Term Project Report 3

Amazon Marketplace

Team members: Cash Hollister, Shiqiang Mo

Table of Contents

Introduction

This report contains the information specified in the report-3.pdf. It will cover relevant sql queries made to extract the desired data from the database. It will cover how those sql queries are utilized by Flask(python) to transform the extracted data into usable information for the Amazon Marketplace application. The report will display images of the different user interfaces, how the users interact with them, and how the sql queries allow the interface to function as designed. Included is a GitHub repository with the functional application that makes use of MySql, Flask, and Next.js(javascript framework). At the end of the report the sql commands and Flask endpoints files will be included for your reference. All code can also be found in the source code repository on GitHub.

Repository

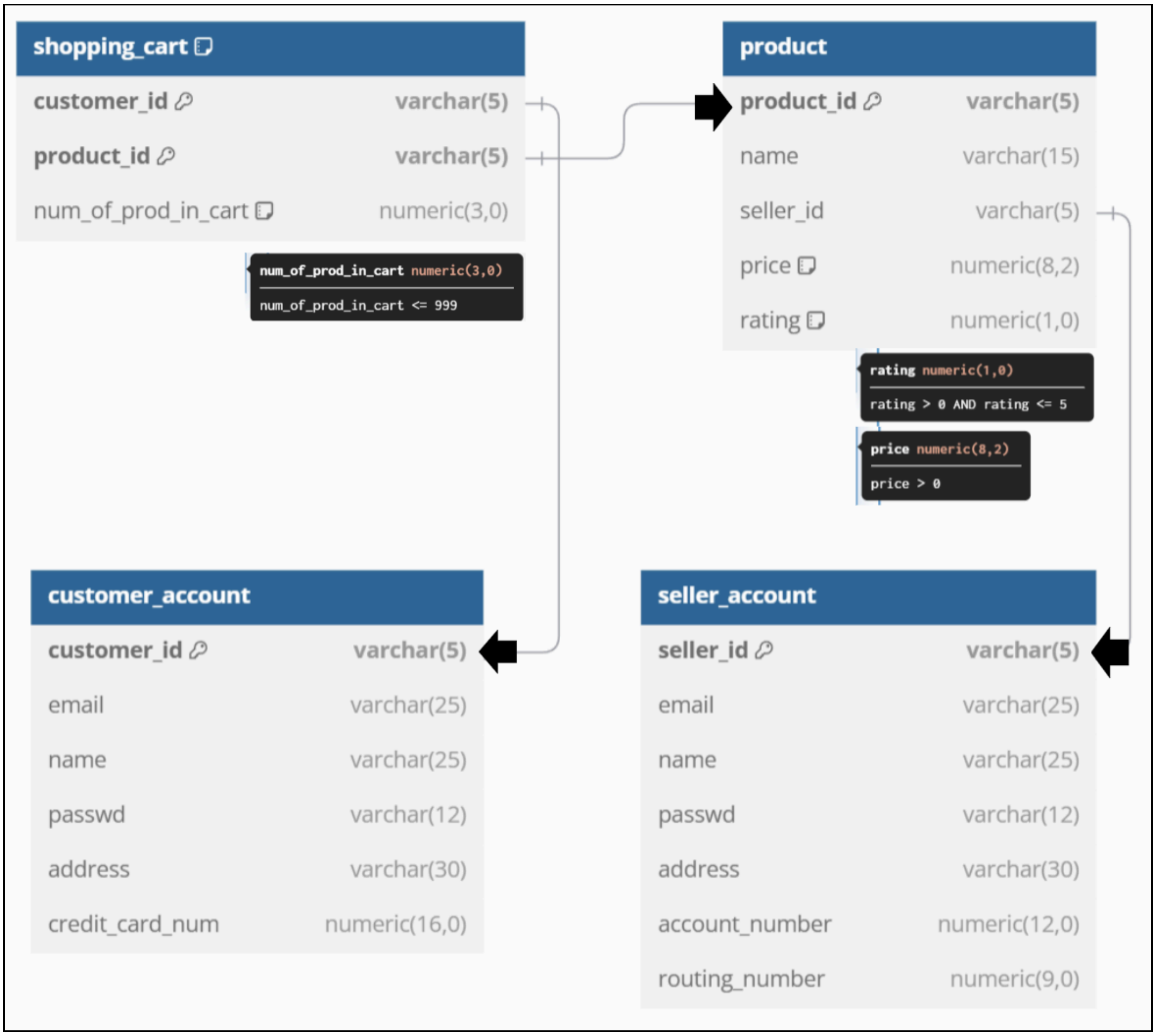
https://github.com/cashhollister2u/Report3.git

Detailed instructions are included in the project's README.md on how to download and start up the application. Application requirements and dependencies are referenced and can easily be installed following the prescribed methods in the README.md.

Part 0

<u>Schema Diagram for database:</u>

- The same tables as used in Report 2



Part 1

1.1)

Category I: More than one table in FROM

SELECT customer_id, product_id, num_of_prod_in_cart
FROM customer_account
NATURAL JOIN shopping_cart
WHERE customer_id = :cust_id;
Host Variable: :cust_id

Category II: Use SET operation

UPDATE product
SET price = :new_price
WHERE product_id = :prod_id;
Host Variables: :new_price, :prod_id

Category III: Use aggregate function and/or GROUP BY
note: This query is slightly changed from "Report 2". It includes the "WHERE" clause to specify a specific customer_id(Host Variable). This was not included in "Report 2". However, it is implemented in the live version of the application.

SELECT customer_id, SUM(price * num_of_prod_in_cart) AS total_cost
FROM shopping_cart NATURAL JOIN product
WHERE customer_id = :cust_id
GROUP BY customer_id;
Host Variable: :cust_id

Category IV: Use SUBQUERY
note: This query is slightly changed from "Report 2". It adds to the existing "WHERE" clause to specify a specific customer_id(Host Variable) and assigns "C" to the customer_account table. This was not included in "Report 2". However, it is implemented in the live version of the application.

SELECT name
FROM customer_account AS C
WHERE C.customer_id = :cust_id AND IN (

SELECT customer_id FROM shopping_cart);

Host Variable: :cust_id

Category V: Use EXISTS or UNIQUE

note: This query is slightly changed from "Report 2". It adds to the existing "WHERE" clause to specify a specific customer_id(Host Variable). This was not included in "Report 2". However, it is implemented in the live version of the application.

SELECT C.customer_id
FROM customer_account AS C
WHERE C.customer_id = :cust_id AND EXISTS(

SELECT S.customer_id
FROM shopping_cart AS S
WHERE S.customer_id = C.customer_id GROUP BY S.customer_id
HAVING COUNT(product_id) > 1);

Host Variable: :cust_id

1.2)

```
import mysql.connector
# Establish the database connection
cnx = mysql.connector.connect(
    user='root',        # Replace with your MySQL username
    password='Msq070489',   # Replace with your MySQL password
    host='localhost',          # Replace with your MySQL server address
    database='amazon_marketplace'    # Replace with your database name
)
# Create a cursor object to interact with the database
cursor = cnx.cursor()
try:
    # Query 1: Select customer information and their shopping cart details
    query1 = (
        "SELECT customer_id, product_id, num_of_prod_in_cart "
        "FROM customer_account NATURAL JOIN shopping_cart "
        "WHERE customer_id = %s"
    )
    customer_id = '00001'
    cursor.execute(query1, (customer_id,))
    results = cursor.fetchall()
    print("Query 1 Results:")
    if not results:
        print("No data found for customer_id '00002'.")
    else:
        for (customer_id, product_id, num_of_prod_in_cart) in results:
            print(f"Customer ID: {customer_id}, Product ID: {product_id}, Number of Products: {num_of_prod_in_cart}")
    # Query 2: Update the price of a specific product
    query2 = ("UPDATE product SET price = %s WHERE product_id = %s")
    product_id = '00001'
    new_price = 10.99
    cursor.execute(query2, (new_price, product_id))
    cnx.commit()  # Commit changes for the update query
    print("\nQuery 2: Product price updated successfully.")
    # Query 3: Calculate total cost for a specified customer based on their cart contents
    query3 = (
            "SELECT customer_id, SUM(price * num_of_prod_in_cart) AS total_cost "
            "FROM shopping_cart NATURAL JOIN product "
            "WHERE customer_id = %s "
            "GROUP BY customer_id "
        )
    cursor.execute(query3, (customer_id,))
    print("\nQuery 3 Results:")
    for (customer_id, total_cost) in cursor:
        print(f"Customer ID: {customer_id}, Total Cost: {total_cost}")
    # Query 4: Select the customer name if the customer has products in their cart
    query4 = (
            "SELECT name "
            "FROM customer_account AS C "
            f"WHERE C.customer_id = {customer_id} AND customer_id IN ( "
            "SELECT customer_id "
            "FROM shopping_cart) "
        )
    cursor.execute(query4)
    print("\nQuery 4 Results:")
    print("Customer with Items in their shopping cart")
    for (name,) in cursor:
        print(f"Customer Name: {name}")
    # Query 5: Select the customer id if the customer has more than one different product in their shopping cart
    query5 = (
            "SELECT C.customer_id "
            "FROM customer_account AS C "
            f"WHERE C.customer_id = {customer_id} AND EXISTS( "
            "SELECT S.customer_id "
            "FROM shopping_cart AS S "
            "WHERE S.customer_id = C.customer_id "
            "GROUP BY S.customer_id "
            "HAVING COUNT(product_id) > 1) "
        )
    cursor.execute(query5)
    print("\nQuery 5 Results:")
    result = cursor.fetchall()
    print(f"Customer id with more than one different products in their shopping cart: ")
    print(result)

except mysql.connector.Error as err:
    print(f"Error: {err}")
finally:
    # Closing the cursor and database connection
    cursor.close()
    cnx.close()
```

1.3)

Terminal Logs

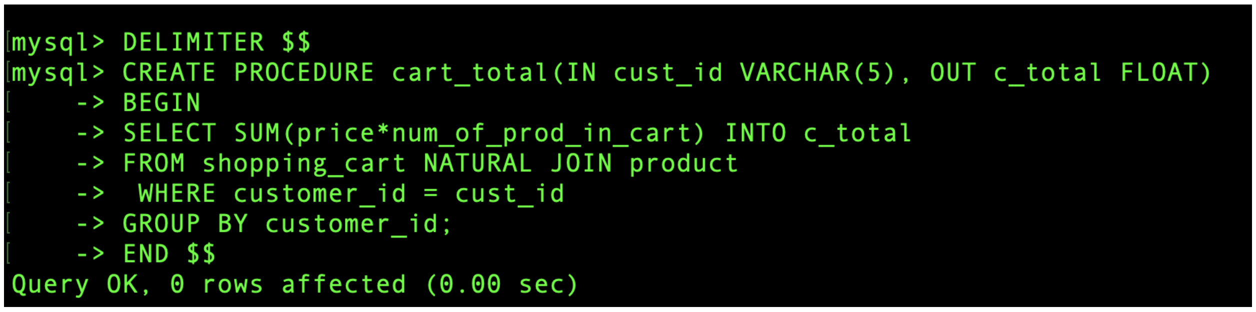
```
Query 1 Results:
Customer ID: 00001, Product ID: 00001, Number of Products: 2
Customer ID: 00001, Product ID: 00002, Number of Products: 2

Query 2: Product price updated successfully.

Query 3 Results:
Customer ID: 00001, Total Cost: 47.96

Query 4 Results:
Customer with Items in their shopping cart
Customer Name: me

Query 5 Results:
Customer id with more than one different products in their shopping cart:
[('00001',)]
```

Part 2

2.1)

Procedure:

