CMPS-350 Phase-2

Mohammed Faheem Ali Zaidi – 202109963  
Sheikh Hasin Ishrak – 202108209  
Nafin Ahsan Kabir Mahmoud – 201913152

Report

|  |  |  |
| --- | --- | --- |
|  | **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:** | | Mohammed Faheem Ali Zaidi [202109963]  Sheikh Hasin Ishrak (202108209)  Nafin Ahsan Kabir Mahmoud (201913152)  **Emails:** mz2109963@student.qu.edu.qa; sa2108209@student.qu.edu.qa; nm1913152@student.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 | Working 100% |  |  |
| Init DB: populate the database with the data from the json files in seed.js | 5 | Working 100% |  |  |
| APIs and Repository Implementation to read/write data from the database | 25 | Working 100% |  |  |
| 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 | 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.**

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Index** | **Page No.** |
|  |
| 1 | Data Model & Description | 4 |  |
| 2 | Database Initialization | 6 |  |
| 3 | Statistics Use Cases | 13 |  |
| 4 | Testing | 17 |  |
| 5 | Contributions | 27 |  |

**Data Model & Description**

A screenshot of a computer

Description automatically generated

1. **User**:
   * Represents a user.
   * Each user possesses a unique **id**, **username**, and **password**.
   * The “**type”** field indicates the type of user, i.e., a buyer, seller, or an admin.
   * Users can have some optional personal information such as **name**, **surname**, **money\_balance**, **company\_name**, or **bank\_account**.
   * Fields responsible for storing shipping address information are **contact\_person\_name**, **street**, **apartment\_suite\_number**, **city**, **state**, **zip\_code**, and **mobile\_number**, mainly for buyers.
   * The user has well defined relationships with purchases and products.
2. **Product**:
   * Represents an item listed for sal.
   * Each product has a unique **id**.
   * There are information fields such as **year**, **make**, **model**, **type**, **price**, **distance**, **image\_url**, and **isSold**, each carrying some information associated with the product.
   * There is a foreign key **seller\_id** referencing the **id** of the user [seller] who lists the product for sale.
   * A product has well defined relationships with users [sellers] and purchases.
3. **Purchase**:
   * Represents a transaction.
   * Each purchase has a unique purchase **id**.
   * There are foreign keys **buyer\_id** and **product\_id** referencing the **id** of the user [buyer] who made the purchase and the **id** of the product being purchased respectively.
   * It includes a **purchase\_date** to denote the date of purchase.
   * A purchase has well defined relationships with users [buyers] and products.

This model allows users to register, list products for sale, purchase products, as well as manage their information. It also maintains clear and concise relationships between users, products, and purchases to facilitate efficient e-commerce transactions.

**Database Initialization**

import { PrismaClient } from '@prisma/client';

const prisma = new PrismaClient();

async function main() {

    // Create buyers

    const buyers = await prisma.user.createMany({

        data: [

        {

            id: 1,

            type: "buyer",

            username: "BUYER1",

            password: "BUYER1@Scambox",

            name: "buyer1\_name",

            surname: "buyer1\_surname",

            money\_balance: 1000,

            contact\_person\_name: "Buyer 1",

            street: "123 Buyer Street",

            apartment\_suite\_number: "Apt 404",

            city: "Doha",

            state: "Qatar",

            zip\_code: "22744",

            mobile\_number: "123-456-7890"

        },

        {

            id: 2,

            type: "buyer",

            username: "BUYER2",

            password: "BUYER2@Scambox",

            name: "buyer2\_name",

            surname: "buyer2\_surname",

            money\_balance: 1500,

            contact\_person\_name: "Buyer 2",

            street: "456 Buyer Street",

            apartment\_suite\_number: "Apt 101",

            city: "Mesaieed",

            state: "Qatar",

            zip\_code: "22744",

            mobile\_number: "987-654-3210"

        },

        {

            id: 3,

            type: "buyer",

            username: "BUYER3",

            password: "BUYER3@Scambox",

            name: "buyer3\_name",

            surname: "buyer3\_surname",

            money\_balance: 2000,

            contact\_person\_name: "Buyer 3",

            street: "789 Buyer Street",

            apartment\_suite\_number: "Apt 303",

            city: "Lucknow",

            state: "India",

            zip\_code: "90210",

            mobile\_number: "111-222-3333"

            }

            ]

        });

    // Create sellers

    const sellers = await prisma.user.createMany({

        data: [

        {

            id: 4,

            type: "seller",

            username: "SELLER1",

            password: "SELLER1@Scambox",

            company\_name: "Seller Company 1",

            bank\_account: "123456789"

        },

        {

            id: 5,

            type: "seller",

            username: "SELLER2",

            password: "SELLER2@Scambox",

            company\_name: "Seller Company 2",

            bank\_account: "987654321"

        }

        ]

    });

