**CMPS 350 Project Phase 2 – Report**

**Education Platform**

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

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

|  |  |
| --- | --- |
| **Group Members** | Amer Alzawawi (202205054)  Abdelrahman Abdelwahed (202207121)  Yaser Osama Radwan (202205739)  **Emails:** aa2205054@student.qu.edu.qa; aa2207121@student.qu.edu.qa; yr2205739@student.qu.edu.qa; |
| **GitHub link** | https://github.com/abdelrahman1425376/Chill-Guys.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 | 10 |  |  |
| Init DB: populate the database with the data from the json files in seed.js | 5 | 5 |  |  |
| Server actions, APIs and Repository Implementation to read/write data from the database | 25 | 25 |  |  |
| Statistics use-case with NextJS | 40 | 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 | 20 |  |  |
| **Total** | 100 | 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

We have implemented our data using Prisma and seeded it by using “seed.js” to populate the database with our JS files. We have created our “courses”, “register” and “user” repositories that contain all the needed functions to add/edit/delete from the database. Also, we created web API files that GET, POST, PUT, DELETE data from our database. Finally, we created the final use case using NextJS and React that contains students and teachers most important stats.

# Data Model

Give entity diagram, Prisma schema,

# Web API, Server Actions and repository

Repo:

* cources-Repo:

1. getAllCources() that gets all meals.
2. getCourceByName(Name) that gets a course by its name.
3. getCourceByCategory(category) that gets courses by their category.

* regester-repo:

1. updateGrade(reg) updates the grades of a registration record.
2. updateSeats(reg) updates the availability of seats of a class.
3. createRegister(reg) creates a new registration record.
4. isCompleteCource(prerequisite, name) checks if a student has a completed prerequisite course.
5. checkRepeatSameCource(courceName, name) Checks if a student has already registered for the same course.
6. getAllClassess() returns all class records.
7. getAllRegister() returns all registration records.
8. getClass(Id) returns a class with a specific id.
9. getCourceStatusByStudent(stat, namee) returns all registered courses that match the given stat (if it is completed or in-progress).
10. getRegisterByInfo(name, CRN, sname) returns an in-progress registered course of a student by giving the instructor name, student name, CRN.
11. getRegisterByInfo(name) returns all in-progress registrations based on a specific instructor.

* user-repo:

1. countStudents() count all students.
2. countAdmins() counts all admins.
3. countTeachers() counts all teachers.
4. Countusers() counts all users, regardless of their roles.
5. AvgStudents() returns the average students per users.
6. AvgAdmins() returns the average admins per users.
7. AvgTeachers() returns the average teachers per users.
8. getUser(username, password) return the first match of the username and password provided.
9. getTeacherWithMostStudents() returns the teacher with the most registerations.
10. getTeacherWithLeastStudents() returns the teacher with the fewest registerations.
11. getHighestSubjectWithHighFail() returns the course with highest “F” grade.

API:

* classess:
  + /route.js (GET: gets all classes)
  + /[id]/route.js (GET: get a class by id, PUT: updates the availability of seats)
* count:
  + /route.js (returns all the statistics of the new use case)
* cources:
  + /route.js (GET: gets all the courses)
  + /category/[category]/route.js (GET: get courses by the category)
  + /name/[name]/route.js (GET: get a course by its name)
* register:
  + /route.js (GET: gets all registrations, POST: creates a new registration, PUT: updates a grade)
  + /instructor/[name]/route.js (GET: get registrations in-progress by instructor)
  + /repeat/[...params]/route.js (GET: check if course has been taken already)
  + /Status/[...params]/route.js (GET: get courses by status)
  + /submitGrade/[...params]/route.js (GET: get an in-progress registration of a student to handle submission of grade)
* Users:
  + /[…params]/route.js (GET: get user by the username and password)

# 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** |
| Amer Alzawawi | Made the api using nextJS and React in two use cases and made the new use case.(35%) |
| Abdelrahman Abdelwahed | Made the api using nextJS and react in three use cases.(35%) |
| Yaser Osama Radwan | Made the api using nextJS and React in one use case and the prisma and the seeding.(30%) |
|  |  |
|  |  |