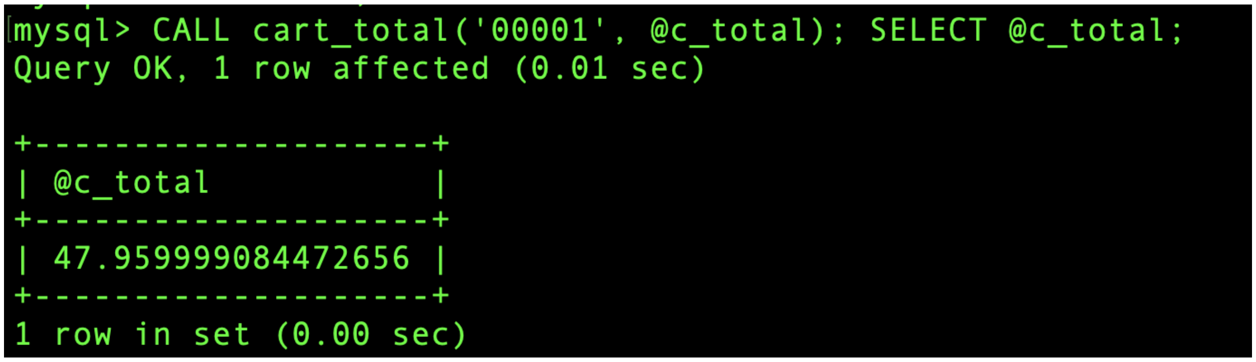
```
DELIMITER $$
CREATE PROCEDURE cart_total(IN cust_id VARCHAR(5), OUT c_total FLOAT)
  BEGIN
    SELECT SUM(price*num_of_prod_in_cart) INTO c_total
    FROM shopping_cart NATURAL JOIN product
    WHERE customer_id = cust_id
    GROUP BY customer_id;
  END $$
Query OK, 0 rows affected (0.00 sec)
DELIMITER ;
```


```
mysql> DELIMITER $$
mysql> CREATE PROCEDURE cart_total(IN cust_id VARCHAR(5), OUT c_total FLOAT)
    -> BEGIN
    -> SELECT SUM(price*num_of_prod_in_cart) INTO c_total
    -> FROM shopping_cart NATURAL JOIN product
    ->  WHERE customer_id = cust_id
    -> GROUP BY customer_id;
    -> END $$
Query OK, 0 rows affected (0.00 sec)
```

Call Procedure:

CALL cart_total('00001', @c_total); SELECT @c_total;

Query OK, 1 row affected (0.01 sec)


```
mysql> CALL cart_total('00001', @c_total); SELECT @c_total;
Query OK, 1 row affected (0.01 sec)

+--------------------+
| @c_total           |
+--------------------+
| 47.959999084472656 |
+--------------------+
1 row in set (0.00 sec)
```

Description:

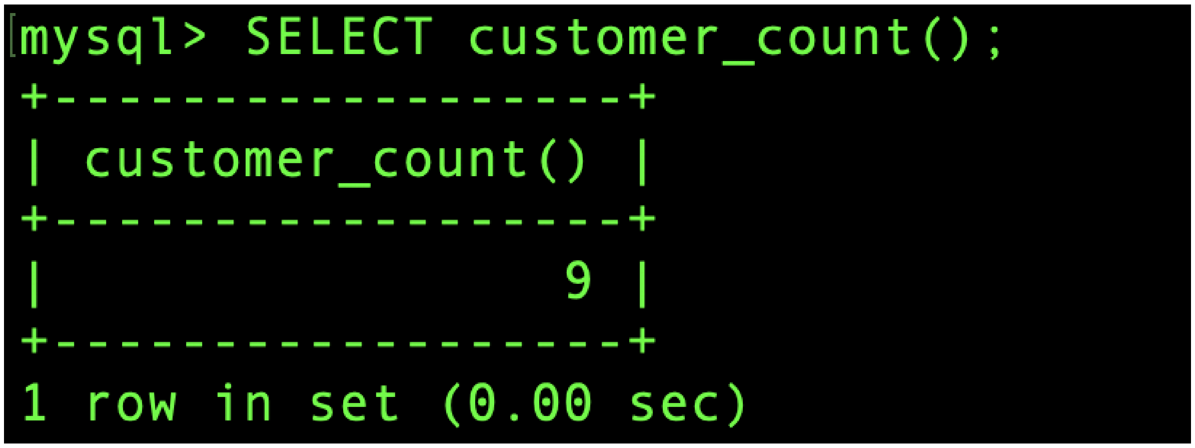
This procedure is utilized to get the total cost of the shopping cart associated with a customer account id.

2.2)

Function:

```
DELIMITER $$
CREATE FUNCTION customer_count()
RETURNS INTEGER
BEGIN
   DECLARE c_count INTEGER;
   SELECT COUNT(*) INTO c_count
   FROM customer_account;
   RETURN c_count;
END $$
Query OK, 0 rows affected (0.00 sec)
```

Call Function:
SELECT customer_count();


```
mysql> SELECT customer_count();
+-------------------+
| customer_count() |
+-------------------+
|                 9 |
+-------------------+
1 row in set (0.00 sec)
```

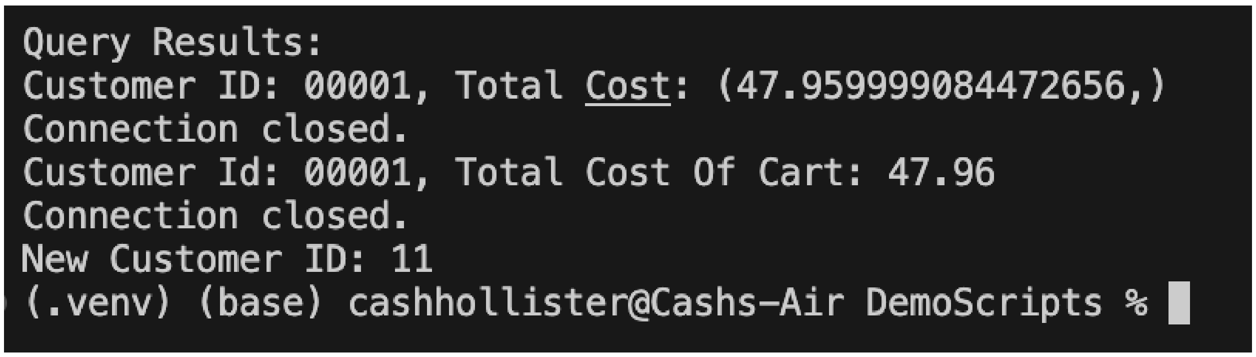
Description:
This function is utilized to get the current number of customers with registered accounts. This function aids in the assignment of the customer_id attribute to new customers on registration.

2.3)

Python Script:
```
import mysql.connector
# Establish the database connection
cnx = mysql.connector.connect(
    user='root',       # Replace with your MySQL username
    password='Msq070489',   # Replace with your MySQL password
    host='localhost',          # Replace with your MySQL server address
    database='amazon_marketplace'     # Replace with your database name
)
# Query 2: Calculate total shopping cart cost for each customer based on their cart contents
def getShoppingCartTotal(customer_id):
    # Create a cursor object to interact with the database
    cursor = cnx.cursor()

    try:
        with cursor:
            # query to execute the procedure
            sub_query = (
                "CALL cart_total(%s, @c_total) "
            )
            cursor.execute(sub_query, (customer_id,))
            #query to select the returned value of the procedure
            query = (
                "SELECT @c_total "
            )
            cursor.execute(query)
            results = cursor.fetchall()
            print(results)
            if not results:
                print(f"No data found for customer_id {customer_id}.")
            else:
                print("\nQuery Results:")
                for (total_cost) in results:
                    print(f"Customer ID: {customer_id}, Total Cost: {total_cost}")
                    # rounded the value in the tuple and reassigned to new tuple => parent expects tuple
object
                    rounded_total_cost = (round(total_cost[0], 2),)
                    return rounded_total_cost

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        cursor.close() # close connection
        print("Connection closed.")

# Query 11 get count of rows in customer_account table
# this is used to assign customer_id on registration
def getCustomerBaseCount():
    # Create a cursor object to interact with the database
    cursor = cnx.cursor()
    try:
        with cursor:
            query = (
                "select customer_count() "
            )
            cursor.execute(query)
            result = cursor.fetchone()

            # return count of rows or number of customer accounts
            return result[0]
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        cursor.close() # close connection
        print("Connection closed.")

customer_1 = '00001'
customer_1_cart_total =getShoppingCartTotal(customer_id=customer_1)
print(f"Customer Id: {customer_1}, Total Cost Of Cart: {customer_1_cart_total[0]} ")

new_customer_id = getCustomerBaseCount() + 1
print(f"New Customer ID: {new_customer_id}")

cnx.close() # close cnx
```

<u>Output:</u>

```
Query Results:
Customer ID: 00001, Total Cost: (47.959999084472656,)
Connection closed.
Customer Id: 00001, Total Cost Of Cart: 47.96
Connection closed.
New Customer ID: 11
```



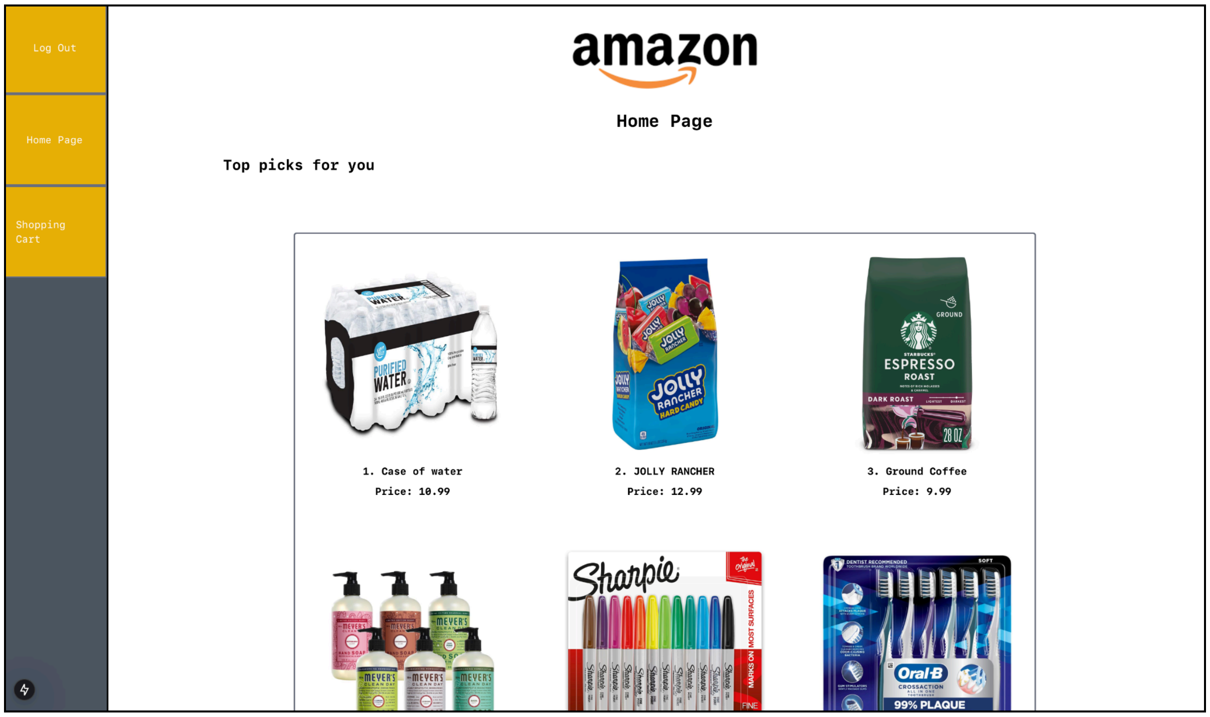
```
Query Results:
Customer ID: 00001, Total Cost: (47.959999084472656,)
Connection closed.
Customer Id: 00001, Total Cost Of Cart: 47.96
Connection closed.
New Customer ID: 11
(.venv) (base) cashhollister@Cashs-Air DemoScripts %
```

Part 3

Implemented Tasks:

3.1)

Amazon Home Page



1. <u>Description</u>

   This is the “Amazon Home Page”. It displays all the stored products from the **product** table in the database. The python program runs a simple query that selects all of the items in the **product** table. It allows the user to view images, price, and name of each of the items listed for sale. The home page also allows the user to route to different locations on the website by either clicking one of the buttons in the taskbar(left side of screen) or one of the images associated with a product.

