PSG College of Technology

Akshayakumar N.G

21z206 (BE CSE)

In this documentation, we have explained each part of the code, detailing how the function works, from the database connection to the execution of the main function.

Database Setup and Connection:

1.db_connection()

Code:

```
def db_connection():
    conn = sqlite3.connect(DB_NAME)
    conn.row_factory = sqlite3.Row
    return conn
```

One-line description : Establishes a connection to the SQLite database and returns the connection object.

2. initialize_db()

Code:

```
def initialize_db():
    conn = db_connection()
    cursor = conn.cursor()
    cursor.executescript("""
    CREATE TABLE IF NOT EXISTS seasonal_flavors (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        description TEXT,
        price REAL NOT NULL,
        available INTEGER DEFAULT 1,
        season TEXT
    );
    ...
    """)
    conn.commit()
    conn.close()
```

One-line description : Initializes the database by creating necessary tables if they do not exist.

Flavor Management

3. add_flavor()

Code:

```
def add_flavor(name, description, price, season):
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO seasonal_flavors (name, description, price, season)
        VALUES (?, ?, ?, ?)
    """, (name, description, price, season))
    conn.commit()
    conn.close()
    print(f"Flavor '{name}' added successfully.")
```

One-line description : Adds a new flavor with details to the seasonal_flavors table.

4. search_flavors():

Code:

```
def search_flavors(search_term=None, filter_season=None):
    conn = db_connection()
    cursor = conn.cursor()
    query = "SELECT * FROM seasonal_flavors WHERE 1"
    params = []
    if search_term:
        query += " AND name LIKE ?"
        params.append(f"%{search_term}%")
    if filter_season:
        query += " AND season = ?"
        params.append(filter_season)
    cursor.execute(query, params)
    rows = cursor.fetchall()
```

One-line description : Searches for flavors by name or filters them by season.

5.Ingredient Management

add_ingredient()

Code:

```
def add_ingredient(name, quantity, unit):
    conn = db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO ingredients (name, quantity, unit)
            VALUES (?, ?, ?)
        """, (name, quantity, unit))
        conn.commit()
        print(f"Ingredient '{name}' added successfully.")
    except sqlite3.IntegrityError:
        print(f"Ingredient '{name}' already exists.")
    conn.close()
```

One-line description : Adds a new ingredient and ensures uniqueness.

6.update_ingredient_quantity()

Code:

```
def update_ingredient_quantity(name, quantity, operation):
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT quantity FROM ingredients WHERE name = ?", (name,))
    row = cursor.fetchone()
    if row:
        current_quantity = row['quantity']
        if operation == 'add':
            new_quantity = current_quantity + quantity
        elif operation == 'subtract':
            if current_quantity >= quantity:
                new_quantity = current_quantity - quantity
```

One-line description : Updates the quantity of an ingredient based on the operation (add or subtract).

7. Customer Suggestions

add_customer_suggestion()

Code:

```
def add_customer_suggestion(customer_name, flavor_name, allergy_concerns=None):
    suggestion_date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO customer_suggestions (customer_name, flavor_name, suggestion_date, allergy_concerns)
        VALUES (?, ?, ?, ?)
    """, (customer_name, flavor_name, suggestion_date, allergy_concerns))
    conn.commit()
    conn.close()
    print(f"Suggestion from {customer_name} for flavor '{flavor_name}' added successfully.")
```

One-line description : Records a customer's flavor suggestion with optional allergy concerns.

8.view_customer_suggestions()

Code:

```
def view_customer_suggestions():
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM customer_suggestions")
    rows = cursor.fetchall()
    ...
```

One-line description : Displays all recorded customer suggestions.

9.Allergen Management

add_allergen()

Code:

```
def add_allergen(name):
    conn = db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO allergens (name)
            VALUES (?)
        """, (name,))
        conn.commit()
        print(f"Allergen '{name}' added successfully.")
    except sqlite3.IntegrityError:
        print(f"Allergen '{name}' already exists.")
    conn.close()
```

One-line description : Adds a new allergen and ensures uniqueness.

10.view_allergens()

Code:

```
def view_allergens():
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM allergens")
    rows = cursor.fetchall()
    ...
```

One-line description : Displays a list of allergens.

Shopping Cart Management

11. add_to_cart()

Code:

```
def add_to_cart():
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM seasonal_flavors WHERE available = 1")
    rows = cursor.fetchall()
    if rows:
        flavor_id = int(input("Enter the Flavor ID you want to add to the cart: "))
        quantity = int(input("Enter the quantity: "))
        ...
```

One-line description : Prompts the user to add a flavor to the cart.

