|  |  |  |
| --- | --- | --- |
|  | | |
|  | **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:** | | Muhammad Fauzan Aristya Putra 202104747  Marwan Ghassan Khankan 202108477  Mohammed Abdo Mohammed Al-ghazali 201704262  Osamah Motea Ahmed Alsumaitti 202006040  **Emails:** [mp2104747@student.qu.edu.qa](mailto:mp2104747@student.qu.edu.qa); [ma1704262@student.qu.edu.qa](mailto:ma1704262@student.qu.edu.qa); [mk2108477@student.qu.edu.qa](mailto:mk2108477@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 (completed 100%)*,** |  |  |
| Init DB: populate the database with the data from the json files in seed.js | 5 | ***Working (completed 100%)*,** |  |  |
| APIs and Repository Implementation to read/write data from the database | 25 | ***Working (completed 100%)*,** |  |  |
| Statistics use-case with NextJS | 40 | ***Working (completed 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 (completed 100%)*,** |  |  |
| **Total** | 100 | ***Working (completed 100%)*,** |  |  |
| Bonus - successful deployment of the app and the Database to a cloud hosting service such as <https://vercel.com/> | 5 | ***Working (completed 100%)*,** |  |  |
| 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

Our platform is supposed to be a clone of the famous drone selling website [DJI](https://www.dji.com/global).

# Data Model



**SQL tables:**

Table User {

id Int [pk, increment]

type String [not null, default: 'customer']

firstName String [not null]

lastName String [not null]

email String [unique, not null]

phone String [not null]

password String [not null]

avatarColor String [not null]

balance Float [not null]

createdAt DateTime [default: `now()`, not null]

updatedAt DateTime [not null]

customer Customer

}

Table Customer {

id Int [pk, increment]

shippingAddresses ShippingAddress

orders Order [not null]

transactions Transaction [not null]

user User [not null]

userId Int [unique, not null]

}

Table ShippingAddress {

id Int [pk, increment]

label String [not null, default: 'Qatar University']

street String [not null, default: 'Street 1234']

city String [not null, default: 'Doha']

country String [not null, default: 'Qatar']

url String [not null, default: 'https://www.google.com/maps/place/Qatar+University/@25.3755282,51.4861504,17z/data=!3m1!4b1!4m6!3m5!1s0x3e45dd1faaf07c23:0x21f2193775153df1!8m2!3d25.3755282!4d51.4887307!16s%2Fm%2F05pdfng?entry=ttu']

customer Customer [not null]

customerId Int [unique, not null]

Order Order [not null]

indexes {

(label, street, city, country) [unique]

}

}

Table Order {

id Int [pk, increment]

quantity Int [not null]

subtotal Float [not null]

shippingFee Float [not null]

total Float [not null]

dateTime DateTime [default: `now()`, not null]

estimatedArrival DateTime [not null]

shippingAddress ShippingAddress [not null]

shippingAddressId Int [not null]

customer Customer [not null]

customerId Int [not null]

product Product [not null]

productId Int [not null]

indexes {

(customerId, productId, quantity, dateTime) [unique]

}

}

Table Product {

id Int [pk, increment]

name String [unique, not null]

modelName String [not null]

description String [not null]

price Float [not null]

quantity Int [not null]

weight Float [not null]

flightTime Float [not null]

rating Float [not null]

numberOfReviews Int [not null]

numberOfSales Int [not null]

numberOfOngoingOrders Int [not null]

imageUrl String [not null]

createdAt DateTime [default: `now()`, not null]

updatedAt DateTime [not null]

series Series [not null]

seriesId Int [not null]

features Feature [not null]

includedItems QuantizedIncludedItem [not null]

faqs Question [not null]

model Model [not null]

modelId Int [not null]

orders Order [not null]

}

Table Series {

id Int [pk, increment]

name String [unique, not null]

description String [not null]

products Product [not null]

}

Table Feature {

id Int [pk, increment]

name String [unique, not null]

imageUrl String [not null]

products Product [not null]

}

Table QuantizedIncludedItem {

id Int [pk, increment]

quantity Int [not null]

item IncludedItem [not null]

itemId Int [not null]

products Product [not null]

indexes {

(quantity, itemId) [unique]

}

}

Table IncludedItem {

id Int [pk, increment]

name String [not null]

imageUrl String [not null]

products QuantizedIncludedItem [not null]

indexes {

(name, imageUrl) [unique]

}

}

Table Question {

id Int [pk, increment]

question String [not null]

answer String [not null]

products Product [not null]

indexes {

(question, answer) [unique]

}

}

Table Model {

id Int [pk, increment]

url String [unique, not null]

positionX Float [not null]

positionY Float [not null]

positionZ Float [not null]

rotationX Float [not null]

rotationY Float [not null]

rotationZ Float [not null]

scale Float [not null]

cameraX Float [not null]

cameraY Float [not null]

cameraZ Float [not null]

products Product [not null]

}

Table Transaction {

id Int [pk, increment]

amount Float [not null]

dateTime DateTime [default: `now()`, not null]

type String [not null]

customer Customer [not null]

customerId Int [unique, not null]

}

Table FeatureToProduct {

featuresId Int [ref: > Feature.id]

productsId Int [ref: > Product.id]

}

Table ProductToQuantizedIncludedItem {

includeditemsId Int [ref: > QuantizedIncludedItem.id]

productsId Int [ref: > Product.id]

}

Table ProductToQuestion {

faqsId Int [ref: > Question.id]

productsId Int [ref: > Product.id]

}

Ref: Customer.userId - User.id

Ref: ShippingAddress.customerId - Customer.id

Ref: Order.shippingAddressId > ShippingAddress.id

Ref: Order.customerId > Customer.id

Ref: Order.productId > Product.id

Ref: Product.seriesId > Series.id

Ref: Product.modelId > Model.id

Ref: QuantizedIncludedItem.itemId > IncludedItem.id

Ref: Transaction.customerId > Customer.id

# Database Initialization

**Objective:** Database Initialization refers to the process of populating the database with default or initial data to ensure that the application has the necessary data to function properly from the start.

**Implementation:** The database initialization script is responsible for seeding the database with default data using services provided by the ProductService and UsersService classes. These services encapsulate the logic for adding default data to the respective tables in the database.

**Procedure:**

1. **Populating Users Data:** The script invokes the addDefaultData method from the UsersService class to add default user data to the database. This includes creating user profiles with basic information such as name, email, password, etc.
2. **Populating Products Data:** Next, the script calls the addDefaultData method from the ProductService class to add default product data to the database. This includes creating product records with details such as name, description, price, quantity, etc.

**Usage:**

1. Developers can execute this database initialization script during the setup or deployment process of the application to ensure that the database is pre-populated with essential data.
2. The script ensures that the application can function correctly immediately after deployment, as it provides the necessary data for user interaction and product management.

**Error Handling:** The script includes error handling mechanisms to catch and log any errors that may occur during the initialization process. If an error occurs, the script logs the error message and gracefully exits while disconnecting from the Prisma client.

**Note:**

* The @ts-nocheck directive is used at the top of the script to disable TypeScript checking for this file, allowing for flexibility in handling types and avoiding unnecessary type errors during initialization.

# Description of the implemented Statistics use-case

# Description

**Objective:** The implemented Statistics use-case aims to provide valuable insights and analytics about the E-commerce.

**Key Features:**

1. **Total Number of Products:** Provides an overview of the total number of products available on the platform.
2. **Total Sales Revenue:** Calculates and displays the total revenue generated from all sales transactions.
3. **Average Order Value:** Calculates and presents the average value of each order placed on the platform.
4. **Top Selling Products:** Identifies and lists the top-selling products based on the number of sales.
5. **Total Number of Users:** Displays the total count of registered users on the platform.
6. **Average Product Rating:** Computes the average rating of all products based on user reviews.
7. **Total Number of Orders:** Presents the total count of orders placed by users.
8. **Sales Revenue by Date Range:** Calculates and displays the total sales revenue within a specified date range.