2.

Python Code:

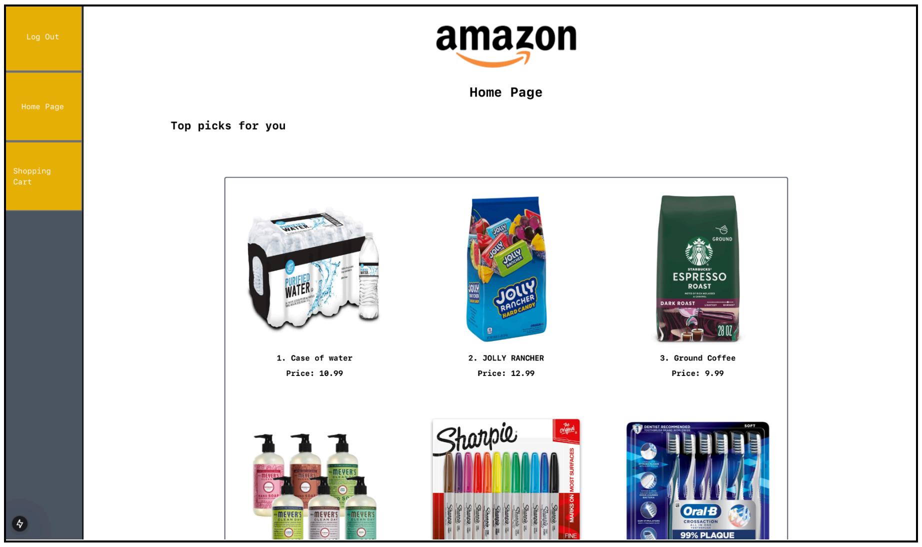
Endpoints(Flask)

```
# handle retrieving Home Page products
@user_bp.route('/products', methods=['POST'])
def products():
    # sql query to retrieve all products on website
    products = getAllProducts()
    return jsonify({"products":products}), 201
```

SQL Commands (MySql)

```
# Query 8 get all products on the website
def getAllProducts():
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query8 = (
                "SELECT * "
                "FROM PRODUCT"
            )
            cursor.execute(query8)
            results = cursor.fetchall()
            products = []
            #seller id is included here but not used by application
            for (product_id, name, seller_id, price, rating) in results:
                product = {
                    "product_id": product_id,
                    "name": name,
                    "image_path": '',
                    "price": price,
                    "rating": rating
                }
                products.append(product)
            return products

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")
```

3. Program Functioning

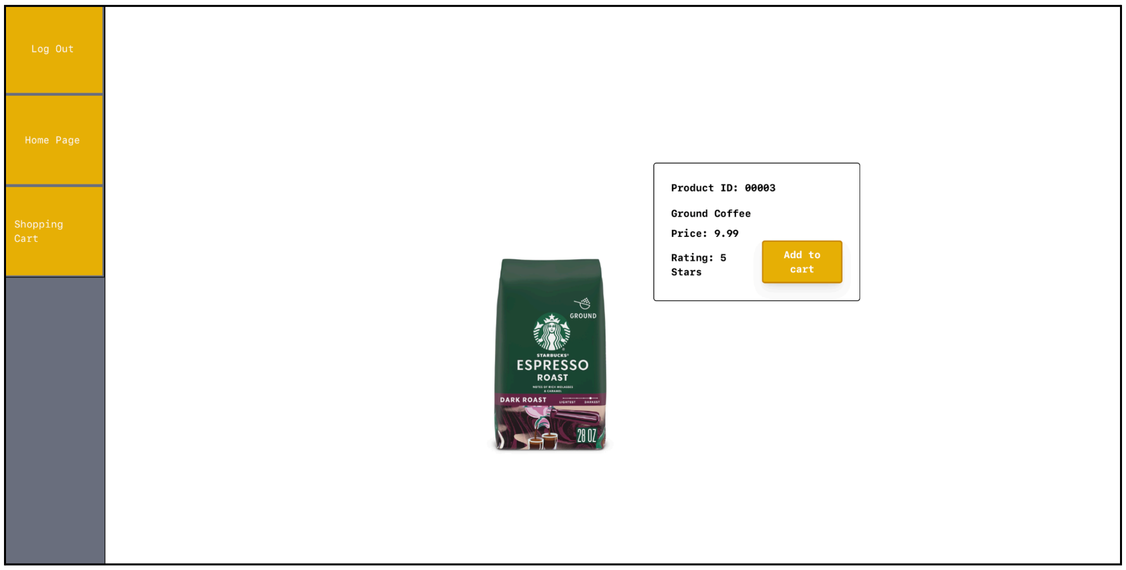


Terminal Output:

```
 GET /HomePage?customer_id=00002 200 in 27ms
127.0.0.1 - - [04/Nov/2024 20:33:42] "OPTIONS /user/products HTTP/1.1" 200 -
Connection acquired from pool.
[{'product_id': '00001', 'name': 'Case of water', 'image_path': '', 'price': Decimal('10.99'),
'rating': Decimal('1')}, {'product_id': '00002', 'name': 'JOLLY RANCHER', 'image_path':
'', 'price': Decimal('12.99'), 'rating': Decimal('4')}, {'product_id': '00003', 'name': 'Ground
Coffee', 'image_path': '', 'price': Decimal('9.99'), 'rating': Decimal('5')}, {'product_id':
'00004', 'name': 'Hand Soap', 'image_path': '', 'price': Decimal('12.99'), 'rating':
Decimal('5')}, {'product_id': '00005', 'name': 'SHARPIES', 'image_path': '', 'price':
Decimal('20.99'), 'rating': Decimal('3')}, {'product_id': '00006', 'name': 'Toothbrushes',
'image_path': '', 'price': Decimal('24.99'), 'rating': Decimal('3')}]
Connection returned to pool.
```

3.2)

Amazon Product Page

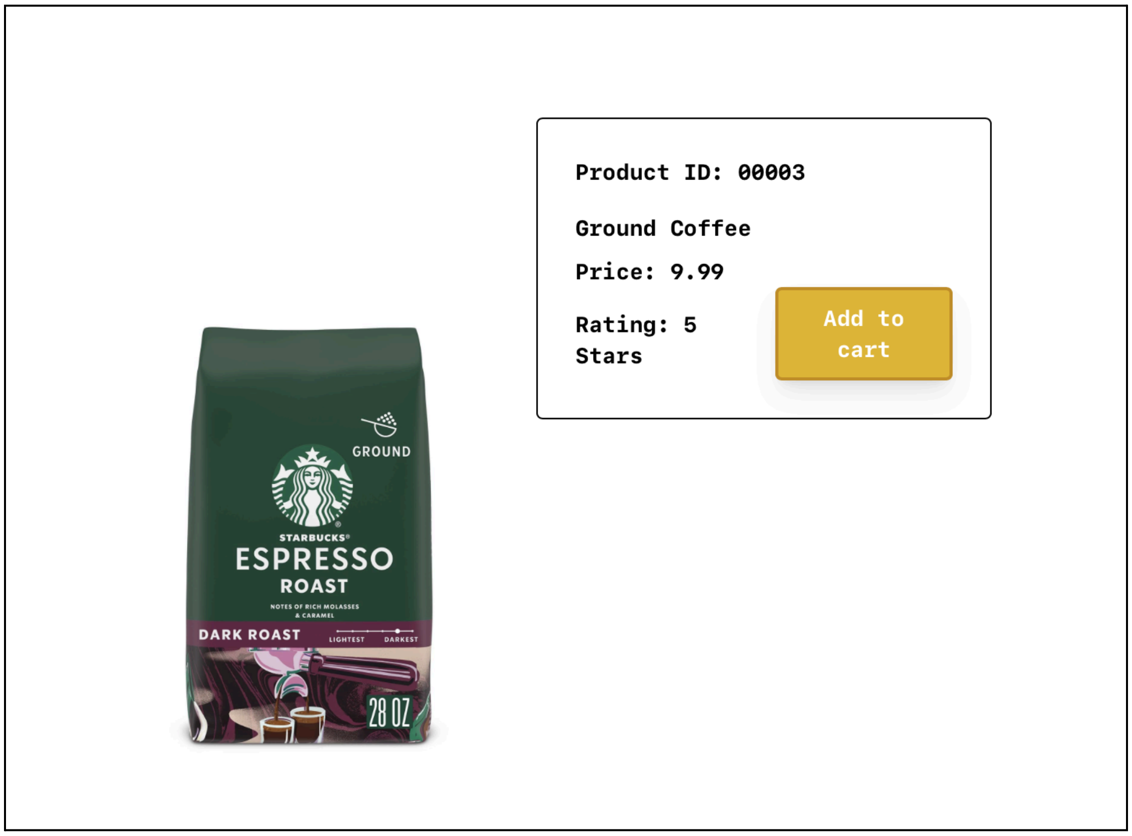


Product ID: 00003

Ground Coffee

Price: 9.99

Rating: 5 Stars



Product ID: 00003

Ground Coffee

Price: 9.99

Rating: 5 Stars

1. <u>Description:</u>
   This is the “Amazon Product Page”. It displays all the information stored in the database related to the particular product that the user selects. The python program runs a query

that selects a **product** from the product table in the database based on the provided host variable “product_id”. The page also allows the user to add that particular item to their cart. It does this by querying the database for the existing items in the customer’s shopping cart from the **shopping_cart** table. Then it checks if the product is already in the customer’s cart based on product_id. If the product is already in the customer’s cart then a query is processed to increment the “num_of_prod_in_cart” for that product in the customer’s cart using the **shopping_cart** table. If the product does not already exist in the customer’s car then a query is made to create a new row in the **shopping_cart** table associated with the customer_id, product_id, and num_of_prod_in_cart(1). Like the home page the customer may navigate utilizing the taskbar.

