|  |  |  |
| --- | --- | --- |
|  | | |
|  | **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:**  **https://github.com/moumenkhha/phase2/tree/main** | |
| **Group Id:** | |  |
| **Group Members:** | | Moumen Abd Alkadr Alkhateb Alhasani (201910156)  **Emails:** ma1910156@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 |  |  |  |
| **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**.

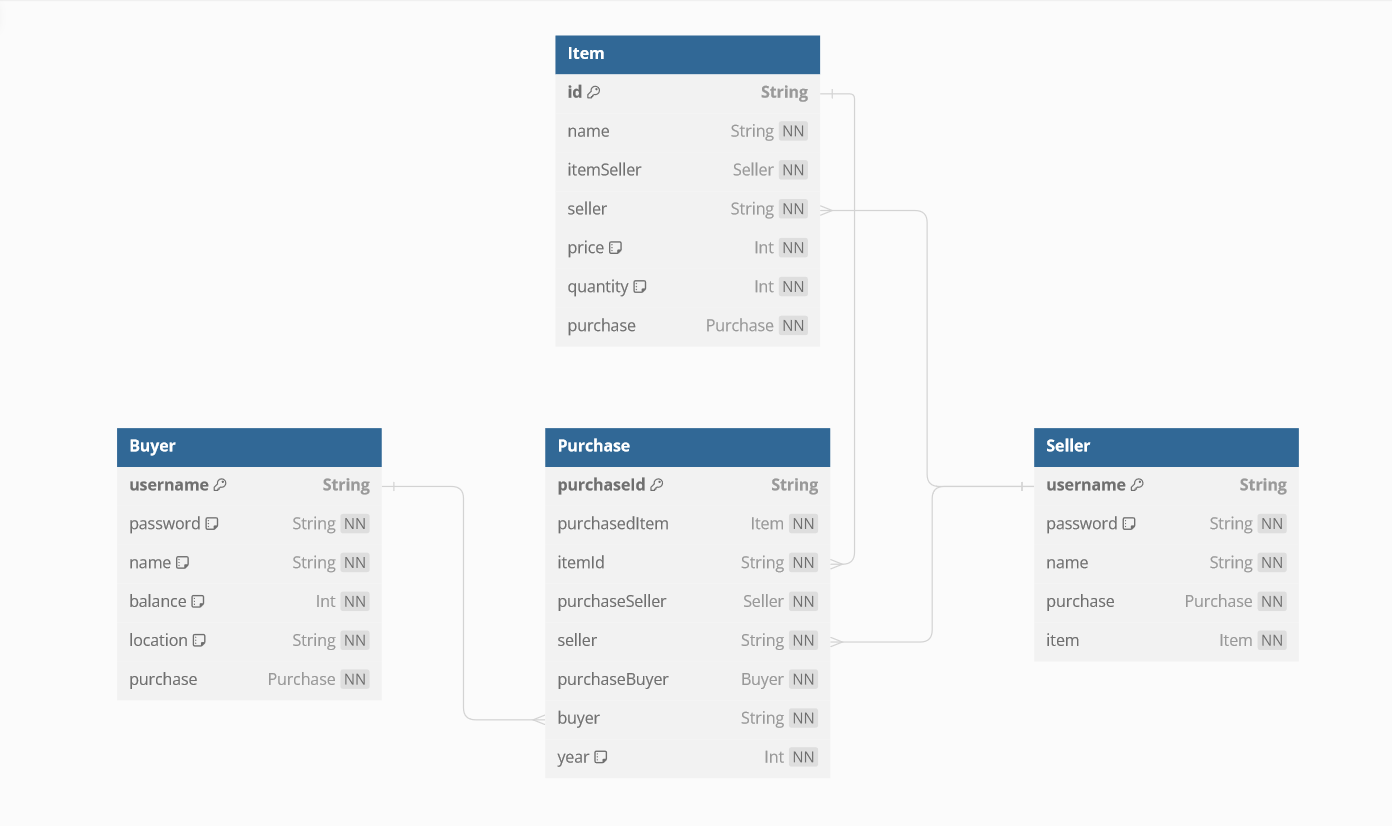
**[Grades: 85-80]:** Will be given only **to fully functional application** **with most of all the quality criteria cited above met** and the project has good design for the various functionalities. **The report is professional**.