# What has been implemented

The use-case is implemented using Next.js and React for the frontend, Prisma ORM for database interactions, and Node.js for backend API development. The backend API routes fetch data from the database using the ProductService class methods, which encapsulate the business logic for calculating statistics.

**Documentation:**

1. **GET /api/products/total-products:** Retrieves the total number of products available on the E-commerce platform.
2. **GET /api/products/total-revenue:** Fetches the total sales revenue generated from all transactions.
3. **GET /api/products/average-order-value:** Calculates the average value of each order placed on the platform.
4. **GET /api/products/top-selling-products:** Retrieves a list of the top-selling products based on sales volume.
5. **GET /api/products/total-users:** Fetches the total count of registered users on the platform.
6. **GET /api/products/average-product-rating:** Calculates the average rating of products based on user reviews.
7. **GET /api/products/total-orders:** Retrieves the total count of orders placed by users.
8. **GET /api/products/sales-revenue-by-date-range:** Calculates the total sales revenue within a specified date range.

**Usage:** Developers and stakeholders can utilize these API endpoints to gain insights into various aspects of the E-commerce platform's performance. The retrieved statistics can be used for strategic planning, marketing campaigns, product optimization, and overall business growth.

# What is not implemented

none

# List of implemented database queries

The implemented database queries in the ProductService class provide various functionalities for interacting with product-related data in the database.

