Student3 full name (StudentId)  **Emails:** student1@student.qu.edu.qa; student2@student.qu.edu.qa; student3@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 Next.js application manages university courses and enrollments for students, instructors and admins. It combines React client components, server actions, Prisma ORM and a SQLite database.

## 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.

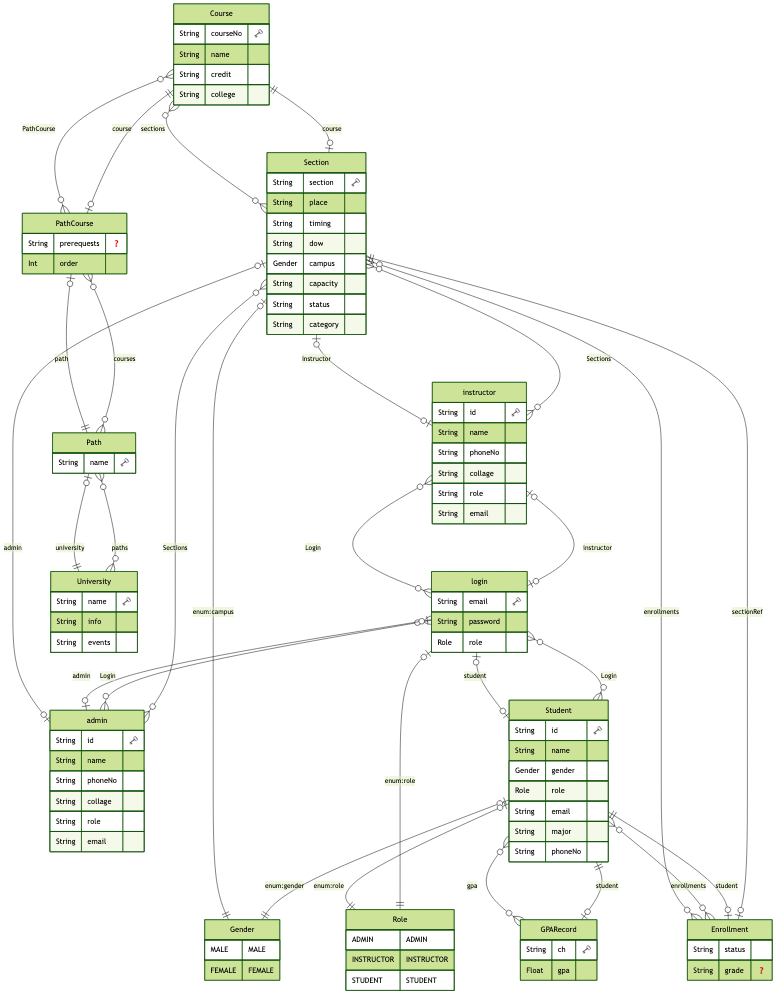
## 5. Key Pages & Components

|  |  |  |
| --- | --- | --- |
| 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

# Implemented statistics use case

# User Interface

# Implemented queries

# Data used in the statics

# Conducted tests

# Implemented queries

# Discussion of the project contribution of each team member

|  |  |
| --- | --- |
| **Student name** | **Student contributions** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |