DailyHarbour

Aditya Aggarwal, Aniket Gupta, Medha Kashyap

April 2024

1 Introduction

Welcome to *DailyHarbour*, your ultimate retail companion. Crafted by Aditya Aggarwal (2022028), Aniket Gupta (2022073), and Medha Kashyap (2022292), *DailyHarbour* is an end-to-end retail app powered by HTML, CSS, JavaScript, Flask, and MySQL. Offering a seamless shopping experience, *DailyHarbour* lets you explore a diverse range of products with ease.

This guide serves the purpose of walking through each and every transaction that has been covered in the application. The transactions that we have covered in our application are as follows:

2 Conflicting Transactions

2.1 add_to_cart_db

2.1.1 Code

```
def add_to_cart_db(cursor, db, user_id, product_id, quantity):
    try:
        if db.in_transaction:
            db.commit()
        db.start_transaction()  # Starting the transaction
        insert_query = "INSERT INTO add_to_cart (user_id, product_id, number_of_units) VALUE
        cursor.execute(insert_query, (user_id, product_id, quantity))
        db.commit()  # Committing the transaction
    except Exception as e:
        db.rollback()  # Rolling back the transaction if an exception occurs
        print(e)
```

2.1.2 Explanation

This is a conflicting transaction since our application consists of a trigger that automatically deducts the the available units of that product in the database. If that particular quantity is being fiddled with by some other transaction at

the same time (for eg. another user adding the same quantity in their cart) this would lead to a write-write conflict.

2.2 delete_user

2.2.1 Code

```
@app.route('/delete_user', methods=["POST"])
def delete_user():
    try:
        data = request.get_json()
        id = data.get('user_id')
        db = get_database_connection()
        cursor = db.cursor()
        try:
            if (db.in_transaction):
                db.commit()

            db.start_transaction()
            query = "DELETE FROM user WHERE user_id = %s"
            cursor.execute(query, (id,))
            db.commit()
            return jsonify({'message': 'User deleted successfully'}), 200
        except Exception as e:
            print(e)
            db.rollback()
            return jsonify({'error': 'Database failed to delete user'}), 500
        finally:
            cursor.close()
            db.close()
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

2.2.2 Explanation

This can lead to a write-write conflict in case a user places an order, and at the same time the admin removes the user's entry from the user table in the database.

2.3 timer_expired

2.3.1 Code

```
@app.route('/timer_expired', methods=["POST"])
def timer_expired():
    user_id = session.get("user_id")
```

```
    if user_id:
        # Implement logic to empty the user's cart in the database
        try:
            db = get_database_connection()
            cursor = db.cursor()

            if db.in_transaction:
                db.commit()

            db.start_transaction()
            query = "DELETE FROM add_to_cart WHERE user_id = %s"
            cursor.execute(query, (user_id,))
            db.commit()
            return jsonify({'message': 'Cart emptied successfully'}), 200
        except Exception as e:
            print(e)
            db.rollback()
            return jsonify({'error': 'Database failed to empty cart'}), 500
    else:
        return jsonify({'error': 'User not authenticated'}), 401
```

2.3.2 Explanation

We have a trigger defined in our application which automatically updates back the available units of products in the database, since the products need to be added back once they are released from the user's cart who was holding them. If, at the same time, another user adds into their cart the same product, this would again lead to a write-write conflict.

2.4 send_address

2.4.1 Code

```
@app.route('/send_address', methods = ["POST"])
def place_order():
    user_id = session.get("user_id")

    if user_id:
        try:
            data = request.get_json()
            address = data.get('address')
            order_no = orderDetails(address, user_id)

            if order_no is not None:
                orderProducts(order_no[0], user_id)
```

```
            # empty the cart for the user once the order has been created
            db = get_database_connection()
            cursor = db.cursor()

            if db.in_transaction:
                db.commit()

            db.start_transaction()
            # conflicting transaction
            query = "DELETE FROM add_to_cart where user_id = %s"
            cursor.execute(query, (user_id,))
            cursor.fetchall()
            db.commit()

            cursor.close()
            db.close()

            return jsonify({'message': 'Fetched address successfully'}), 200
        except Exception as e:
            print(e)
            return jsonify({'error': 'Address not provided'}), 400
    else:
        return jsonify({'error': 'User not authenticated'}), 401
```

2.4.2 Explanation

2.5 update_product

2.5.1 Code

2.5.2 Explanation

This can again lead to a write-write conflict if the admin updates a product, and, at the same time, a user adds the product to their cart.

2.6 delete_product

2.6.1 Code

```
@app.route('/delete_product', methods=["POST"])
def delete_product():
    try:
        data = request.get_json()
        id = data.get('product_id')
        db = get_database_connection()
```

```
        cursor = db.cursor()
        try:
            query = "DELETE FROM product WHERE product_id = %s"
            cursor.execute(query, (id,))
            db.commit()
            return jsonify({'message': 'Product deleted successfully'}), 200
        except Exception as e:
            print(e)
            return jsonify({'error': 'Database failed to delete product'}), 500
        finally:
            cursor.close()
            db.close()
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

2.6.2 Explanation

This can also lead to a write-write conflict since if admin deletes a product, at the same time, a user may add that product to their cart.

3 Non-Conflicting Transactions

3.1 add_user

3.1.1 Code

```
@app.route('/add_user', methods=["POST"])
def add_user():
    try:
        data = request.get_json()
        id = data.get('user_id')  # Assuming the JSON key is 'user_id'
        first_name = data.get('first_name')
        last_name = data.get('last_name')
        middle_name = data.get('middle_name')
        gender = data.get('gender')
        dob = data.get('date_of_birth')
        phone = data.get('mobile_number')
        password = data.get('password_hash')

        db = get_database_connection()
        cursor = db.cursor()
        try:
            query = "INSERT INTO user (mobile_number, first_name, middle_name, last_name, pa
            cursor.execute(query, (phone, first_name, middle_name,
                                   last_name, password, gender, dob))
            db.commit()
```

```
            return jsonify({'message': 'User added successfully'}), 200
        except Exception as e:
            print(e)
            return jsonify({'error': 'Database failed to add user'}), 500
        finally:
            cursor.close()
            db.close()
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

3.1.2 Explanation

Through this method, the admin can get a user registered into the database. This is a non-conflicting transaction since it only makes an insertion into the user table without modifying any other values.

3.2 orderDetails

3.2.1 Code

```
def orderDetails(address, user_id):
    db = get_database_connection()
    cursor = db.cursor()

    try:
        order_value = get_order_value(cursor, user_id)
        number_of_products = get_number_of_products(cursor, user_id)

        if db.in_transaction:
            db.commit()

        db.start_transaction()

        # Insert order details into the database
        query = '''INSERT INTO order_details (user_id, address_name, order_date, total_numbe
        cursor.execute(query, (user_id, address, '2021-09-01', number_of_products, order_val
        db.commit()

        # obtain order_no of the order
        query = '''SELECT order_no FROM order_details WHERE user_id = %s AND order_date = %s
        cursor.execute(query, (user_id, '2021-09-01'))
        order_no = cursor.fetchall()


        return order_no[0] if order_no else None
```

```
    except Exception as e:
        print("Error in orderDetails:", e)
        return None

    finally:
        cursor.close()
        db.close()
```

3.2.2 Explanation

This method creates an entry which contains the details of the order that has been placed by a user. This is also a non-conflicting transaction since it simply makes an insertion in the order_details table, and does not modify any other values elsewhere.

3.3 orderProducts

3.3.1 Code

```
def orderProducts(order_no, user_id):
    # first get the products in the cart
    cart_data = get_cart_data2(user_id)
    db = get_database_connection()
    cursor = db.cursor()

    if db.in_transaction:
        db.commit()

    db.start_transaction()

    for product_id in cart_data:
        query = '''INSERT INTO order_products (order_no, product_id, number_of_units, price_
        cursor.execute(query, (order_no, product_id, cart_data[product_id]['number_of_units'
        db.commit()

    cursor.close()
    db.close()
```

3.3.2 Explanation

This method creates an entry which contains the mapping of an order to the products that it contains. This is also a non-conflicting transaction since it simply makes an insertion in the order_products table, and does not modify any other values elsewhere.

3.4 registerUser

3.4.1 Code