**[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.

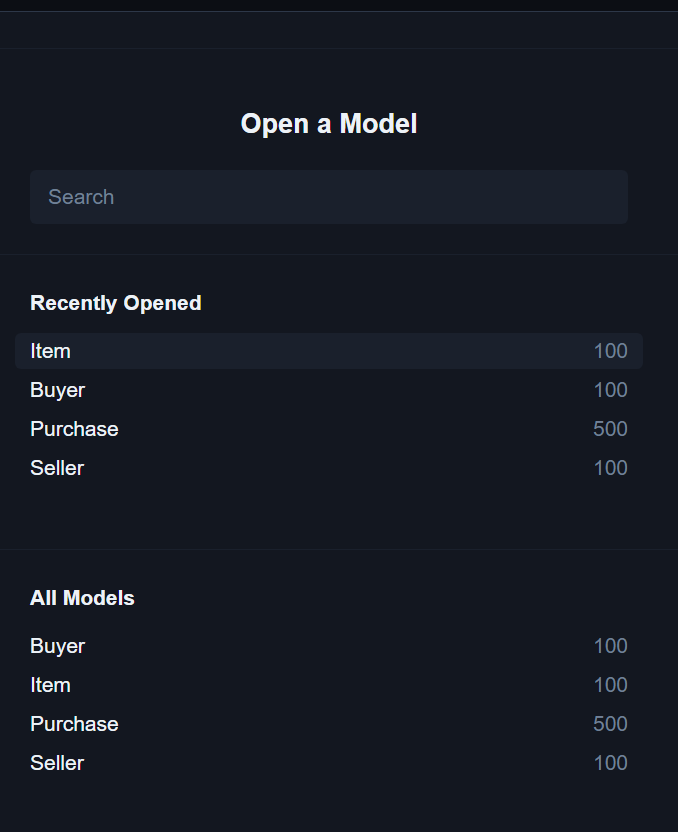
The grades are not negotiable. We expect that only a small portion (around 15%) of the class will be able to meet the criteria for the grades **[100-85]. You should work hard to and demonstrate the merits of your application to earn those grades.**

# Description of your proposed platform: It fetches various statistics related to buying and selling activities and renders them in a tabular format on a web page. It fetches the top countries buying, top items, items never purchased, sellers who never sell, top sellers between 2013-2023, and the best buyer in 2024 using the imported functions. It renders these statistics in separate tables. Top countries buying are displayed with their respective total purchases. Top sellers between 2013-2023 are displayed with their respective total sales. Top items bought are displayed with their respective total purchases. The best buyer in 2024 is displayed. Items never purchased are displayed with their details such as item ID, seller, price, and quantity. Sellers who never sell are displayed. Each table is styled with borders and captions for better readability.