2. Python Code:

Endpoints(Flask)

```
# handle retrieving product info
@user_bp.route('/product_info', methods=['POST'])
def productInfo():
    data = request.get_json()
    product_id = data['product_id']
    # SQL query to retrieve demo product info from db ####
    product_details = getProductDetails(product_id=product_id)

    return jsonify({"product":product_details}), 201
# add item to Shopping Cart
@user_bp.route('/add_to_cart', methods=['POST'])
def addTOCart():
    data = request.get_json()
    customer_id = data['customer_id']
    product_id = data['product_id']
    # call sql database for user shopping cart
    account_specific_shopping_cart = getShoppingCart(customer_id=customer_id)
    try:
        #check if product in cart
        product_in_cart = False
        for curr_product in account_specific_shopping_cart:
            if curr_product['product_id'] == product_id:
                product_in_cart = True
                #sql query to update the sql db for incremented product in cart
                addExistingProductToCart(customer_id=customer_id,
product_id=product_id)
                break

        if not product_in_cart:
            # sql query for adding new prod to cart
            addNewProductToCart(customer_id=customer_id,product_id=product_id)

        return jsonify({"message":"Product Added"}), 200
    except:
        # handle none type for account_specific_shopping_cart if user has no cart
        addNewProductToCart(customer_id=customer_id,product_id=product_id) #sql
query
        return jsonify({"message":"Product Added"}), 200
```

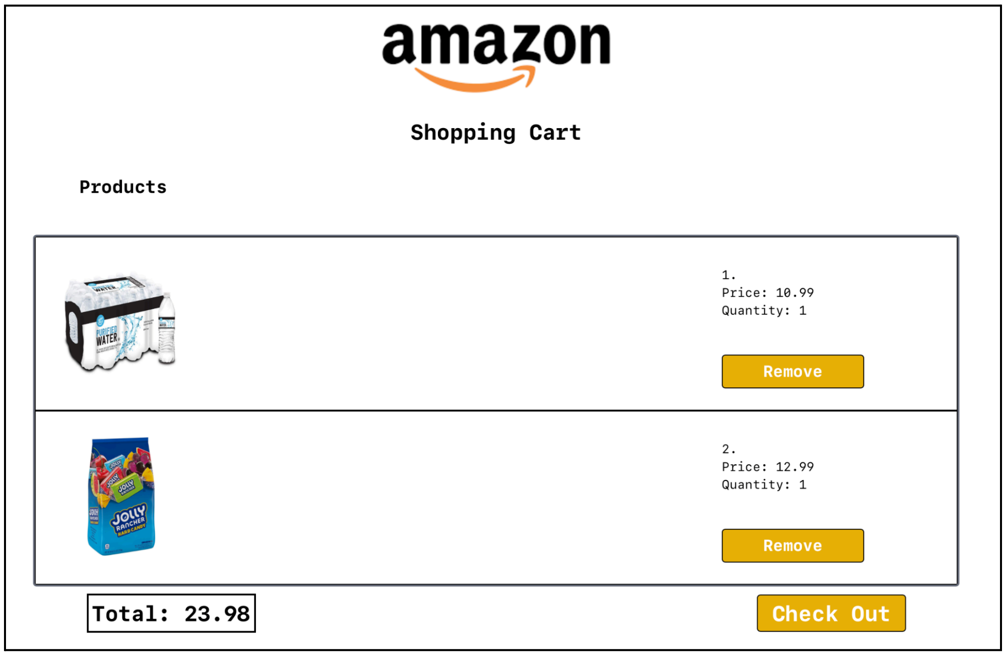
SQL Commands (MySql)

```
# Query 9 get product details based on product_id
def getProductDetails(product_id):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "SELECT * "
                "FROM product "
                "WHERE product_id = %s"
            )
            cursor.execute(query, (product_id,))
            result = cursor.fetchone()

            if result:
                product_id, name, seller_id, price, rating = result
                product = {
                    "product_id": product_id,
                    "name": name,
                    "image_path": '',
                "price": price,
                "rating": rating
            }
            return product
        else:
            print("No product found.")
            return None

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 1: Select customer information and their shopping cart details
def getShoppingCart(customer_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query1 = (
                "SELECT customer_id, S.product_id, num_of_prod_in_cart, price "
                "FROM customer_account NATURAL JOIN shopping_cart AS S JOIN
product AS P ON S.product_id = P.product_id "
                "WHERE customer_id = %s"
            )
            cursor.execute(query1, (customer_id,))
            results = cursor.fetchall()
            print("Query 1 Results:")
            if not results:
                print(f"No data found for customer_id {customer_id}.")
            else:
                customer_cart = []
                for (customer_id, product_id, num_of_prod_in_cart, price) in results:
```

```
                #print(f"Customer ID: {customer_id}, Product ID: {product_id}, Number of
Products: {num_of_prod_in_cart}, Price: {price}")
                cart_item = {
                    "customer_id":customer_id,
                    "product_id":product_id,
                    "num_of_prod_in_cart": num_of_prod_in_cart,
                    "price":price
                }
                customer_cart.append(cart_item)
            return customer_cart

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 3 increments products in the cart that already exist in the cart
def addExistingProductToCart(customer_id, product_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query6 = (
                "UPDATE shopping_cart "
                "SET num_of_prod_in_cart =  num_of_prod_in_cart + 1 "
                "WHERE customer_id = %s AND product_id = %s "
            )

            cursor.execute(query6, (customer_id,product_id))
            connection.commit()  # Commit changes for the update query
            print(f"Product ID: {product_id} Added to Customer ID: {customer_id}")

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 4 add products to cart that don't currently exist in cart
def addNewProductToCart(customer_id, product_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query7 = (
                "INSERT INTO shopping_cart VALUES(%s, %s, '1') "
            )
            cursor.execute(query7, (customer_id,product_id))
            connection.commit()  # Commit changes for the update query
            print(f"Product ID: {product_id} Added to Customer ID: {customer_id}")
```

3. Program Functioning

Before


Shopping Cart

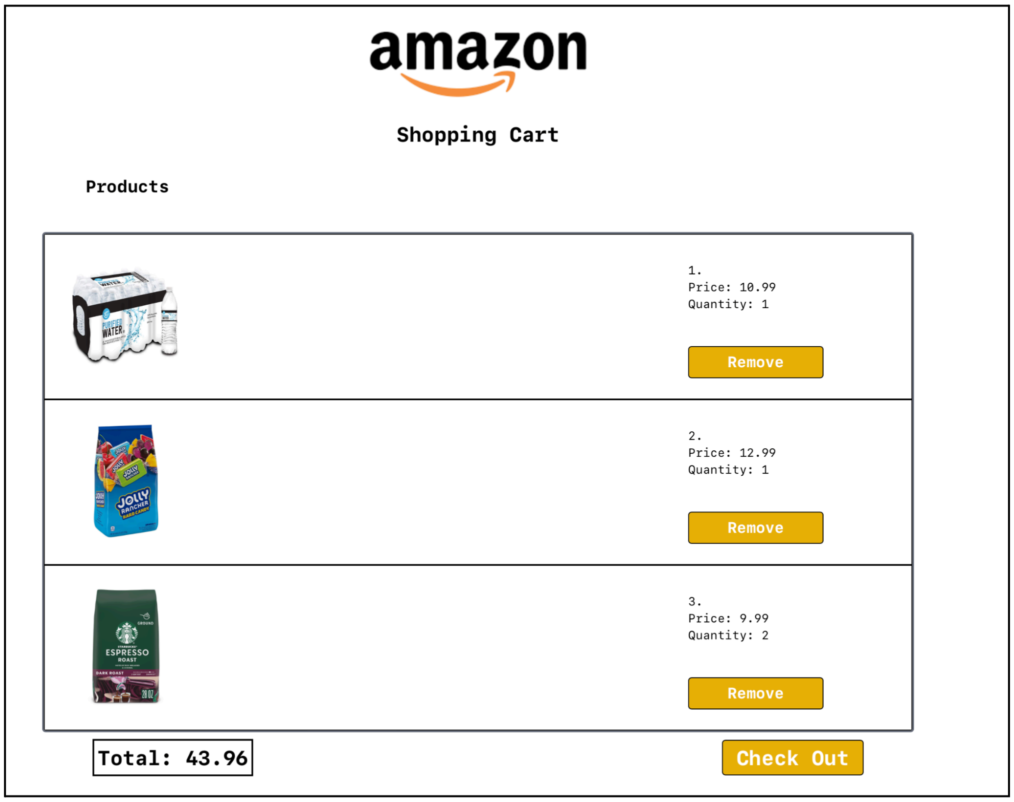
Products

1.
Price: 10.99
Quantity: 1

2.
Price: 12.99
Quantity: 1

Total: 23.98

After


Shopping Cart

Products

1.
Price: 10.99
Quantity: 1

2.
Price: 12.99
Quantity: 1

3.
Price: 9.99
Quantity: 2

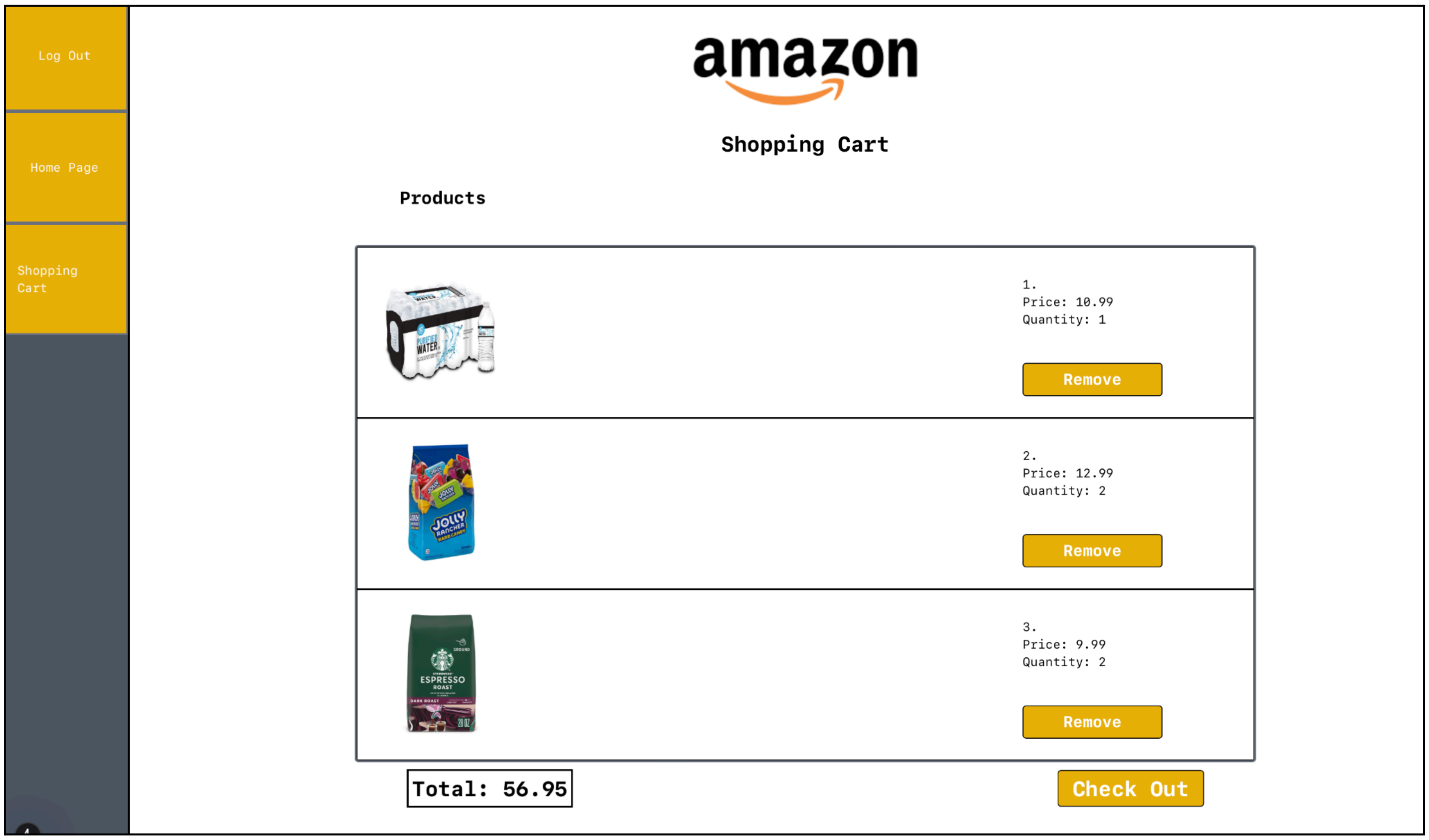
Total: 43.96

Terminal Output:

```
{'product_id': '00002', 'name': 'JOLLY RANCHER', 'image_path': '', 'price':
Decimal('12.99'), 'rating': Decimal('4')}
Product ID: 00002 Added to Customer ID: 00002
```

3.3)

Shopping Cart Page



1. Description

   This is the Shopping Cart Page. It Allows the customer to view the products that they have added to their cart. The python code makes a query to the **shopping_cart** table to retrieve the relevant rows associated with the customer's customer_id. It does this by naturally joining **customer_account** and **shopping_cart** and joining with **product** via shared product_id. It specifies selected items by customer_id. A separate query is made to calculate the total cost of the customer's shopping cart. This is done by calling the cart_total procedure. This procedure selects the sum of the (price * num_of_prod_in_cart) from the **shopping_cart** naturally joined with **product**, groups them by customer_id and selects the total associated with the current customer. There is functionality to remove items from their cart. This behaves similarly to how items are added. The python code makes one of two queries based on the quantity of a product. If the customer has more than one of a product they system decrements the num_of_prod_in_cart by 1. If there is only one of that product the system removes the row associated with the product. The check out feature simulates the functionality by clearing the shopping cart table of all the rows associated with that particular customer_id.