**List of Implemented Queries:**

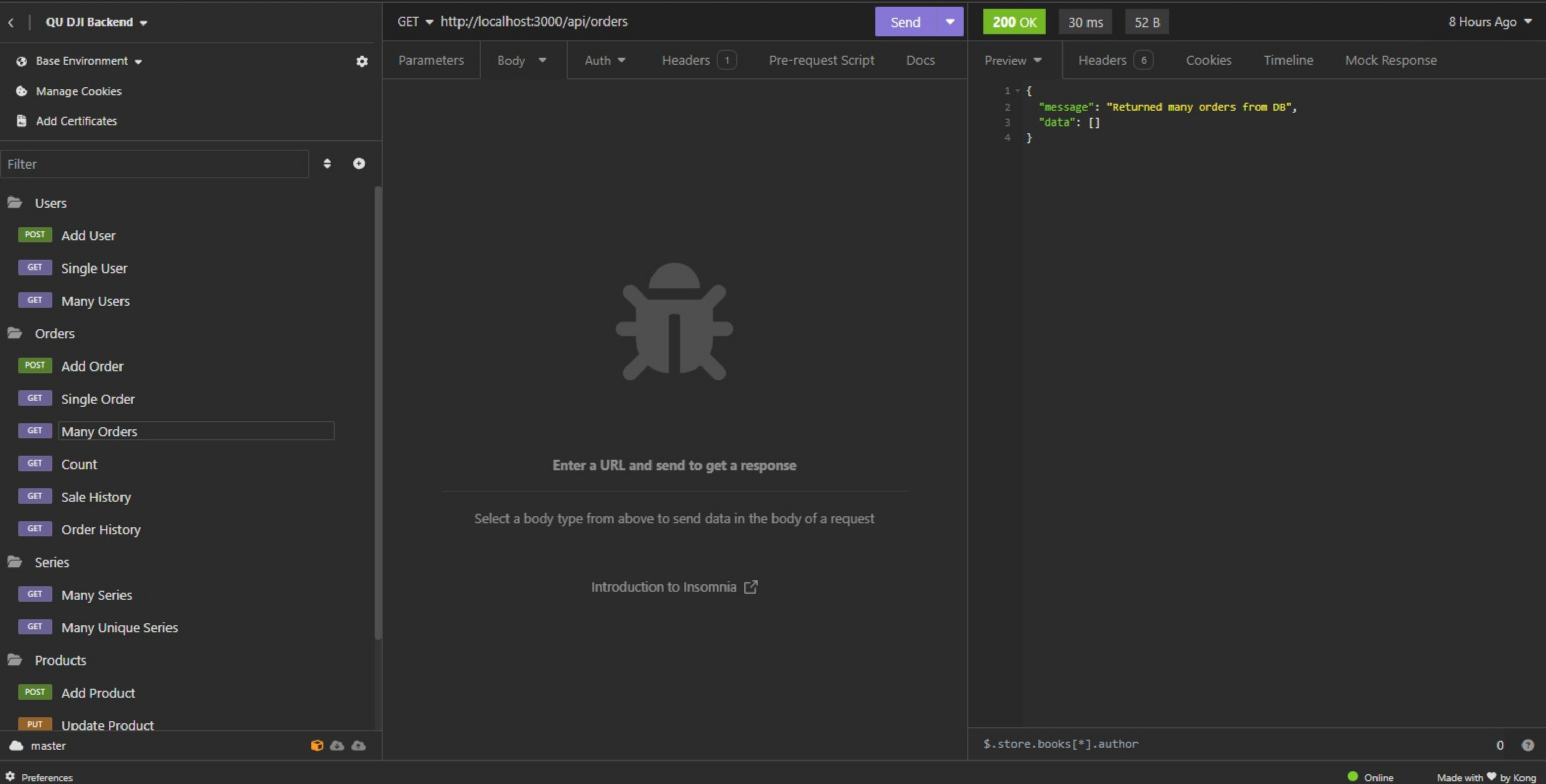
1. **Get Many Products**
   * Method: getMany
   * Description: Retrieves multiple products based on specified criteria such as series name or description.
2. **Get Unique Product**
   * Method: getUnique
   * Description: Retrieves a single product by its unique identifier (ID).
3. **Get Products Count**
   * Method: getCount
   * Description: Counts the number of products based on specified criteria such as series name or description.
4. **Add One Product**
   * Method: addOne
   * Description: Adds a new product to the database or updates an existing one if specified.
5. **Update One Product**
   * Method: updateOne
   * Description: Updates an existing product in the database based on its unique identifier (ID).
6. **Add Default Data**
   * Method: addDefaultData
   * Description: Adds default product data to the database by reading from a JSON file.
7. **Calculate Total Sales Revenue**
   * Method: calculateTotalSalesRevenue
   * Description: Calculates the total sales revenue generated from all orders.
8. **Calculate Average Order Value**
   * Method: calculateAverageOrderValue
   * Description: Calculates the average order value based on the total order value and the number of orders.
9. **Get Top Selling Products**
   * Method: getTopSellingProducts
   * Description: Retrieves a list of top selling products based on the number of sales, limited by a specified count.
10. **Calculate Total Users**
    * Method: calculateTotalUsers
    * Description: Counts the total number of users registered in the system.
11. **Calculate Average Product Rating**
    * Method: calculateAverageProductRating
    * Description: Calculates the average rating of all products based on user reviews.
12. **Get Total Number of Orders**
    * Method: getTotalNumberOfOrders
    * Description: Counts the total number of orders placed in the system.
13. **Calculate Total Sales Revenue by Date Range**
    * Method: calculateTotalSalesRevenueByDateRange
    * Description: Calculates the total sales revenue generated within a specific date range.