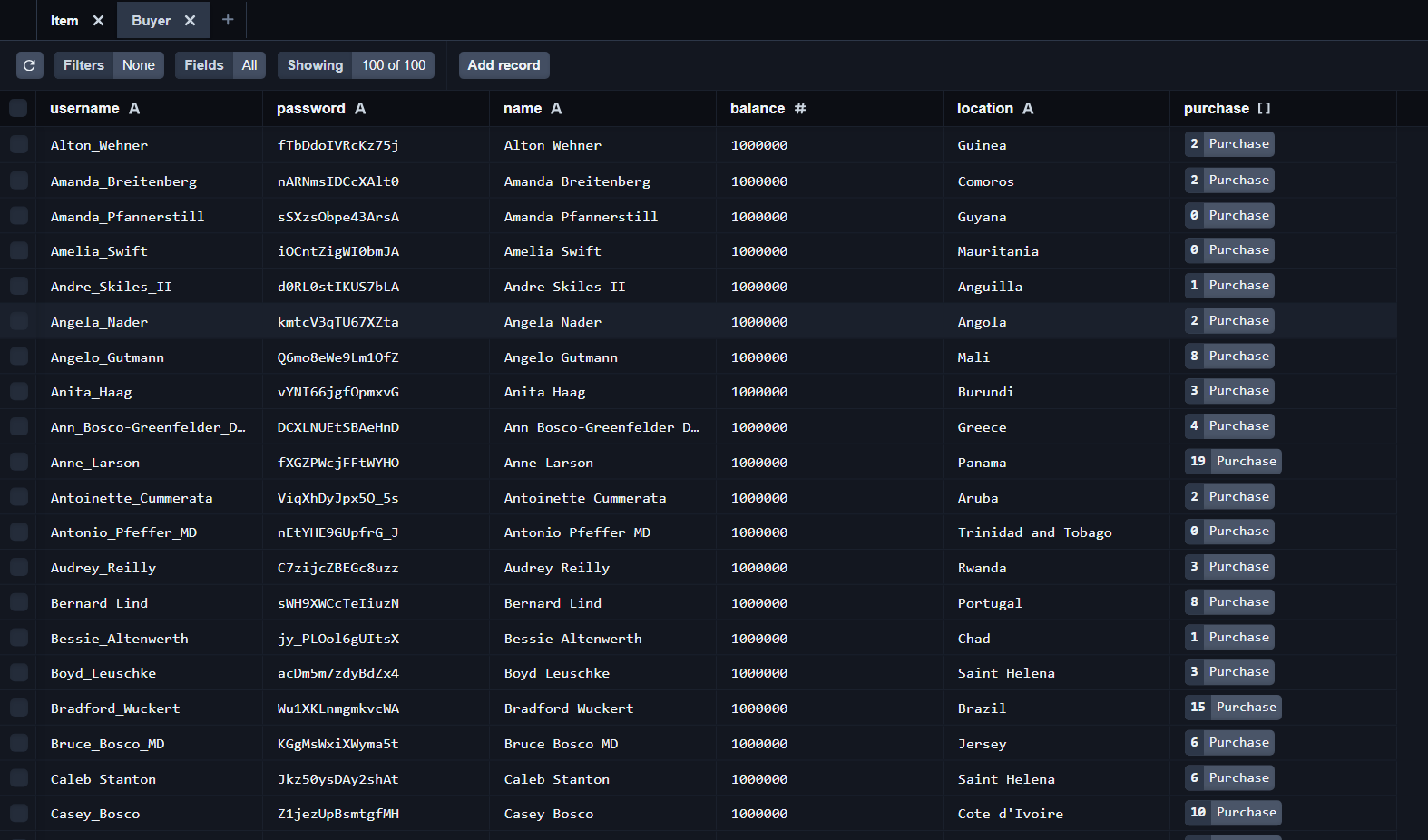
# Data Model

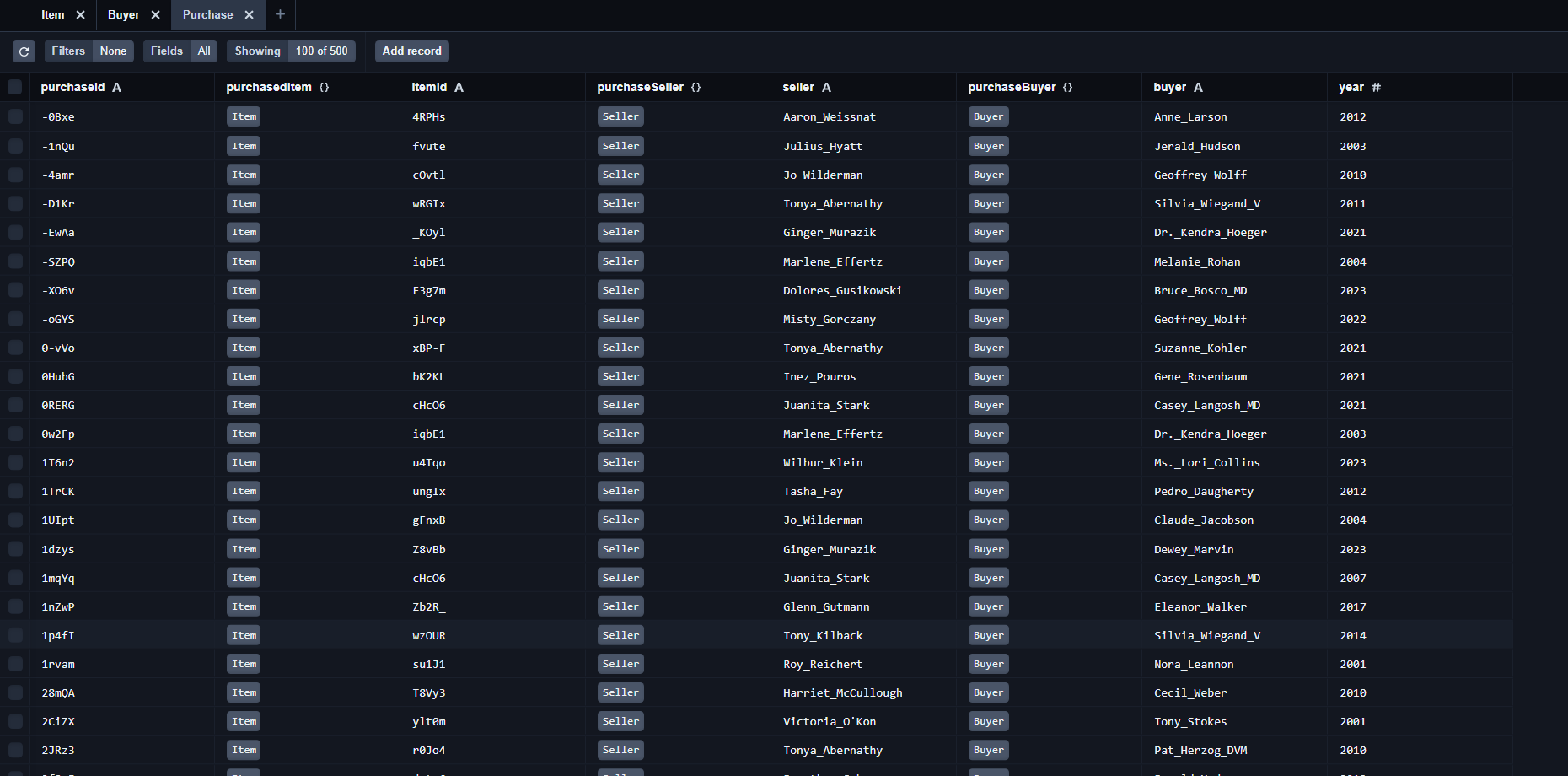


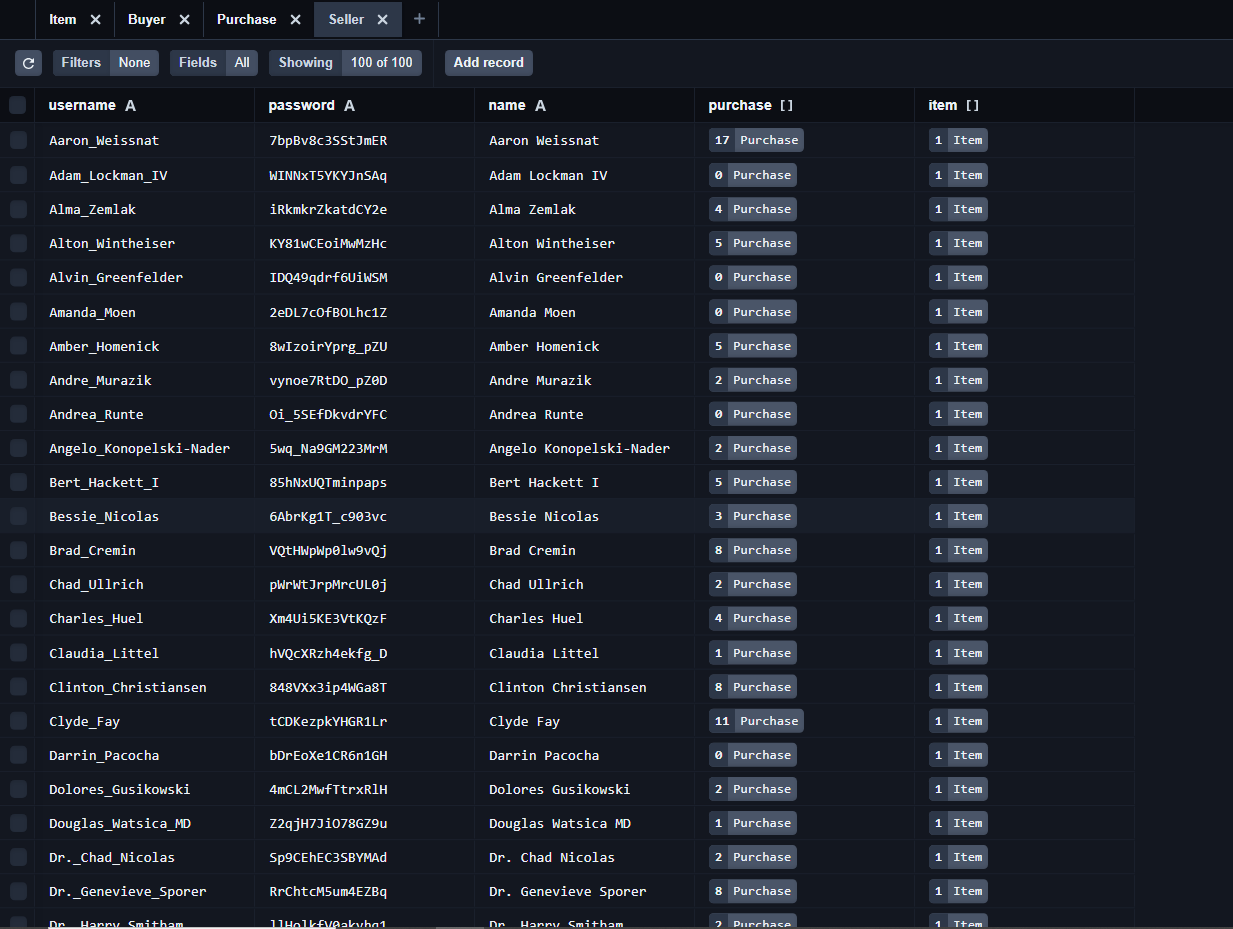
# Database Initialization











# Description of the implemented Statistics use-case

# Description

**topCountriesBuying: Fetches the top 5 countries with the highest number of purchases. It groups purchases by country location, counts the number of purchases for each location, orders the results by the count in descending order, and limits the result to the top 5 locations.**