2. Python Code:

Endpoints(Flask)

```
    # handle retrieving Customer Account Shopping Cart
@user_bp.route('/cart', methods=['POST'])
def cart():
    data = request.get_json()
    try:
        # call sql database for user shopping cart
        account_specific_shopping_cart getShoppingCart(customer_id=data['customer_id'])
        # sql query to calculate Shopping Cart Total
        total = getShoppingCartTotal(customer_id=data['customer_id'])

        return jsonify({"shopping_cart": account_specific_shopping_cart, "total": total}),
201
    except:
        return jsonify({"shopping_cart": [], "total": 0}), 201

# remove product from Shopping Cart
@user_bp.route('/remove_from_cart', methods=['POST'])
def removeFromCart():
    data = request.get_json()
    customer_id = data['customer_id']
    product_id = data['product_id']
    account_specific_shopping_cart =
getShoppingCart(customer_id=data['customer_id']) #sql query

    # sql query to remove individual items from cart
    for product in account_specific_shopping_cart:
        if int(product['product_id']) == product_id:
            if product['num_of_prod_in_cart'] > 1:

decrimentProductCountFromCart(customer_id=customer_id,product_id=product_i
d) #sql query decriments count by 1
            else:

removeProductFromCart(customer_id=customer_id,product_id=product_id) #sql
query removes item entirely
                break

    return jsonify({"message":"Product Removed"}), 200

# handle Customer Account Shopping Cart check out
@user_bp.route('/check_out', methods=['POST'])
def checkOut():
    data = request.get_json()
    customer_id = data['customer_id']
    account_specific_shopping_cart = getShoppingCart(customer_id=customer_id) #
sql query for retrieving customer cart

    #sql query to simulate customer check out
    processCheckOut(customer_id=customer_id)

    if not account_specific_shopping_cart:
        return jsonify({"message": "Cart is already empty or undefined"}), 400

    return jsonify({"message":"Customer Checked Out"}), 200
```

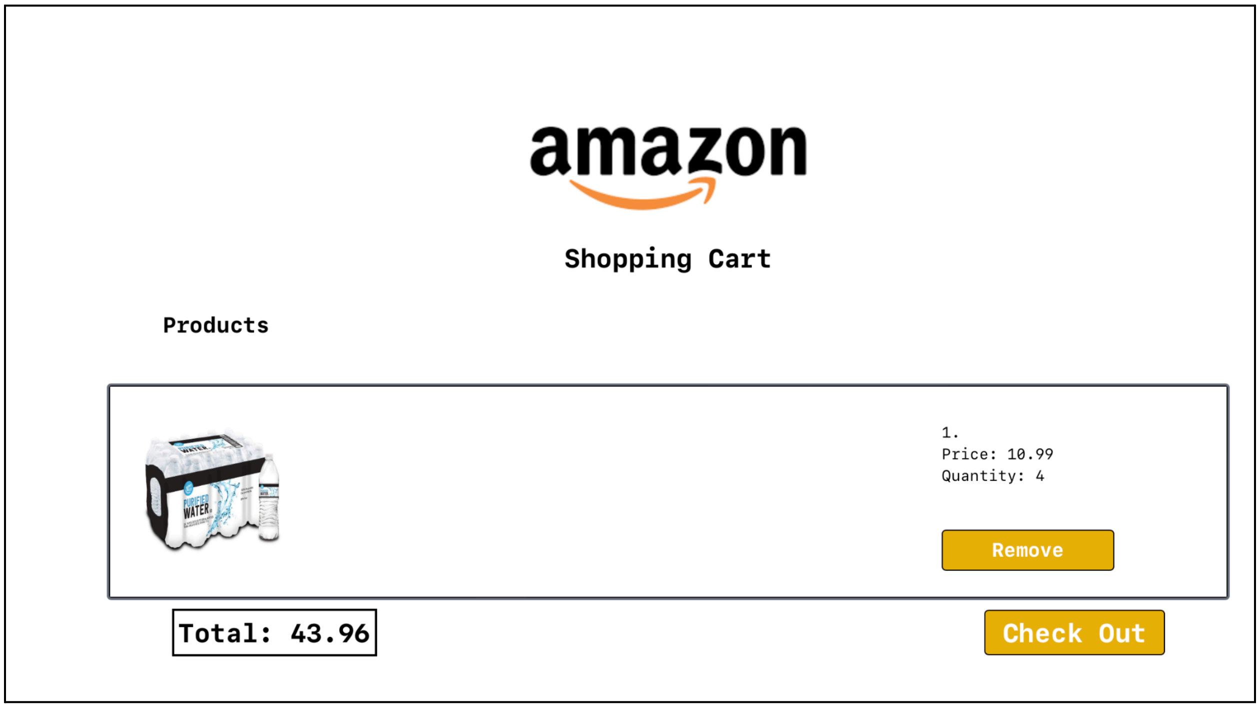
SQL Commands (MySql)

```
# Query 1: Select customer information and their shopping cart details
def getShoppingCart(customer_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
```

```
        query1 = (
            "SELECT customer_id, S.product_id, num_of_prod_in_cart, price "
            "FROM customer_account NATURAL JOIN shopping_cart AS S JOIN
product AS P ON S.product_id = P.product_id "
            "WHERE customer_id = %s"
        )
        cursor.execute(query1, (customer_id,))
        results = cursor.fetchall()
        print("Query 1 Results:")
        if not results:
            print(f"No data found for customer_id {customer_id}.")
        else:
            customer_cart = []
            for (customer_id, product_id, num_of_prod_in_cart, price) in results:
                #print(f"Customer ID: {customer_id}, Product ID: {product_id}, Number of
Products: {num_of_prod_in_cart}, Price: {price}")
                cart_item = {
                    "customer_id":customer_id,
                    "product_id":product_id,
                    "num_of_prod_in_cart": num_of_prod_in_cart,
                    "price":price
                }
                customer_cart.append(cart_item)
            return customer_cart

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 2: Calculate total shopping cart cost for each customer based on their cart
contents
def getShoppingCartTotal(customer_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None
    try:
        with connection.cursor(buffered=True) as cursor:
        # query to execute the procedure
        sub_query = (
            "CALL cart_total(%s, @c_total) "
        )
        cursor.execute(sub_query, (customer_id,))
        #query to select the returned value of the procedure
        query = (
            "SELECT @c_total "
        )
        cursor.execute(query)
        results = cursor.fetchall()
        print(results)
        if not results:
            print(f"No data found for customer_id {customer_id}.")
        else:
            print("\nQuery 3 Results:")
            for (total_cost) in results:
                print(f"Customer ID: {customer_id}, Total Cost: {total_cost}")
                # rounded the value in the tuple and reassigned to new tuple => parent
expects tuple object
                rounded_total_cost = (round(total_cost[0], 2),)
                return rounded_total_cost

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 6 deciment product count from cart
def decrimentProductCountFromCart(customer_id, product_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
        query6 = (
            "UPDATE shopping_cart "
            "SET num_of_prod_in_cart =  num_of_prod_in_cart - 1 "
            "WHERE customer_id = %s AND product_id = %s "
        )

        cursor.execute(query6, (customer_id,product_id))
        connection.commit()  # Commit changes for the update query
        print(f"Product ID: {product_id} Decrimented to Customer ID: {customer_id}")


    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 5 remove products from cart
def removeProductFromCart(customer_id, product_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
        query7 = (
            "DELETE FROM shopping_cart "
            "WHERE customer_id = %s AND product_id = %s "
        )
        cursor.execute(query7, (customer_id,product_id))
        connection.commit()  # Commit changes for the update query
        print(f"Product ID: {product_id} Removed from Customer ID: {customer_id}")
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 7 clears shopping cart for particular user simulates check out
def processCheckOut(customer_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query7 = (
                    "DELETE FROM shopping_cart "
                    "WHERE customer_id = %s "
                )
            cursor.execute(query7, (customer_id,))
            connection.commit()  # Commit changes for the update query
            print(f"Customer Checked Out All Products Removed from Customer ID:
{customer_id}")


    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")
```

3. Program Functioning

Before

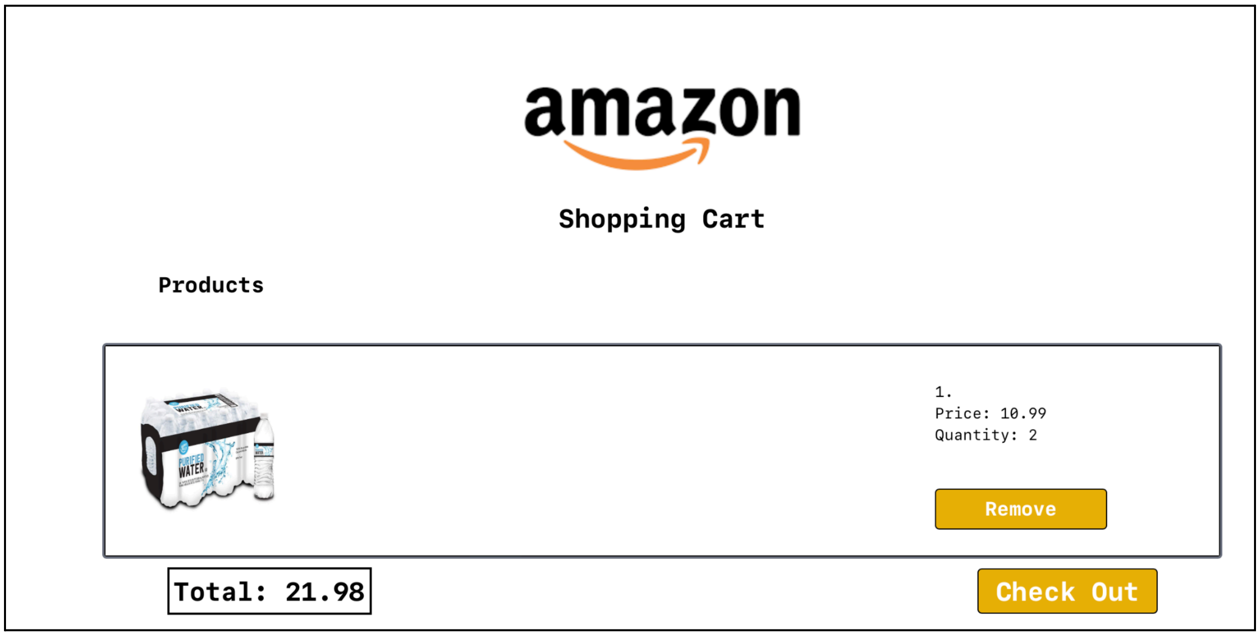

Shopping Cart

Products

1.
Price: 10.99
Quantity: 4

Total: 43.96

After


Shopping Cart

Products

1.
Price: 10.99
Quantity: 2

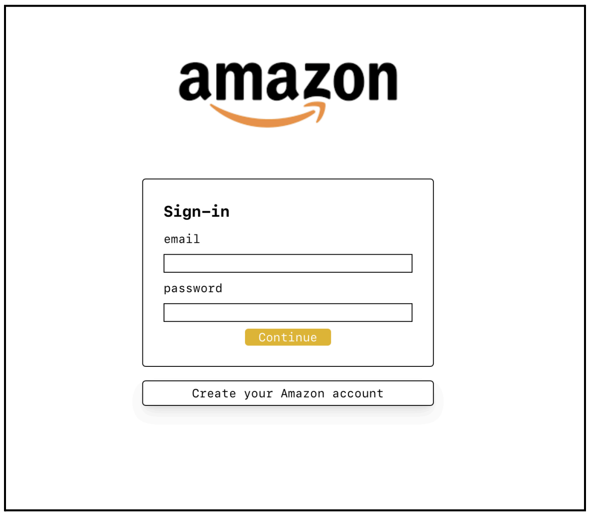
Total: 21.98

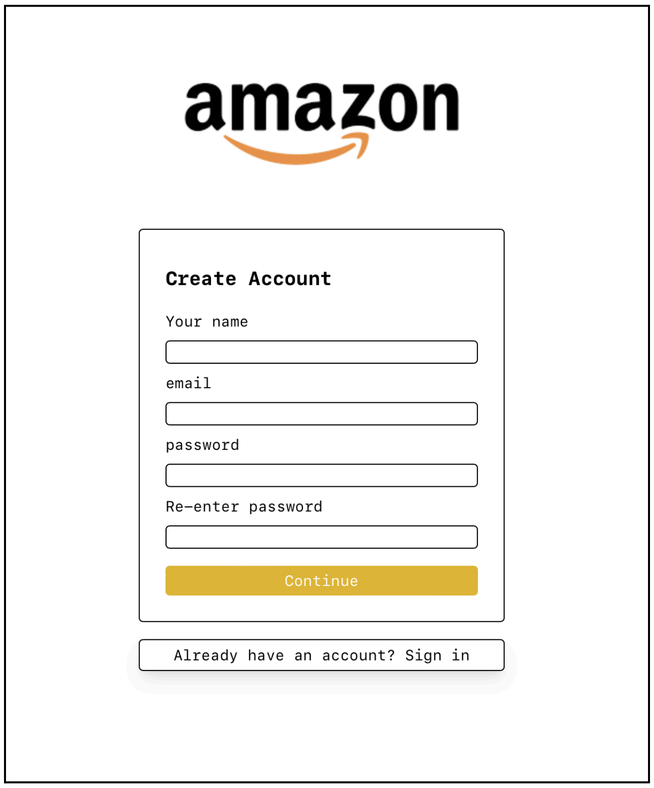
Terminal Output:

```
[{'customer_id': '00002', 'product_id': '00001', 'num_of_prod_in_cart': Decimal('2'),
'price': Decimal('10.99')}]
Customer ID: 00002, Total Cost: (45.959999084472656,)
Product ID: 2 Decremented to Customer ID: 00002
Product ID: 1 Removed from Customer ID: 00002
Customer Checked Out All Products Removed from Customer ID: 00002
```

3.4)

Login/Register Pages





1. <u>Description:</u>
   These are the “Login/Register Pages”. They allow the customer to both login and register for the amazon marketplace web application. The registration page allows the customer to enter their information based on the allotted input fields. When the customer submits the form the python script sends a query to the database checking if the customer account already exists. It checks the unique email attribute associated with a **customer_account** entry. If the account does not exist, a query is made to create a new **customer_account** entry. A third query is utilized

to get the current count of the customer base. This is done through a custom sql function “customer_count( )”. This count generates an appropriate customer_id(count + 1). The login page behaves similarly however if the account exists then the password in the database is compared to the password provided by the customer attempting to login.

2. Python Code:

Endpoints(Flask)

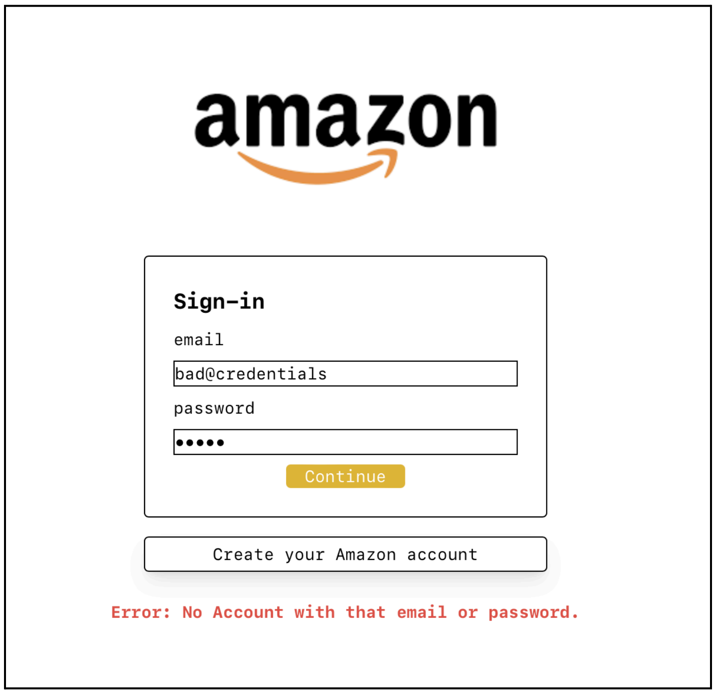
```
# handle user registration
@user_bp.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    # get the data passed
    email = data['email']
    passwd = data['passwd']
    name = data['name']
    address = "12345 road lane" # hard coded no address functionality
    credit_card_num = data['credit_card_num']

    #sql query to retrieve the customer account based on email input
    customer_account = getCustomerAccount(email=email)
    #sql query to retrieve count of total customer_accounts
    customer_id = int(getCustomerBaseCount()) + 1
    # front fill with '0's to conform to predefined structure
    customer_id = str(customer_id).zfill(5)

    # return 400 if account exists
    if customer_account:
        response = {
            'message': 'Email already registered.',
        }
        return response, 400
    else:
        # Hash passwd
        password = passwd
        hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
        createCustomerAccount(customer_id=customer_id, email=email, name=name,
passwd=hashed_password, address=address, credit_card_num=credit_card_num)
```

```
        response = {
            'message': 'Customer Account Created',
        }
        return jsonify(response), 201

# handle user login
@user_bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    # get the data passed
    email = data['email']
    passwd = data['passwd']

    try:
        #sql query to retrieve the customer account based on email input
        customer_account = getCustomerAccount(email=email)
        # check provided email w/ the email in db
        if customer_account[1] == email:
            if bcrypt.check_password_hash(customer_account[3], passwd): #check the hased
passwd with the one provided by the customer
                return jsonify(access_token="dummy token",
customer_id=customer_account[0]), 200 # return customer_id and dummy token
            else:
                return jsonify({"message":"Invalid credentials"}), 401
    except:
        return jsonify({"message":"Invalid credentials"}), 401
```

SQL Commands (MySql)

```
# Query 10 get customer account information
def getCustomerAccount(email):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "SELECT * "
                "FROM customer_account "
                "WHERE email = %s"
            )
            cursor.execute(query, (email,))
            result = cursor.fetchone()

            # return the customer account associated w/ email or none
            return result

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 11 get count of rows in customer_account table
# this is used to assign customer_id on registration
def getCustomerBaseCount():
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "select customer_count() "
            )
            cursor.execute(query)
            result = cursor.fetchone()

            # return count of rows or number of customer accounts
            return result[0]

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 12 create customer account
def createCustomerAccount(customer_id, email, name, passwd, address,
credit_card_num):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "INSERT INTO customer_account VALUES"
                "(%s,%s,%s,%s,%s,%s)"
            )
            cursor.execute(query, (customer_id, email, name, passwd, address,
credit_card_num,))
            connection.commit()

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")
```

3. Program Functioning


Sign-in

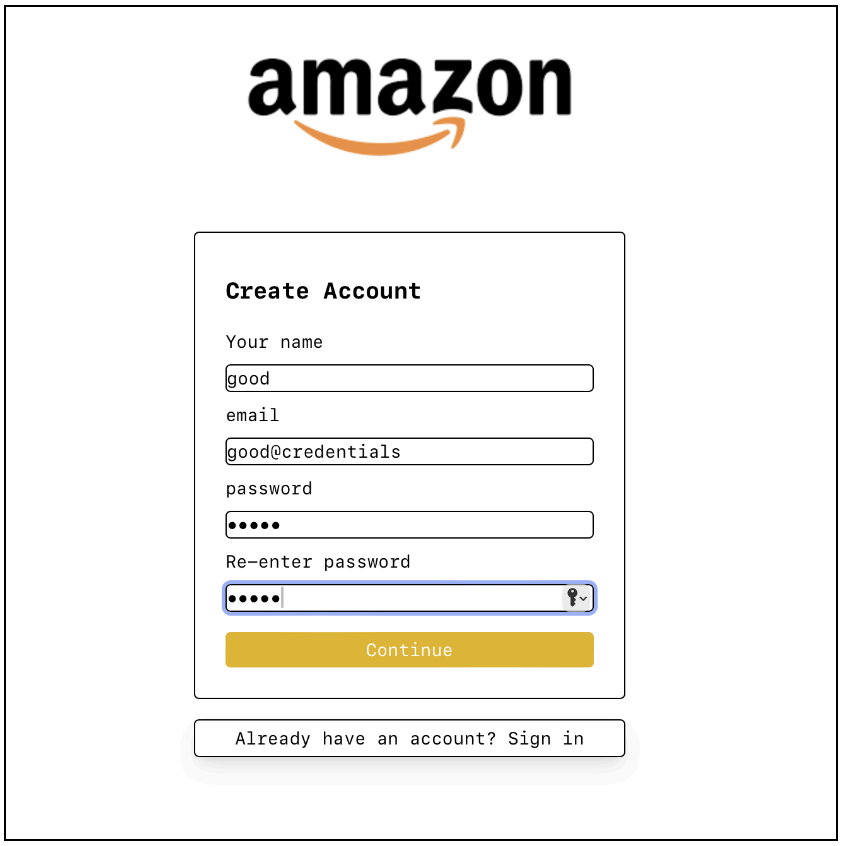
email

bad@credentials

password

•••••

Error: No Account with that email or password.


Create Account

Your name

good

email

good@credentials

password

•••••

Re-enter password

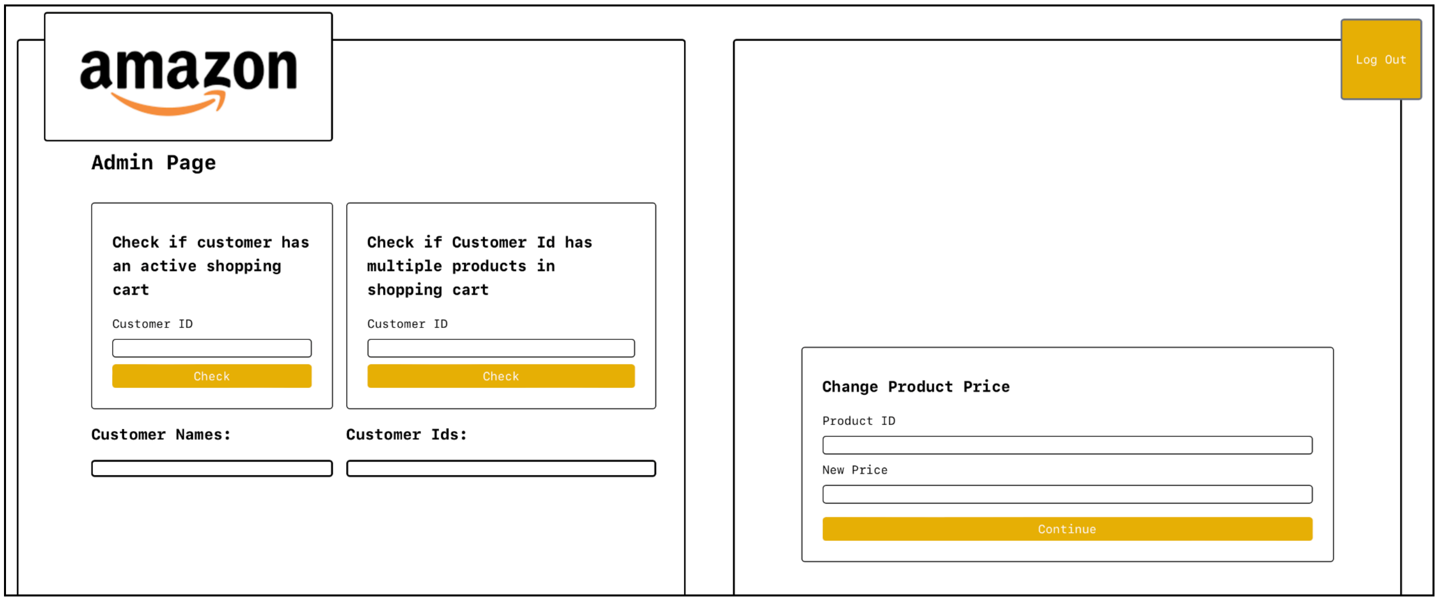
•••••

Terminal Output:

```
New Customer_Account Created
Bad Login Credencials
```

3.5)

Admin Page



1. <u>Description</u>

   This is the “Admin Page”. It allows admin personnel to check if a customer has an active shopping cart, if the customer has multiple items in their cart, and to change the price of a product. To update the price the admin user inputs the product_id and new_price for an existing product in the **product** table. A preliminary query is called to check if the product exists based on the product_id host variable. If the product exists an update query is called to set the price of the product to the new price based on the host variables new_price and product_id. The admin user checks if a customer has an active shopping cart based on the customer_id host variable. The query checks if the customer id is in the **shopping_cart** table and that it matches the customer_id host variable. It then returns the name of that customer from the **customer_account** table. To check if the customer has multiple items in their shopping cart a subquery is utilized that selects the customer_ids from the **shopping_cart** table that match the customer id in the host variable, groups them by customer_id and isolates only the grouplings that have a product_id greater than 1. The main query checks if the host variable exists in that subquery and returns the customer_id.