    // Create admins

    const admins = await prisma.user.createMany({

        data: [

        {

            id: 6,

            type: "admin",

            username: "ADMIN1",

            password: "ADMIN1@Scambox"

        }

        ]

    });

    //console.log("Sellers:", sellers[1]);

    // Create products

    const products = await prisma.product.createMany({

        data: [

        {

            id: 1001,

            year: 2000,

            make: "Toyota",

            model: "Corolla",

            type: "sedan",

            price: 50000,

            distance: 250000,

            image\_url: "https://www.cars.com/i/large/in/v2/stock\_photos/3e36757b-ec3b-455e-82e7-eb6f65c82695/eb348de0-cb6b-409a-9ddd-fde9efcc4b20.png",

            seller\_id: 4

        },

        {

            id: 1002,

            year: 2015,

            make: "Toyota",

            model: "Rav4",

            type: "suv",

            price: 60000,

            distance: 30000,

            image\_url: "https://ymimg1.b8cdn.com/resized/car\_model/1509/pictures/1158171/mobile\_listing\_main\_2014\_Toyota\_Rav4\_Front\_2.jpg",

            seller\_id: 4

        },

        {

            id: 1003,

            year: 2020,

            make: "Honda",

            model: "Accord",

            type: "coupe",

            price: 70000,

            distance: 12000,

            image\_url: "https://static1.hotcarsimages.com/wordpress/wp-content/uploads/2020/10/Accord-EX-L-V6-Coupe-Front.jpg",

            seller\_id: 5

        },

        {

            id: 1004,

            year: 2012,

            make: "Nissan",

            model: "Tiida",

            type: "hatchback",

            price: 80000,

            distance: 2500,

            image\_url: "https://media.drive.com.au/obj/tx\_rs:fit:1920:1080,q:50,w:1920/driveau/upload/cms/uploads/NIH2m6nQSa2pxpGJk7Gn",

            seller\_id: 5

        },

        {

            id: 1005,

            year: 2017,

            make: "Polestar",

            model: "1",

            type: "hyper",

            price: 90000,

            distance: 0,

            image\_url: "https://static1.srcdn.com/wordpress/wp-content/uploads/2020/09/car-in-need-for-speed-heat-.jpg",

            seller\_id: 5

        },

        {

            id: 1006,

            year: 2017,

            make: "Dodge",

            model: "Viper",

            type: "sports",

            price: 100000,

            distance: 0,

            image\_url: "https://di-uploads-pod16.dealerinspire.com/kendalldodgechryslerjeepram1/uploads/2021/10/Viper-10-Kendall-Dodge.jpg",

            seller\_id: 4

        },

        {

            id: 1007,

            year: 2020,

            make: "Chevrolet",

            model: "Camaro SS",

            type: "convertible",

            price: 110000,

            distance: 0,

            image\_url: "https://file.kelleybluebookimages.com/kbb/base/house/2020/2020-Chevrolet-Camaro-FrontSide\_CHCAMRSS2001\_640x480.jpg",

            seller\_id: 5

        },

        {

            id: 1008,

            year: 2024,

            make: "Porsche",

            model: "911 GT3 R",

            type: "sports",

            price: 120000,

            distance: 0,

            image\_url: "https://mediaassets.pca.org/pages/pca/images/content/img\_9(3).jpg",

            seller\_id: 4

        },

        {

            id: 1009,

            year: 2021,

            make: "Mercedes",

            model: "CLE",

            type: "coupe",

            price: 130000,

            distance: 150,

            image\_url: "https://www.mercedes-benz.co.uk/content/dam/hq/passengercars/cars/cle/cle-coupe-c236-pi/modeloverview/06-2023/images/mercedes-benz-cle-coupe-c236-model-overview-696x392-06-2023.png",

            seller\_id: 5

        },

        {

            id: 1010,

            year: 2024,

            make: "Koenigsegg",

            model: "Gemera",

            type: "hyper",

            price: 140000,

            distance: 0,

            image\_url: "https://media.drive.com.au/obj/tx\_rs:fit:1920:1080,q:50,w:1920/caradvice/private/iippmjm5pwsz350e6hxb",

            seller\_id: 5

        },

        {

            id: 1011,

            year: 2024,

            make: "Toyota",

            model: "Camry",

            type: "sedan",

            price: 150000,

            distance: 0,

            image\_url: "https://i.gaw.to/vehicles/photos/40/36/403605-2024-toyota-camry.jpg?640x400",

            seller\_id: 5

        },

        {

            id: 1012,

            year: 2009,

            make: "Ford",

            model: "Fusion",

            type: "sedan",

            price: 160000,

            distance: 125000,

            image\_url: "https://i.gaw.to/vehicles/photos/00/98/009825\_2009\_Ford\_Fusion.jpg?1024x640",

            seller\_id: 5

        },

        ]

    });