**topItems: Fetches the top 5 items with the highest number of purchases. It groups purchases by item ID, counts the number of purchases for each item ID, orders the results by the count in descending order, and limits the result to the top 5 items.**

**topSellers\_13\_23: Fetches the top 5 sellers with the highest number of sales between the years 2013 and 2023. It groups purchases by seller, counts the number of sales for each seller within the specified years, orders the results by the count in descending order, and limits the result to the top 5 sellers.**

**ItemsNeverPurchased: Fetches items that have never been purchased. It queries the item table and filters out items that do not have any associated purchases.**

**sellersNaverSell: Fetches sellers who have never sold any items. It queries the seller table and filters out sellers who do not have any associated purchases.**

**bestBuyer\_24: Fetches the best buyer in 2024. It executes a raw SQL query using Prisma's $queryRaw method to select the buyer with the highest number of purchases in the Purchase table, ordering them in descending order, and returning the top buyer.**

# What has been implemented

* topCountriesBuying()
* topItems()
* topSellers\_13\_23()
* ItemsNeverPurchased()
* sellersNaverSell()
* bestBuyer\_24()



# What is not implemented: All six different stats are implemented.

# List of implemented database queries

SELECT location, COUNT(\*) AS total

FROM Purchase

GROUP BY location

ORDER BY total DESC

LIMIT 5;

SELECT itemId, COUNT(\*) AS total

FROM Purchase

GROUP BY itemId

ORDER BY total DESC

LIMIT 5;

SELECT seller, COUNT(\*) AS total

FROM Purchase

WHERE year >= 2013 AND year <= 2023

GROUP BY seller

ORDER BY total DESC

LIMIT 5;

SELECT id, seller, price, quantity

FROM Item

WHERE id NOT IN (

SELECT DISTINCT itemId

FROM Purchase

);

SELECT name

FROM Seller

WHERE username NOT IN (

SELECT DISTINCT seller

FROM Purchase

);

SELECT username, COUNT(\*) AS total\_purchases

FROM Purchase

WHERE year = 2024

GROUP BY username

ORDER BY total\_purchases DESC

LIMIT 1;

This is one example of DB queries by prisma

import prisma from '@/repository/prisma';

export async function get(username) { // Done

if (!username) {

return await prisma.buyer.findMany({

include: {

purchase: true

}

});

}

return await prisma.buyer.findUnique({

where: {

username,

}

});

}

export async function add(props) { // Done

return await prisma.buyer.create({

data: props

});

}

export async function remove(username) { // Done

if (await prisma.buyer.findUnique({

where: {

username,

}

}))

return await prisma.buyer.delete({

where: {

username,

}

});