2. Python Code:

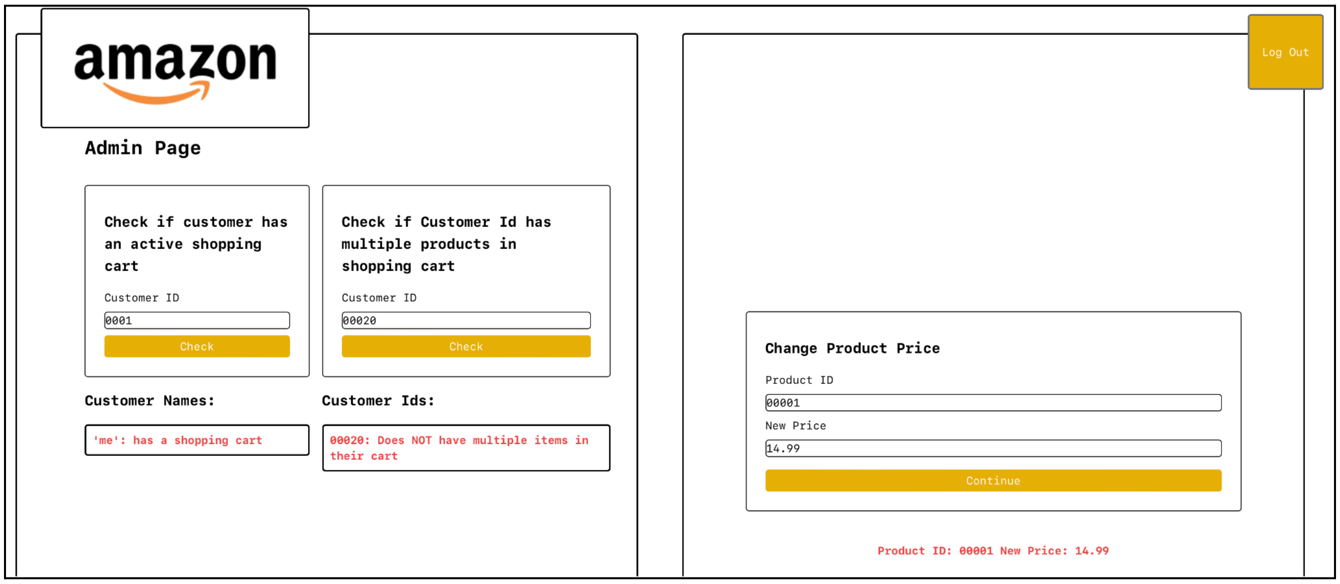
Endpoints(Flask)

```
# handle change of product price
@user_bp.route('/change_price', methods=['POST'])
def change_product_price():
    data = request.get_json()
    product_id = data['product_id']
    new_price = data['new_price']

    price_changed = updateProductPrice(product_id=product_id, new_price=new_price)
#sql query to update the price of product

    if not price_changed:
        return jsonify({"message":f"ERROR: Product ID: {product_id} does NOT exist in
DataBase"}), 200
    else:
        return jsonify({"message":f"Product ID: {product_id} New Price: {new_price}"}),
200


# handle retrieving customer ids w/ products in cart
@user_bp.route('/unique_prod_cart', methods=['POST'])
def unique_prod_in_cart():
    data = request.get_json()
    customer_id = data['customer_id']
    try:
        validated_customer_id = getUsersWithUniqueProducts(customer_id=customer_id) #
sql query for above note

        return jsonify(f"'{str(validated_customer_id[0])}': \nHas multiple items in their
cart"), 200
    except:
        return jsonify(f"{str(customer_id)}: \nDoes NOT have multiple items in their
cart"), 200
# handle retrieving customer names w/ active shopping cart
@user_bp.route('/active_carts', methods=['POST'])
def activeShoppingCart():
    data = request.get_json()
    customer_id = data['customer_id']
    try:
        customer_name = getUsersWithCart(customer_id=customer_id) # sql query for
above note
        return jsonify(f"'{customer_name[0]}': has a shopping cart"), 200
    except:
        return jsonify("No Cart Associated with Custome Id"), 200
```

SQL Commands (MySql)

```
# Query 13 checks if specified customer has an active shopping cart
def getUsersWithCart(customer_id):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "SELECT name "
                "FROM customer_account AS C "
                f"WHERE C.customer_id = {customer_id} AND customer_id IN ( "
                "SELECT customer_id "
                "FROM shopping_cart) "
            )
            cursor.execute(query)
            result = cursor.fetchone()
            return result
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 14 updates the price of a specific product
def updateProductPrice(product_id, new_price):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            # implimented to give user feedback on if the product exists
            sub_query = (
                f"SELECT * FROM product WHERE product_id = {product_id}"
            )
            cursor.execute(sub_query)
            result = cursor.fetchone()
            print(result)
            if not result:
                return False
            query = (
                f"UPDATE product SET price = {new_price} WHERE product_id = 
{product_id}"
            )
            cursor.execute(query)
            connection.commit()
            return True
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 15 displays the customer_id from the customer_account table if it has more than 
one product in their shopping cart
def getUsersWithUniqueProducts(customer_id):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "SELECT C.customer_id "
                "FROM customer_account AS C "
                f"WHERE C.customer_id = {customer_id} AND EXISTS( "
                "SELECT S.customer_id "
                "FROM shopping_cart AS S "
                "WHERE S.customer_id = C.customer_id "
                "GROUP BY S.customer_id "
                "HAVING COUNT(product_id) > 1) "
            )
            cursor.execute(query)
            result = cursor.fetchone()
            print(result)

            return result
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")
```

3. Program Functioning



Check if customer has an active shopping cart

Customer ID: 0001

Check if Customer Id has multiple products in shopping cart

Customer ID: 00020

Customer Names:

'me': has a shopping cart

Customer Ids:

00020: Does NOT have multiple items in their cart

Change Product Price

Product ID: 00001

New Price: 14.99

Product ID: 00001 New Price: 14.99

Terminal Output:

```
('me',) has an active shopping cart
None
('00001', 'Case of water', '00001', Decimal('14.99'), Decimal('1'))
00001 New Price: 14.99, Updated
```

Part 4

Extra Credit Graphic UI Development:

- A video file along with the source code for the project can be found in the github repository at this URL:

https://github.com/cashhollister2u/Report3/releases/tag/1.3

Reference Material

sql_commands.py

SQL Queries that the Flask Server(python) utilizes to call the DataBase:
Note: MySql “connector” and “pooling” had to be used due to memory constraints.