    // Create purchases

    const purchases = await prisma.purchase.createMany({

        data: [

        {

            buyer\_id: 1,

            product\_id: 1001

        },

        {

            buyer\_id: 2,

            product\_id: 1002

        },

        {

            buyer\_id: 3,

            product\_id: 1003

        },

        {

            buyer\_id: 2,

            product\_id: 1005

        },

        {

            buyer\_id: 2,

            product\_id: 1008

        },

        {

            buyer\_id: 3,

            product\_id: 1011

        },

        {

            buyer\_id: 2,

            product\_id: 1012

        }

        ]

    });

    // Output entire data for products and purchases

    console.log("Buyers:");

    console.log(buyers);

    console.log("Sellers:");

    console.log(sellers);

    console.log("Admins:");

    console.log(admins);

    console.log("Products:");

    console.log(products);

    console.log("Purchases:");

    console.log(purchases);

    console.log("Database seeded successfully!");

}

main()

    .catch((e) => {

        console.error(e);

        process.exit(1);

    })

    .finally(async () => {

        await prisma.$disconnect();

    });

**Statistics Use-Cases**

1. **Description**

The implemented statistics involve the **total purchases** made on the platform and total revenue generated, the buyers per state, the **top 3 selling product** types, the **average** product pricing across the platform, the **sold vs unsold** statistics and the top 3 most **successful seller** statistics.

1. **Implementations**

A StatisticsChart.jsx file and a page.jsx file.

1. **Unimplementations**

Plagiarism was not implemented

1. **Implemented data queries**

1. GET /api/stats/1-total-purchases Total Purchase Statistics

A screenshot of a computer program

Description automatically generated

2. GET /api/stats/2-buyers-per-state Buyers Per State Statistics

A screenshot of a computer

Description automatically generated

3. GET /api/stats/3-top-products-on-sale Top Products Statistics

A screenshot of a computer

Description automatically generated

4. GET /api/stats/4-average-pricing Average Pricing Statistics

A screenshot of a computer

Description automatically generated

5. GET /api/stats/5-sold-to-unsold Sold/Unsold Statistics

A screenshot of a computer

Description automatically generated

6. GET /api/stats/6-top-sellers Top Seller Statistics

A screenshot of a computer

Description automatically generated

**Testing - Database Queries**

1. GET /api/products Returns all Products

A screenshot of a computer

Description automatically generated

2. POST /api/products Creates a Product

A screenshot of a computer

Description automatically generated

3. GET /api/products/[type] Returns Specific Products

A screenshot of a computer

Description automatically generated

5. PUT /api/products/[id] Updates a Specific Product

A screenshot of a computer

Description automatically generated

6. DEL /api/products/[id] Deletes a Specific Product

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

7. GET /api/purchases Return all Purchases

A screenshot of a computer

Description automatically generated

8. POST /api/purchases Creates a Purchase

A screenshot of a computer

Description automatically generated

9. GET /api/purchases/[purchase\_id] Returns a Specific Purchase

A screenshot of a computer

Description automatically generated

10. PUT /api/purchases/[purchase\_id] Updates a Specific Purchase

A screenshot of a computer

Description automatically generated

11. DEL /api/purchases/[purchase\_id] Deletes a Specific Purchase

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

12. GET /api/users Returns all Users

A screenshot of a computer

Description automatically generated

13. POST /api/users Creates a User

A screenshot of a computer

Description automatically generated

14. GET /api/users/[user\_id] Returns a Specific User

A screenshot of a computer

Description automatically generated

15. PUT /api/users/[user\_id] Updates a Specific User

A screenshot of a computer program

Description automatically generated

16. DEL /api/users/[user\_id] Deletes a Specific User

A screenshot of a computer

Description automatically generated  
A screenshot of a computer

Description automatically generated

**Testing - User Interface**

The remaining is the same as phase-1, with all storage references now using the APIs.

**Contributions**

List of Accomplishments:

* Migration to Nextjs environment.
* Repo File Creation.
* 16 Data API Route Creations.
* 6 Statistics API Route Creations.
* Prisma Installation & Schema Creation.
* Seed File Creation.
* Nextjs Use Case UI Creation.
* Report Creation.
* Deployment [Currently Incomplete].

Contribution Table:

|  |  |
| --- | --- |
| **Name** | **Contributions** |
|  |
| Mohammed Faheem Ali Zaidi | Migration to Nextjs environment.  Repo File Creation.  16 Data API Route Creations.  6 Statistics API Route Creations.  Prisma Installation & Schema Creation.  Seed File Creation.  Nextjs Use Case UI Creation.  Report Creation. |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| Sheikh Hasin Ishrak | Populating seed.js with JSON data  Api/products/route.js  Api/stats /2-buyers-per-state /route.js  Api/stats /4-average-pricing/route.js  Api/stats /6-top-sellers/route.js  Global.css |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| Nafin Mahmoud | Next.js api routes  Repo  Prisma  Deployment |  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |