|  |  |  |
| --- | --- | --- |
|  | | |
|  | **CMPS 350 Project Phase 2 – WebApp UI Design and Implementation**  **E Commerce Platform**  **(15% of the course grade)**  **The project code is accessible on the following (github) link:** | |
| **Group Id:** | | G? |
| **Group Members:** | | Abdulrazzaq Mohammed Alsiddiq (202004464)  **Emails:** aa2004464@qu.edu.qa; |

**Grading Rubric - In the Functionality column please specify either: *Working (completed x%)*, *Not Working (completed x%)* or *Not done*.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Criteria** | **Weight%** | **Functionality\***  **(implementation percentage)** | **Quality of the implementation** | **Your Grade** |
| Design and implement the Data Model. | 10 |  |  |  |
| Init DB: populate the database with the data from the json files in seed.js | 5 |  |  |  |
| 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 |  |  |  |
| Bonus - successful deployment of the app and the Database to a cloud hosting service such as <https://vercel.com/> | 5 |  |  |  |
| Copying and/or plagiarism or not being able to explain or answer questions about the implementation. | 0 |  |  |  |

***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 fully 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 iformation.

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:

The platform of phase 2 is a continuation of phase one website with some modifications.

The modifications:

* Implementing a web API using Next.js routing.
* Replacing data from Jason files by a real database using Prisma
* Modifying the repo.js to utilize Prisma queries to search and manipulate data.
* Implementing the new statistics use case to facilitate the viewing of useful statistics.

# A screenshot of a computer Description automatically generatedData Model:

# Database Initialization:

# The models:

|  |
| --- |
| // This is your Prisma schema file,  // learn more about it in the docs: https://pris.ly/d/prisma-schema  generator client {    provider = "prisma-client-js"  }  datasource db {    provider = "sqlite"    url      = env("DATABASE\_URL")  }  generator dbml {    provider = "prisma-dbml-generator"  }  model Seller {    id            Int     @id @default(autoincrement())    company\_name  String    username      String  @unique    password      String    bank\_account  String  @unique    sellHistory   Item[]    money\_balance Decimal  }  model Customer {    id              Int        @id @default(autoincrement())    name            String    surname         String    country         String    username        String     @unique    password        String    money\_balance   Decimal    purchaseHistory Purchase[]  }  model Item {    id             Int     @id @default(autoincrement())    name           String    category       String    price          Decimal    description    String    owner\_username String    quantity       Int    url            String    Seller   Seller @relation(fields: [sellerId], references: [id], onUpdate: Cascade, onDelete: Cascade)    sellerId Int  }  model Purchase {    purchaseNo Int      @id @default(autoincrement())    itemId     Int    country    String    quantity   Int    date       DateTime    amount     Decimal    Customer   Customer @relation(fields: [customerId], references: [id], onUpdate: Cascade, onDelete: Cascade)    customerId Int} |

# The seed.js file that is used to populate the data into the database:

|  |
| --- |
| import fs from 'fs-extra'  import path from 'path'  import { PrismaClient } from '@prisma/client'  const prisma = new PrismaClient()  const customersPath = path.join(process.cwd(), 'dataPhase2/customer.json')  const sellersPath = path.join(process.cwd(), 'dataPhase2/seller.json')  const purchasesPath = path.join(process.cwd(), 'dataPhase2/purchase.json')  async function main() {      try {            const sellers = await fs.readJSON(sellersPath)          const customers = await fs.readJSON(customersPath)          const purchases = await fs.readJSON(purchasesPath)          for (const customer of customers) await prisma.customer.create({ data: customer })          for (const purchase of purchases) await prisma.purchase.create({ data: purchase })          for (const seller of sellers) await prisma.seller.create({ data: seller })                console.log("Seeded successfully");      } catch (error) {          console.log(error);          return { error: error.message }      }  }  main()      .then(async () => {          await prisma.$disconnect()      })      .catch(async (e) => {          console.error(e)          await prisma.$disconnect()          process.exit(1)      }) |

We must not forget to add “type”:”module” to the package.json file before running the seed.js file after that we should remove the “type”:”module” statement from the package.json.

# Description of the implemented Statistics use-case

# Description

# What has been implemented

# What is not implemented

# List of implemented database queries

# Conducted Tests and evidence:

# API Testing using Postman:

# Discussion of the project contribution of each team member:

|  |  |
| --- | --- |
| Student | Contribution (%) |
| Abdulrazzaq Mohammed Alsiddiq (202004464) | 100% |