- Utilized to manage cursor instances and clean up between sql queries.

```
import mysql.connector
from mysql.connector import pooling

# Replaced cnx w/ the Connection Pooling
# Massive memory issues if not implimented like this
connection_pool = pooling.MySQLConnectionPool(
    pool_name="amazon_mkt_place",
    pool_size=5,
    pool_reset_session=True,
    user='root',       # Replace with your MySQL username
    password='Msq070489',   # Replace with your MySQL password
    host='localhost',          # Replace with your MySQL server address
    database='amazon_marketplace'   # Replace with your database name
)

# funcition that allows working funcitons to connect to msql db
# optimizes memory and cleans up instances of cursor that
# may have been left unchecked
def get_connection_from_pool():
    try:
        connection = connection_pool.get_connection()
        print("Connection acquired from pool.")
        return connection
    except pooling.PoolError as err:
        print(f"Error acquiring connection from pool: {err}")
        return None

# Query 1: Select customer information and their shopping cart details
def getShoppingCart(customer_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None
    try:
        with connection.cursor(buffered=True) as cursor:
            query1 = (
                "SELECT customer_id, S.product_id, num_of_prod_in_cart, price "
                "FROM customer_account NATURAL JOIN shopping_cart AS S JOIN product AS P ON
S.product_id = P.product_id "
                "WHERE customer_id = %s"
            )
            cursor.execute(query1, (customer_id,))
            results = cursor.fetchall()
            print("Query 1 Results:")
            if not results:
                print(f"No data found for customer_id {customer_id}.")
            else:
                customer_cart = []
                for (customer_id, product_id, num_of_prod_in_cart, price) in results:
                    #print(f"Customer ID: {customer_id}, Product ID: {product_id}, Number of Products:
{num_of_prod_in_cart}, Price: {price}")
                    cart_item = {
                        "customer_id":customer_id,
                        "product_id":product_id,
                        "num_of_prod_in_cart": num_of_prod_in_cart,
                        "price":price
                    }
                    customer_cart.append(cart_item)
                    print(customer_cart)
                return customer_cart

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")


# Query 2: Calculate total shopping cart cost for each customer based on their cart contents
def getShoppingCartTotal(customer_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            # query to execute the procedure
            sub_query = (
                "CALL cart_total(%s, @c_total) "
            )
            cursor.execute(sub_query, (customer_id,))
            #query to select the returned value of the procedure
            query = (
                "SELECT @c_total "
            )
            cursor.execute(query)
            results = cursor.fetchall()
            print(results)
            if not results:
                print(f"No data found for customer_id {customer_id}.")
            else:
                print("\nQuery 3 Results:")
                for (total_cost) in results:
                    print(f"Customer ID: {customer_id}, Total Cost: {total_cost}")
                    # rounded the value in the tuple and reassigned to new tuple => parent expects tuple object
                    rounded_total_cost = (round(total_cost[0], 2),)
                    return rounded_total_cost

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")


# Query 3 increments products in the cart that already exist in the cart
def addExistingProductToCart(customer_id, product_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query6 = (
                "UPDATE shopping_cart "
                "SET num_of_prod_in_cart =  num_of_prod_in_cart + 1 "
                "WHERE customer_id = %s AND product_id = %s "
            )

            cursor.execute(query6, (customer_id,product_id))
            connection.commit()  # Commit changes for the update query
            print(f"Product ID: {product_id} Added to Customer ID: {customer_id}")

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")


# Query 4 add products to cart that don't currently exist in cart
def addNewProductToCart(customer_id, product_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query7 = (
                "INSERT INTO shopping_cart VALUES(%s, %s, '1') "
            )
            cursor.execute(query7, (customer_id,product_id))
            connection.commit()  # Commit changes for the update query
            print(f"Product ID: {product_id} Added to Customer ID: {customer_id}")


    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")


# Query 5 remove products from cart
def removeProductFromCart(customer_id, product_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query7 = (
                "DELETE FROM shopping_cart "
                "WHERE customer_id = %s AND product_id = %s "
            )
            cursor.execute(query7, (customer_id,product_id))
            connection.commit()  # Commit changes for the update query
            print(f"Product ID: {product_id} Removed from Customer ID: {customer_id}")


    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")


# Query 6 deciment product count from cart
def decrimentProductCountFromCart(customer_id, product_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query6 = (
                "UPDATE shopping_cart "
                "SET num_of_prod_in_cart =  num_of_prod_in_cart - 1 "
            "WHERE customer_id = %s AND product_id = %s "
        )

        cursor.execute(query6, (customer_id,product_id))
        connection.commit()  # Commit changes for the update query
        print(f"Product ID: {product_id} Decrimented to Customer ID: {customer_id}")


    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 7 clears shopping cart for particular user simulates check out
def processCheckOut(customer_id):
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query7 = (
                "DELETE FROM shopping_cart "
                "WHERE customer_id = %s "
            )
            cursor.execute(query7, (customer_id,))
            connection.commit()  # Commit changes for the update query
            print(f"Customer Checked Out All Products Removed from Customer ID: {customer_id}")


    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")
```

```
# Query 8 get all products on the website
def getAllProducts():
    # Create a cursor object to interact with the database
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query8 = (
                "SELECT * "
                "FROM PRODUCT"
            )
            cursor.execute(query8)
            results = cursor.fetchall()
            products = []
            #seller id is included here but not used by application
            for (product_id, name, seller_id, price, rating) in results:
                product = {
                    "product_id": product_id,
                    "name": name,
                    "image_path": '',
                    "price": price,
                    "rating": rating
                }
                products.append(product)
            print(products)
            return products

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 9 get product details based on product_id
def getProductDetails(product_id):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None
    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "SELECT * "
                "FROM product "
                "WHERE product_id = %s"
            )
            cursor.execute(query, (product_id,))
            result = cursor.fetchone()

            if result:
                product_id, name, seller_id, price, rating = result
                product = {
                    "product_id": product_id,
                    "name": name,
                    "image_path": '',
                    "price": price,
                    "rating": rating
                }
                print(product)
                return product
            else:
                print("No product found.")
                return None

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 10 get customer account information
def getCustomerAccount(email):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "SELECT * "
                "FROM customer_account "
                "WHERE email = %s"
            )
            cursor.execute(query, (email,))
            result = cursor.fetchone()

            # return the customer account associated w/ email or none
            return result

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 11 get count of rows in customer_account table
# this is used to assign customer_id on registration
def getCustomerBaseCount():
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "select customer_count() "
            )
            cursor.execute(query)
            result = cursor.fetchone()

            # return count of rows or number of customer accounts
            return result[0]

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")
```

```
# Query 12 create customer account
def createCustomerAccount(customer_id, email, name, passwd, address, credit_card_num):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "INSERT INTO customer_account VALUES"
                "(%s,%s,%s,%s,%s,%s)"
            )
            cursor.execute(query, (customer_id, email, name, passwd, address, credit_card_num,))
            connection.commit()

    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")


##### misc admin functionality tools #####

# Query 13 checks if specified customer has an active shopping cart
def getUsersWithCart(customer_id):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "SELECT name "
                "FROM customer_account AS C "
                f"WHERE C.customer_id = {customer_id} AND customer_id IN ( "
                "SELECT customer_id "
                "FROM shopping_cart) "
            )
            cursor.execute(query)
            result = cursor.fetchone()
            return result
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")

# Query 14 updates the price of a specific product
def updateProductPrice(product_id, new_price):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            # implimented to give user feedback on if the product exists
            sub_query = (
                f"SELECT * FROM product WHERE product_id = {product_id}"
            )
            cursor.execute(sub_query)
            result = cursor.fetchone()
            print(result)
            if not result:
                return False
            query = (
                f"UPDATE product SET price = {new_price} WHERE product_id = {product_id}"
            )
            cursor.execute(query)
            connection.commit()
            return True
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")
```

```
# Query 15 displays the customer_id from the customer_account table if it has more than one product in
their shopping cart
def getUsersWithUniqueProducts(customer_id):
    connection = get_connection_from_pool()
    if connection is None:
        print("Failed to get a connection from the pool.")
        return None

    try:
        with connection.cursor(buffered=True) as cursor:
            query = (
                "SELECT C.customer_id "
                "FROM customer_account AS C "
                f"WHERE C.customer_id = {customer_id} AND EXISTS( "
                "SELECT S.customer_id "
                "FROM shopping_cart AS S "
                "WHERE S.customer_id = C.customer_id "
                "GROUP BY S.customer_id "
                "HAVING COUNT(product_id) > 1) "
            )
            cursor.execute(query)
            result = cursor.fetchone()
            print(result)

            return result
    except mysql.connector.Error as err:
        print(f"Error: {err}")
        return None

    finally:
        connection.close()  # Return the connection to the pool
        print("Connection returned to pool.")
```

endpoints.py

```
from flask import Blueprint, jsonify, request

# Custom imports
from extensions import bcrypt
from sql_commands import getUsersWithCart, getUsersWithUniqueProducts, updateProductPrice,
createCustomerAccount, getCustomerBaseCount, getCustomerAccount, processCheckOut,
decrimentProductCountFromCart, removeProductFromCart, getProductDetails, getAllProducts,
getShoppingCart, getShoppingCartTotal, addExistingProductToCart, addNewProductToCart
```

```
user_bp = Blueprint('user', __name__)


# handle user registration
@user_bp.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    # get the data passed
    email = data['email']
    passwd = data['passwd']
    name = data['name']
    address = "12345 road lane" # hard coded no address functionality
    credit_card_num = data['credit_card_num']

    #sql query to retrieve the customer account based on email input
    customer_account = getCustomerAccount(email=email)
    #sql query to retrieve count of total customer_accounts
    customer_id = int(getCustomerBaseCount()) + 1
    # front fill with '0's to conform to predefined structure
    customer_id = str(customer_id).zfill(5)

    # return 400 if account exists
    if customer_account:
        response = {
            'message': 'Email already registered.',
        }
        return response, 400
    else:
        # Hash passwd
        password = passwd
        hashed_password = bcrypt.generate_password_hash(password).decode('utf-8')
        createCustomerAccount(customer_id=customer_id, email=email, name=name,
passwd=hashed_password, address=address, credit_card_num=credit_card_num)

        response = {
            'message': 'Customer Account Created',
        }
        return jsonify(response), 201


# handle user login
@user_bp.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    # get the data passed
    email = data['email']
    passwd = data['passwd']

    try:
        #sql query to retrieve the customer account based on email input
        customer_account = getCustomerAccount(email=email)
        # check provided email w/ the email in db
        if customer_account[1] == email:
            if bcrypt.check_password_hash(customer_account[3], passwd): #check the hased passwd with the
one provided by the customer
                return jsonify(access_token="dummy token", customer_id=customer_account[0]), 200 # return
customer_id and dummy token
            else:
                return jsonify({"message":"Invalid credentials"}), 401
    except:
        return jsonify({"message":"Invalid credentials"}), 401


# handle retrieving Home Page products
@user_bp.route('/products', methods=['POST'])
def products():
    # sql query to retrieve all products on website
    products = getAllProducts()
    return jsonify({"products":products}), 201


# handle retrieving product info
@user_bp.route('/product_info', methods=['POST'])
def productInfo():
    data = request.get_json()
    product_id = data['product_id']
    # SQL query to retrieve demo product info from db ####
    product_details = getProductDetails(product_id=product_id)

    return jsonify({"product":product_details}), 201


# handle retrieving Customer Account Shopping Cart
@user_bp.route('/cart', methods=['POST'])
def cart():
    data = request.get_json()
    try:
        # call sql database for user shopping cart
        account_specific_shopping_cart = getShoppingCart(customer_id=data['customer_id'])
        # sql query to calculate Shopping Cart Total
        total = getShoppingCartTotal(customer_id=data['customer_id'])

        return jsonify({"shopping_cart": account_specific_shopping_cart, "total": total}), 201
    except:
        return jsonify({"shopping_cart": [], "total": 0}), 201

# add item to Shopping Cart
@user_bp.route('/add_to_cart', methods=['POST'])
def addTOCart():
    data = request.get_json()
    customer_id = data['customer_id']
    product_id = data['product_id']
    # call sql database for user shopping cart
    account_specific_shopping_cart = getShoppingCart(customer_id=customer_id)
    try:
        #check if product in cart
        product_in_cart = False
        for curr_product in account_specific_shopping_cart:
            if curr_product['product_id'] == product_id:
                product_in_cart = True
                #sql query to update the sql db for incremented product in cart
                addExistingProductToCart(customer_id=customer_id, product_id=product_id)
                break

        if not product_in_cart:
            # sql query for adding new prod to cart
            addNewProductToCart(customer_id=customer_id,product_id=product_id)

        return jsonify({"message":"Product Added"}), 200
    except:
        # handle none type for account_specific_shopping_cart if user has no cart
        addNewProductToCart(customer_id=customer_id,product_id=product_id) #sql query
        return jsonify({"message":"Product Added"}), 200


# remove product from Shopping Cart
@user_bp.route('/remove_from_cart', methods=['POST'])
def removeFromCart():
    data = request.get_json()
    customer_id = data['customer_id']
    product_id = data['product_id']
    account_specific_shopping_cart = getShoppingCart(customer_id=data['customer_id']) #sql query

    # sql query to remove individual items from cart
    for product in account_specific_shopping_cart:
        if int(product['product_id']) == product_id:
            if product['num_of_prod_in_cart'] > 1:
                decrimentProductCountFromCart(customer_id=customer_id,product_id=product_id) #sql query
decriments count by 1
            else:
                removeProductFromCart(customer_id=customer_id,product_id=product_id) #sql query
removes item entirely
                break

    return jsonify({"message":"Product Removed"}), 200


# handle Customer Account Shopping Cart check out
@user_bp.route('/check_out', methods=['POST'])
def checkOut():
    data = request.get_json()
    customer_id = data['customer_id']
    account_specific_shopping_cart = getShoppingCart(customer_id=customer_id) # sql query for
retrieving customer cart

    #sql query to simulate customer check out
    processCheckOut(customer_id=customer_id)

    if not account_specific_shopping_cart:
        return jsonify({"message": "Cart is already empty or undefined"}), 400

    return jsonify({"message":"Customer Checked Out"}), 200


#### admin endpoints

# handle change of product price
@user_bp.route('/change_price', methods=['POST'])
def change_product_price():
    data = request.get_json()
    product_id = data['product_id']
    new_price = data['new_price']

    price_changed = updateProductPrice(product_id=product_id, new_price=new_price) #sql query to
update the price of product
```

```
    if not price_changed:
        return jsonify({"message":f"ERROR: Product ID: {product_id} does NOT exist in DataBase"}),
200
    else:
        return jsonify({"message":f"Product ID: {product_id} New Price: {new_price}"}), 200


# handle retrieving customer ids w/ products in cart
@user_bp.route('/unique_prod_cart', methods=['POST'])
def unique_prod_in_cart():
    data = request.get_json()
    customer_id = data['customer_id']
    try:
        validated_customer_id = getUsersWithUniqueProducts(customer_id=customer_id) # sql query for
above note

        return jsonify(f"'{str(validated_customer_id[0])}': \nHas multiple items in their cart"), 200
    except:
        return jsonify(f"{str(customer_id)}: \nDoes NOT have multiple items in their cart"), 200
# handle retrieving customer names w/ active shopping cart
@user_bp.route('/active_carts', methods=['POST'])
def activeShoppingCart():
    data = request.get_json()
    customer_id = data['customer_id']
    try:
        customer_name = getUsersWithCart(customer_id=customer_id) # sql query for above note
        return jsonify(f"'{customer_name[0]}': has a shopping cart"), 200
    except:
        return jsonify("No Cart Associated with Custome Id"), 200
```