```
@app.route('/send_user_details', methods=["POST"])
def register_user():
    try:
        data = request.get_json()
        first_name = data.get('firstName')
        last_name = data.get('lastName')
        middle_name = data.get('middleName')
        gender = data.get('gender')
        dob = data.get('dob')
        phone = data.get('phone')
        password = data.get('password')

        db = get_database_connection()
        cursor = db.cursor()

        try:
            if (db.in_transaction):
                db.commit()

            # Start transaction
            db.start_transaction()

            query = "INSERT INTO user (mobile_number, first_name, middle_name, last_name, pa
            cursor.execute(query, (phone, first_name, middle_name, last_name, password, genc

            # Commit transaction
            db.commit()
            return jsonify({'message': 'User added successfully'}), 200
        except mysql.connector.Error as err:
            # Rollback transaction in case of errors
            db.rollback()
            return jsonify({'error': 'Database failed to add user'}), 500
        finally:
            cursor.close()
            db.close()
            # Ensure to end the transaction even in case of exceptions
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

3.4.2 Explanation

Users register themselves on the database through this method. This is a non-conflicting transaction since the user is not present yet in the database, and no

other transaction being processed at the same time can lead to any modifications.

3.5 api_address

3.5.1 Code

```
@app.route('/send_user_details', methods=["POST"])
def register_user():
    try:
        data = request.get_json()
        first_name = data.get('firstName')
        last_name = data.get('lastName')
        middle_name = data.get('middleName')
        gender = data.get('gender')
        dob = data.get('dob')
        phone = data.get('phone')
        password = data.get('password')

        db = get_database_connection()
        cursor = db.cursor()

        try:
            if (db.in_transaction):
                db.commit()

            # Start transaction
            db.start_transaction()

            query = "INSERT INTO user (mobile_number, first_name, middle_name, last_name, pa
            cursor.execute(query, (phone, first_name, middle_name, last_name, password, genc

            # Commit transaction
            db.commit()
            return jsonify({'message': 'User added successfully'}), 200
        except mysql.connector.Error as err:
            # Rollback transaction in case of errors
            db.rollback()
            return jsonify({'error': 'Database failed to add user'}), 500
        finally:
            cursor.close()
            db.close()
            # Ensure to end the transaction even in case of exceptions
    except Exception as e:
        return jsonify({'error': str(e)}), 500
```

3.5.2 Explanation

Through this method, a user can create a new shipping address for themselves. Since it is not present yet in the database, it cannot be modified by another transaction being processed on that address at the same time. Thus, this is a non-conflicting transaction.

3.6 add_product

3.6.1 Code

```
@app.route('/add_product', methods=["POST"])
def add_product():
    try:
        data = request.get_json()
        print(data)
        product_name = data.get('product_name')
        unit_of_measure = data.get('unit_of_measure')
        selling_price = data.get('selling_price')
        available_units = data.get('avail_units')
        category_id = data.get('category_id')
        mrp = data.get('mrp')
        quantity_per_unit = data.get('quantity_per_unit')

        db = get_database_connection()
        cursor = db.cursor()

        if db.in_transaction:
            db.commit()

        db.start_transaction()

        query = '''INSERT INTO product (product_name, unit_of_measure, selling_price, availa

        cursor.execute(query, (product_name, unit_of_measure, selling_price, available_units
        db.commit()

        cursor.close()
        db.close()

        return jsonify({'message': 'Product added successfully'}), 200
    except Exception as e:
        print(e)
        return jsonify({'error': 'Product not added'}), 500
```

3.6.2 Explanation

Through this method, the admin of the database can enter a new product in the database. We know that the product cannot be modified at the same time while the admin is adding it, since it is not present in the database yet. Thus, this is a non-conflicting transaction.