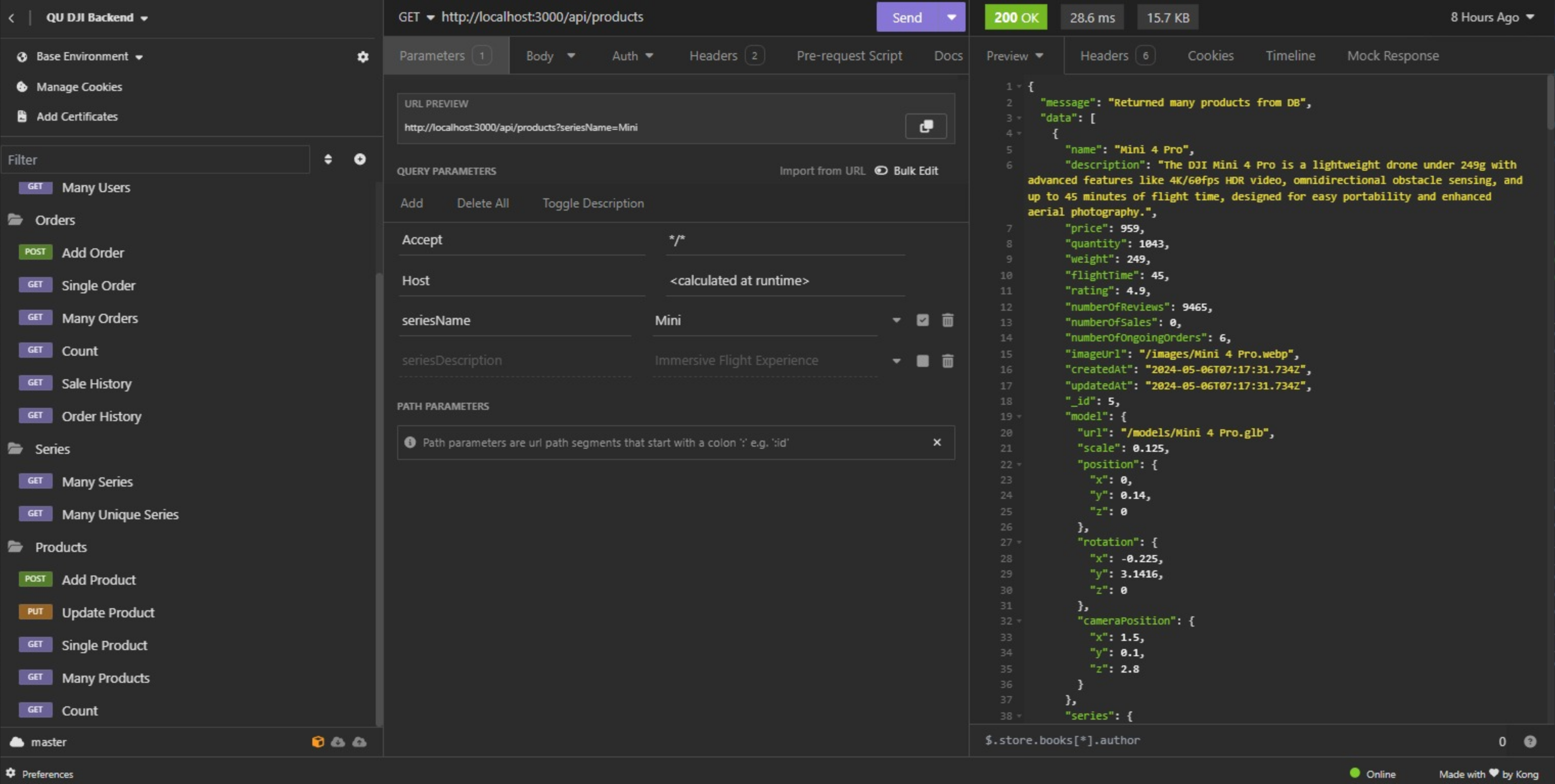
# Conducted Tests and evidence.

API testing:

A screenshot of a computer program

Description automatically generated





Analytics page testing:

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

# Discussion of the project contribution of each team member

Muhammad Fauzan: Modernized the techniques we used to create the website. Helping other members understand how to do their tasks.

Osamah Alsumaitti: Made half of the analytics page. Helped getting the products data from the API.

Marwan Khankan: Created the API and helped in backend and documentation.

Mohammed Al-ghazali: create the analytics page. Helped other members in their tasks