A black background with pink text

Description automatically generated

Web Development Fundamentals

Phase 2

**The Closet**

Student 1: Haya Al-Subaey QUID: 202002417

Student 2: Roudha Al-Khalaf QUID: 202002203

Student 3: Nouf Al-Kaabi QUID: 202002196

Section: L51

Date: 11 May 10, 2024

**Githhub link:** <https://github.com/Roudha-ra2002203/project.git>

* Data Model:

A screenshot of a computer

Description automatically generated

* Queries:
* **Database Queries:**
* **async getLeastSellingProducts() {**
* **try {**
* **const result = await this.client.purchase.groupBy({**
* **by: ["productId"],**
* **orderBy: {**
* **\_count: {**
* **quantity: "asc", // Sort by the count of purchases in ascending order**
* **},**
* **},**
* **take: 5,**
* **});**
* **return result;**
* **} catch (error) {**
* **throw new Error(**
* **"Error fetching least selling products: " + error.message**
* **);**
* **}**
* **}**
* **async getAveragePurchaseFrequencyPerBuyer() {**
* **try {**
* **const purchaseCountsPerBuyer = await this.client.purchase.groupBy({**
* **by: ["buyerId"],**
* **\_count: true, // Count the number of purchases per buyer**
* **});**
* **// Calculate the total number of buyers**
* **const totalBuyers = purchaseCountsPerBuyer.length;**
* **// Calculate the total number of purchases**
* **const totalPurchases = purchaseCountsPerBuyer.reduce(**
* **(acc, buyer) => acc + buyer.\_count,**
* **0**
* **);**
* **// Calculate the average purchase frequency per buyer**
* **const averagePurchaseFrequency = totalPurchases / totalBuyers;**
* **return averagePurchaseFrequency;**
* **} catch (error) {**
* **throw new Error(**
* **"Error fetching average purchase frequency per buyer: " + error.message**
* **);**
* **}**
* **}**
* **async getItemsNeverPurchased() {**
* **try {**
* **const allItems = await this.client.product.findMany();**
* **const purchasedItems = await this.client.purchase.findMany({**
* **select: {**
* **productId: true,**
* **},**
* **});**
* **const purchasedItemIds = new Set(**
* **purchasedItems.map((item) => item.itemId)**
* **);**
* **const itemsNeverPurchased = allItems.filter(**
* **(item) => !purchasedItemIds.has(item.id)**
* **);**
* **return itemsNeverPurchased;**
* **} catch (error) {**
* **throw new Error("Error fetching items never purchased: " + error.message);**
* **}**
* **}**
* **async getTopProductsBySales() {**
* **//--**
* **const result = await this.client.purchase.groupBy({**
* **by: ["productId"],**
* **orderBy: {**
* **\_sum: {**
* **quantity: "desc",**
* **},**
* **},**
* **\_sum: {**
* **quantity: true,**
* **},**
* **take: 5,**
* **});**
* **return result;**
* **}**
* **async getBuyersPerLocation() {**
* **//--**
* **try {**
* **const result = await this.client.buyer.groupBy({**
* **by: ["city", "Country"],**
* **\_count: true,**
* **});**
* **return result;**
* **} catch (error) {**
* **throw new Error("Error fetching buyers per location: " + error.message);**
* **}**
* **}**
* **async getTotalPurchasesPerSeller() {**
* **//--**
* **const result = await this.client.purchase.groupBy({**
* **by: ["sellerId"],**
* **\_count: true,**
* **});**
* **return result;**
* **}**
* **async signIn(uname, pwd) {**
* **try {**
* **let user = await this.client.buyer.findUnique({**
* **where: {**
* **username: uname,**
* **},**
* **});**
* **if (!user || user.password !== pwd) {**
* **user = await this.client.seller.findUnique({**
* **where: {**
* **username: uname,**
* **},**
* **});**
* **if (!user || user.password !== pwd) {**
* **throw new Error("Invalid username or password");**
* **}**
* **}**
* **return user;**
* **} catch (error) {**
* **throw new Error("Error during sign-in: " + error.message);**
* **}**
* **}**
* **async addItem(name, price, qty, sellerId, img, desc) {**
* **let highestItemId = await this.getHighestItemId();**
* **console.log(highestItemId)**
* **let id = (parseInt(highestItemId)+1).toString()**
* **console.log("id is",id)**
* **try {**
* **const newItem = await this.client.product.create({**
* **data: {**
* **id,**
* **name,**
* **price,**
* **quantity: qty,**
* **seller: { connect: { id: sellerId } },**
* **imageUrl: img,**
* **description: desc,**
* **},**
* **});**
* **return newItem;**
* **} catch (error) {**
* **throw new Error("Error adding item: " + error.message);**
* **}**
* **}**
* **async getItems() {**
* **try {**
* **const items = await this.client.product.findMany();**
* **return items;**
* **} catch (error) {**
* **throw new Error("Error fetching items: " + error.message);**
* **}**
* **}**
* **async getTransactions() {**
* **try {**
* **const transactions = await this.client.purchase.findMany();**
* **return transactions;**
* **} catch (error) {**
* **throw new Error("Error fetching transactions: " + error.message);**
* **}**
* **}**
* **async purchase(productId, buyerId, qty) {**
* **console.log(productId, buyerId, qty)**
* **try {**
* **const product = await this.client.product.findUnique({**
* **where: {**
* **id: productId,**
* **},**
* **});**
* **console.log("product found")**
* **if (!product) {**
* **throw new Error("Product not found");**
* **}**
* **const totalPrice = product.price \* qty;**
* **const buyer = await this.client.buyer.findUnique({**
* **where: {**
* **id: buyerId,**
* **},**
* **});**
* **console.log("buyer Found")**
* **if (!buyer || buyer.balance < totalPrice) {**
* **throw new Error("Insufficient balance");**
* **}**
* **console.log("Not Insufficient balance")**
* **const transaction = await this.client.purchase.create({**
* **data: {**
* **productId,**
* **sellerId:product.sellerId ,**
* **buyerId,**
* **quantity: qty,**
* **totalPrice,**
* **},**
* **});**
* **await this.client.buyer.update({**
* **where: {**
* **id: buyerId,**
* **},**
* **data: {**
* **balance: {**
* **decrement: totalPrice,**
* **},**
* **},**
* **});**
* **return transaction;**
* **} catch (error) {**
* **throw new Error("Error during purchase: " + error.message);**
* **}**
* **}**
* **async getSellerInfo(id) {**
* **try {**
* **const seller = await this.client.seller.findUnique({**
* **where: {**
* **id,**
* **},**
* **});**
* **if (!seller) {**
* **throw new Error("Seller not found");**
* **}**
* **return seller;**
* **} catch (error) {**
* **throw new Error("Error fetching seller info: " + error.message);**
* **}**
* **}**
* **async getUserInfo(id) {**
* **try {**
* **const user = await this.client.buyer.findUnique({**
* **where: {**
* **id,**
* **},**
* **});**
* **if (!user) {**
* **throw new Error("User not found");**
* **}**
* **return user;**
* **} catch (error) {**
* **throw new Error("Error fetching user info: " + error.message);**
* **}**
* **}**
* **async close() {**
* **await this.client.$disconnect();**
* **}**
* **}**
* **module.exports = DataHandler;**
* **Test Cases:**
* **Test Case: Sign in with valid credentials**
  + Objective: Verify that a user can sign in with valid credentials and is redirected to the appropriate main page based on their user type (seller or buyer).
  + Preconditions:

1. The application is accessible, and the sign-in page is displayed.
2. Valid test user account exists in the system with defined roles (seller or buyer).
   * Test steps:
3. Navigate to the application’s sign-in page

A screenshot of a login form

Description automatically generated

1. Enter valid credentials (username/email and password) of a registered user.
   * + Username: [valid username]
     + Password: [valid password]

A screenshot of a login form

Description automatically generated

1. Click on the “Sign In” button
   * Expected behavior:
     + After successful sign-in, verify that the user is redirected to the seller's main page.
     + Confirm that the URL or displayed content indicates the seller's interface.

A screenshot of a computer

Description automatically generated

* If the user is a buyer:
  + After successful sign-in, verify that the user is redirected to the buyer's homepage or main page.
  + Confirm that the URL or displayed content reflects the buyer's dashboard.

A screenshot of a website

Description automatically generated

* **Test Case: Search Items by Name or Type**
  + Objective: Validate the functionality of searching items by name or type (category) on the e-commerce platform.
  + Preconditions:

1. The application is accessible and the user is logged in.
2. The search functionality is available and visible on the homepage or designated search page.
   * Test Steps:

**Scenario 1: Search by Item Nam**

1. Navigate to the homepage or designated search page of the e-commerce platform.
2. Locate the search bar/input field
3. Enter a specific item name in the search field.
4. Click on the search button/icon or press "Enter" on the keyboard.

A screenshot of a computer

Description automatically generated

Expected Behavior:

* The search results page should display items matching the entered item name.
* Verify that the displayed items' names contain the searched keyword ("Running Shoes").
* **Test Case: Seller Posts Item for Sale**
  + Objective: Verify that a seller user can successfully post an item for sale with specified details (quantity, price, category) on the e-commerce platform.
  + Preconditions:

1. The seller user is logged in to their account on the e-commerce platform.
2. The seller has access permissions to add and manage items for sale.
   * Test Steps:
3. Navigate to the "Sell" or "Add Item" page on the e-commerce platform.
4. Enter the following details for the item:

* **Item Name:** [Enter a unique item name]
* **Quantity Available:** [Enter a valid quantity available for sale]
* **Price per Unit:** [Enter a valid price for the item]
* **Category:** Select the appropriate category for the item

A screenshot of a computer

Description automatically generated

3. Click on the "Submit" or "Post Item" button to add the item for sale.

* + Expected Behavior:
* The recently added item should be listed among the seller's and buyer availabel items for sale.
* The item details (name, quantity, price, category) should match the information entered during item posting.

A screenshot of a computer

Description automatically generated

A screenshot of a phone

Description automatically generated

* **Test Case: Buying an Item and Completing Checkout**
  + Objective: Validate the process of buying an item from the e-commerce platform by selecting an item, specifying quantity, adding to cart, entering shipping details, and completing the checkout process.
  + Preconditions:

1. The user is logged in to their account on the e-commerce platform.
2. The user has navigated to the item listing page or search results page.
   * Expected Behavior:

The item details (name, price, image) should be displayed accurately.

A screenshot of a phone

Description automatically generated

* + Expected Behavior:.
* The user should be able to input and confirm the quantity for purchase.

**-**Step 3: Enter Shipping Details

5. On the checkout page, enter the required shipping details:

* **Shipping Address:** [Enter a valid shipping address]

Expected Behavior:

* The shipping form should accept and validate the entered details (address, contact information).

A screenshot of a computer

Description automatically generated

**Step 4: Complete Order**

7. Click on the "Confirm Purchase" button to finalize the order.

Expected Behavior:

* The system should process the order successfully and provide an order confirmation.
* **Statistics:**

A screenshot of a computer

Description automatically generated