}

export async function update(username, props) { // Done

return await prisma.buyer.update({

where: {

username

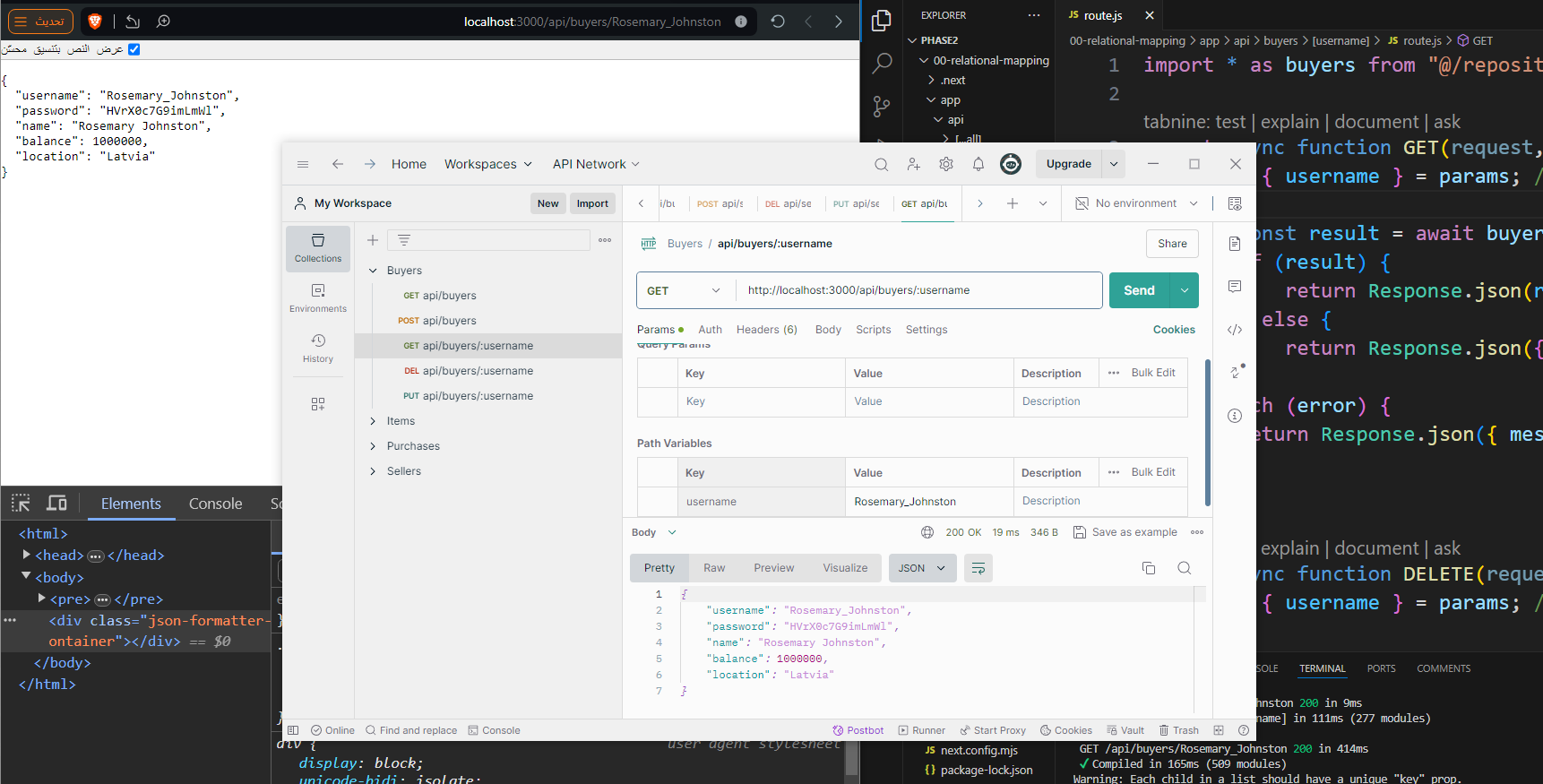
},

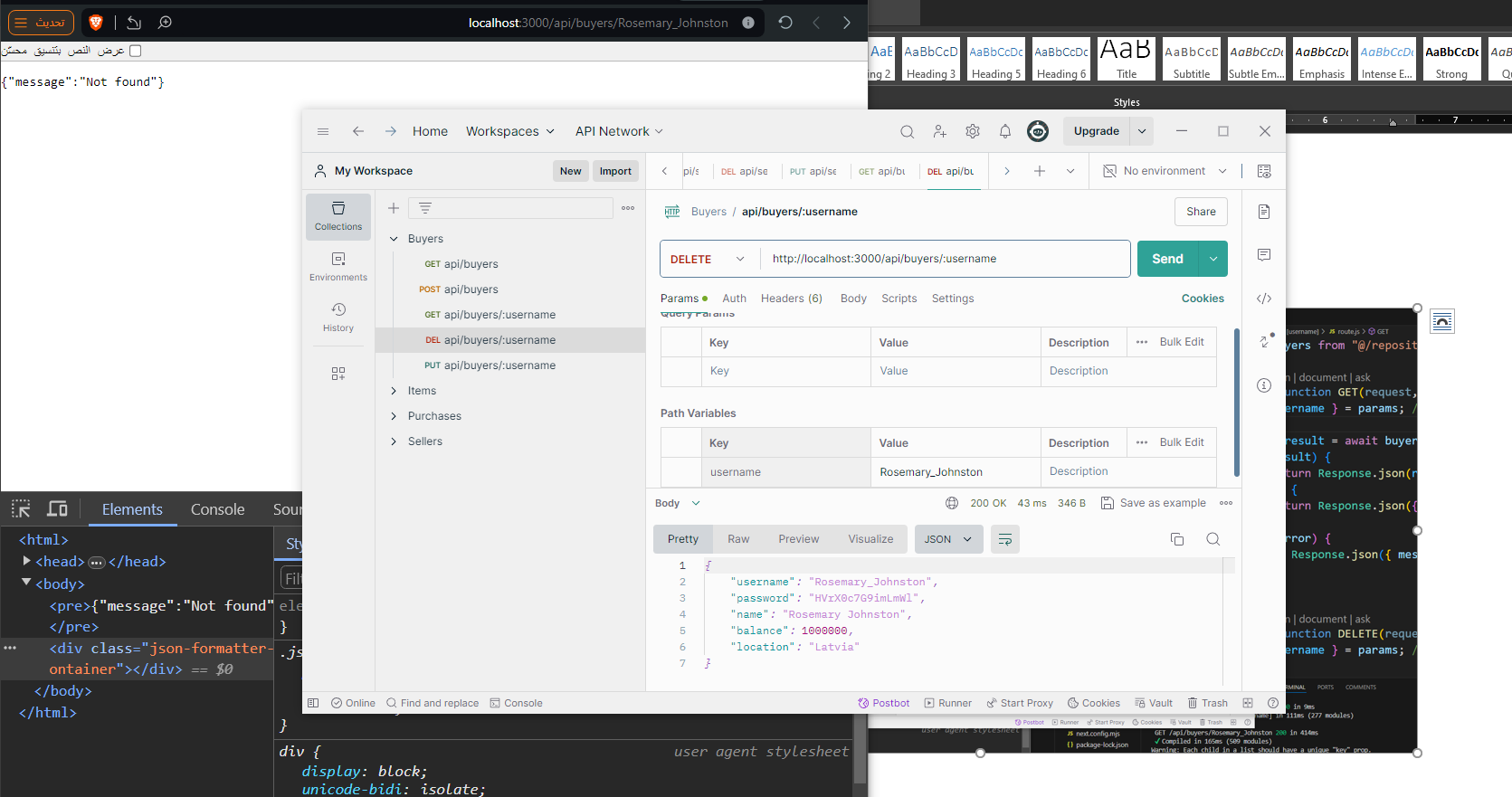
data: props

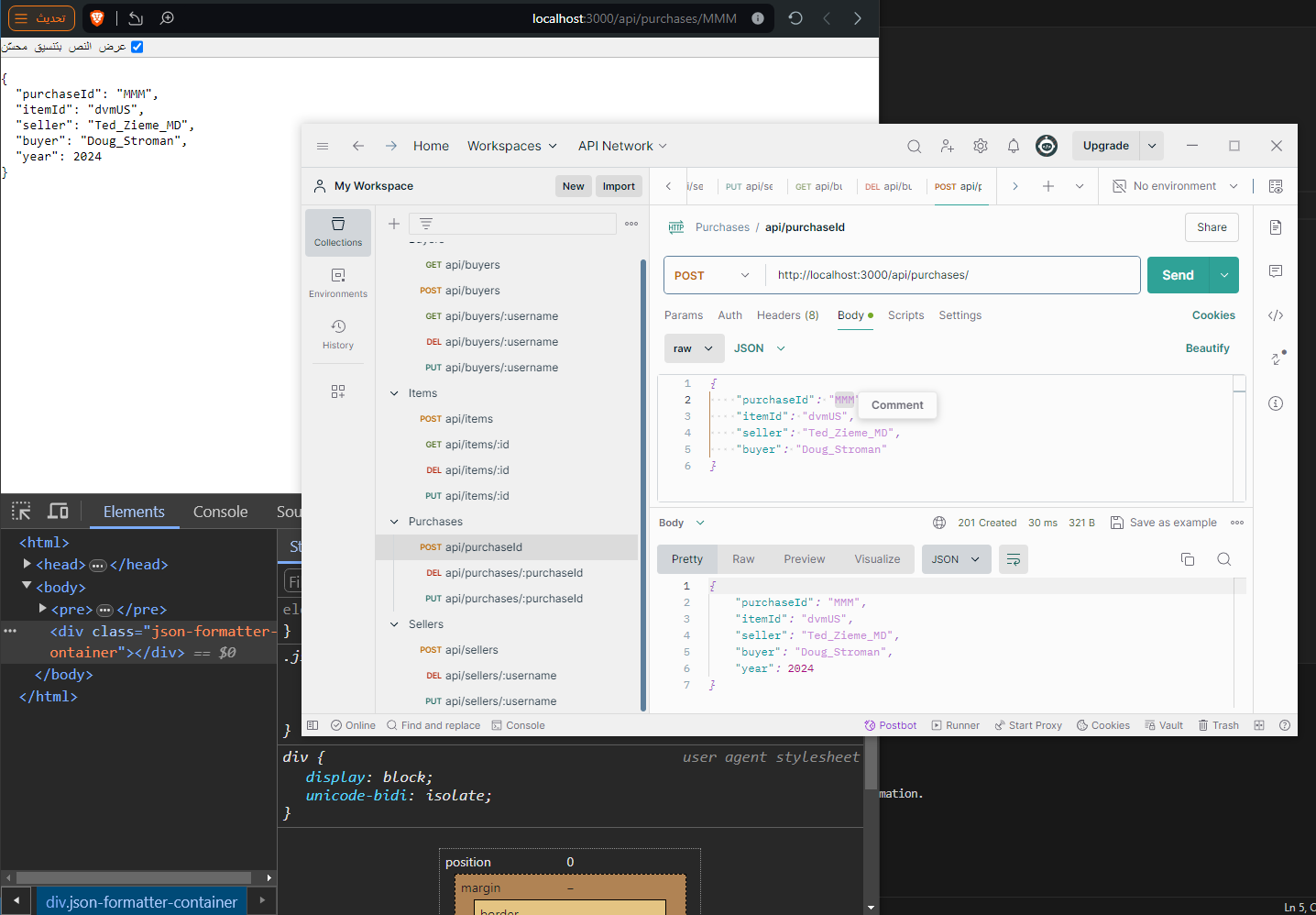
})

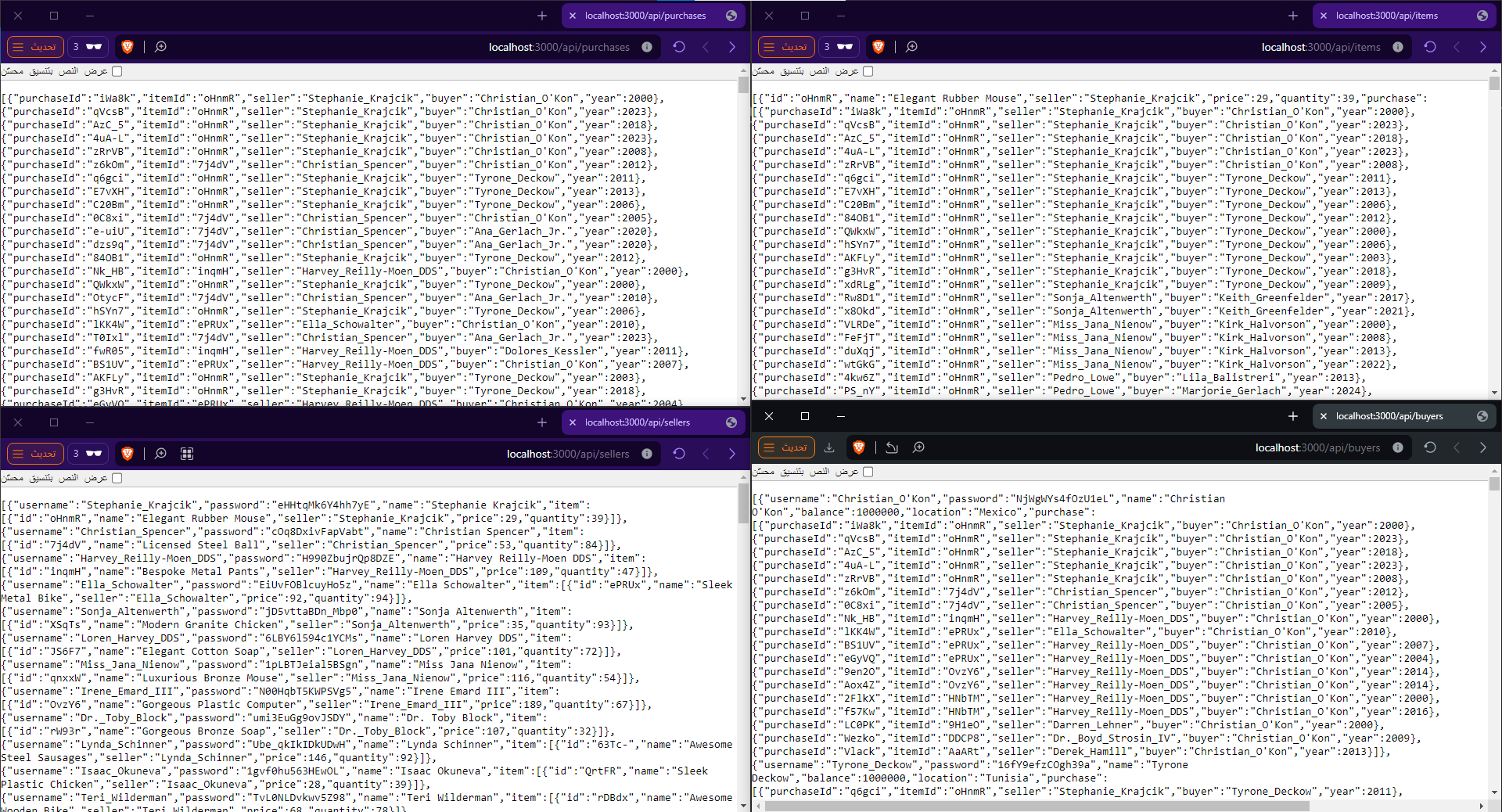
}

# Conducted Tests and evidence.









# Discussion of the project contribution of each team member