12. view_cart()

Code:

```
def view_cart():
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        SELECT c.id, s.name, c.quantity, s.price, (c.quantity * s.price) AS total_price
        FROM cart c
        JOIN seasonal_flavors s ON c.flavor_id = s.id
    """)
    rows = cursor.fetchall()
    ...
```

One-line description : Displays cart details with a grand total.

13.remove_from_cart()

Code:

```
def remove_from_cart(cart_id):
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM cart WHERE id = ?", (cart_id,))
    conn.commit()
    ...
```

One-line description : Removes an item from the cart using its ID.

Main Menu

14.main_menu()

```
def main_menu():
    print("\nWelcome to the Ice Cream Parlor Management System!")
    user_role = input("Are you an 'Owner' or a 'Customer'? ").strip().lower()
    if user_role not in ['owner', 'customer']:
        print("Invalid role. Please restart and choose either 'Owner' or 'Customer'.")
        return
    ...
```

One-line description : The main entry point, displaying role-specific options based on whether the user is an owner or a customer.

Full Code:

```
import sqlite3
import datetime


DB_NAME = "ice_cream_parlor.db"


# Function to establish a connection to the database
def db_connection():
    conn = sqlite3.connect(DB_NAME)
    conn.row_factory = sqlite3.Row  # Allows access by column name instead of index
    return conn
```

```
# Function to initialize the database with tables
def initialize_db():
    conn = db_connection()
    cursor = conn.cursor()
    # Creating tables for seasonal_flavors, ingredients, customer_suggestions, allergens, and cart
    cursor.executescript("""
    CREATE TABLE IF NOT EXISTS seasonal_flavors (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        description TEXT,
        price REAL NOT NULL,
        available INTEGER DEFAULT 1,
        season TEXT
    );

    CREATE TABLE IF NOT EXISTS ingredients (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL UNIQUE,
        quantity REAL NOT NULL,
        unit TEXT NOT NULL
    );

    CREATE TABLE IF NOT EXISTS customer_suggestions (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        customer_name TEXT NOT NULL,
        flavor_name TEXT NOT NULL,
        suggestion_date TEXT NOT NULL,
        allergy_concerns TEXT
    );

    CREATE TABLE IF NOT EXISTS allergens (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
```

```
        name TEXT NOT NULL UNIQUE
    );

    CREATE TABLE IF NOT EXISTS cart (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        flavor_id INTEGER NOT NULL,
        quantity INTEGER NOT NULL,
        FOREIGN KEY (flavor_id) REFERENCES seasonal_flavors (id)
    );
    """)
    conn.commit()
    conn.close()

initialize_db()

# Function to add a new flavor to the seasonal_flavors table
def add_flavor(name, description, price, season):
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO seasonal_flavors (name, description, price, season)
        VALUES (?, ?, ?, ?)
    """, (name, description, price, season))
    conn.commit()
    conn.close()
    print(f"Flavor '{name}' added successfully.")

# Function to search and filter flavors based on search term and season
def search_flavors(search_term=None, filter_season=None):
    conn = db_connection()
    cursor = conn.cursor()

    query = "SELECT * FROM seasonal_flavors WHERE 1"
```

```
    params = []

    if search_term:
        query += " AND name LIKE ?"
        params.append(f"%{search_term}%")
    if filter_season:
        query += " AND season = ?"
        params.append(filter_season)

    cursor.execute(query, params)
    rows = cursor.fetchall()

    if rows:
        for row in rows:
            print(f"Name: {row['name']}, Description: {row['description']}, Price: {row['price']}, Season:
{row['season']}")
    else:
        print("No flavors found.")

    conn.close()

# Function to add a new ingredient to the ingredients table
def add_ingredient(name, quantity, unit):
    conn = db_connection()
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO ingredients (name, quantity, unit)
            VALUES (?, ?, ?)
        """, (name, quantity, unit))
        conn.commit()
        print(f"Ingredient '{name}' added successfully.")
    except sqlite3.IntegrityError:
```

```
        print(f"Ingredient '{name}' already exists.")
    conn.close()

# Function to add a customer suggestion to the customer_suggestions table
def add_customer_suggestion(customer_name, flavor_name, allergy_concerns=None):
    suggestion_date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("""
        INSERT INTO customer_suggestions (customer_name, flavor_name, suggestion_date, allergy_concerns)
        VALUES (?, ?, ?, ?)
    """, (customer_name, flavor_name, suggestion_date, allergy_concerns))
    conn.commit()
    conn.close()
    print(f"Suggestion from {customer_name} for flavor '{flavor_name}' added successfully.")

# Function to view all customer suggestions
def view_customer_suggestions():
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM customer_suggestions")
    rows = cursor.fetchall()
    if rows:
        for row in rows:
            print(f"Customer: {row['customer_name']}, Flavor: {row['flavor_name']}, Date:
{row['suggestion_date']}, Allergy Concerns: {row['allergy_concerns']}")
    else:
        print("No customer suggestions found.")
    conn.close()

# Function to add an allergen to the allergens table
def add_allergen(name):
    conn = db_connection()
```

```
    cursor = conn.cursor()
    try:
        cursor.execute("""
            INSERT INTO allergens (name)
            VALUES (?)
        """, (name,))
        conn.commit()
        print(f"Allergen '{name}' added successfully.")
    except sqlite3.IntegrityError:
        print(f"Allergen '{name}' already exists.")
    conn.close()

# Function to view all allergens in the allergens table
def view_allergens():
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM allergens")
    rows = cursor.fetchall()
    if rows:
        for row in rows:
            print(f"Allergen: {row['name']}")
    else:
        print("No allergens found.")
    conn.close()

# Function to add a flavor to the cart
def add_to_cart():
    conn = db_connection()
    cursor = conn.cursor()

    # Display all available flavors with details
    cursor.execute("SELECT * FROM seasonal_flavors WHERE available = 1")
    rows = cursor.fetchall()
```

```
    if rows:
        print("\nAvailable Flavors:")
        for row in rows:
            print(f"ID: {row['id']}\nFlavor: {row['name']}\nDescription: {row['description']}\nPrice:
${row['price']:.2f}\n")

        # Prompt the user for flavor ID and quantity
        flavor_id = int(input("Enter the Flavor ID you want to add to the cart: "))
        quantity = int(input("Enter the quantity: "))
        try:
            # Verify the selected flavor exists and is available
            cursor.execute("SELECT * FROM seasonal_flavors WHERE id = ? AND available = 1", (flavor_id,))
            flavor = cursor.fetchone()
            if flavor:
                # Add the flavor to the cart
                cursor.execute("""
                    INSERT INTO cart (flavor_id, quantity)
                    VALUES (?, ?)
                """, (flavor_id, quantity))
                conn.commit()
                print(f"Added {quantity} of '{flavor['name']}' to the cart.")
            else:
                print("Invalid Flavor ID or flavor is not available.")
        except ValueError:
            print("Invalid input. Please enter valid numbers.")
    else:
        print("No flavors are currently available.")

    conn.close()

# Function to view the cart with total cost
def view_cart():
```

```
    conn = db_connection()
    cursor = conn.cursor()

    # Query to get cart details including price and quantity
    cursor.execute("""
        SELECT c.id, s.name, c.quantity, s.price, (c.quantity * s.price) AS total_price
        FROM cart c
        JOIN seasonal_flavors s ON c.flavor_id = s.id
    """)
    rows = cursor.fetchall()
    if rows:
        grand_total = 0  # To keep track of the total cost of the cart
        print("\nCart Details:")
        for row in rows:
            print(f"Cart ID: {row['id']}, Flavor: {row['name']}, Quantity: {row['quantity']}, Price per unit: ${row['price']:.2f}, Total: ${row['total_price']:.2f}")
            grand_total += row['total_price']  # Accumulate the total price

        print(f"\nGrand Total: ${grand_total:.2f}")
    else:
        print("Your cart is empty.")

    conn.close()

# Function to remove an item from the cart by Cart ID
def remove_from_cart(cart_id):
    conn = db_connection()
    cursor = conn.cursor()
    cursor.execute("DELETE FROM cart WHERE id = ?", (cart_id,))
    conn.commit()
    if cursor.rowcount > 0:
        print(f"Item with Cart ID {cart_id} removed from cart.")
    else:
```

```
        print(f"Item with Cart ID {cart_id} not found.")
    conn.close()

# Function to update the ingredient quantity (add or subtract)
def update_ingredient_quantity(name, quantity, operation):
    conn = db_connection()
    cursor = conn.cursor()

    cursor.execute("SELECT quantity FROM ingredients WHERE name = ?", (name,))
    row = cursor.fetchone()

    if row:
        current_quantity = row['quantity']

        if operation == 'add':
            new_quantity = current_quantity + quantity
        elif operation == 'subtract':
            if current_quantity >= quantity:
                new_quantity = current_quantity - quantity
            else:
                print(f"Cannot subtract {quantity} from {name}, insufficient stock.")
                conn.close()
                return
        else:
            print("Invalid operation. Use 'add' or 'subtract'.")
            conn.close()
            return

        cursor.execute("""
            UPDATE ingredients
            SET quantity = ?
            WHERE name = ?
        """, (new_quantity, name))
```

```
            conn.commit()
            print(f"Updated '{name}' quantity to {new_quantity}.")
        else:
            print(f"Ingredient '{name}' does not exist in the database.")

        conn.close()

# Function to display the main menu and allow user to select options based on their role
def main_menu():
    print("\nWelcome to the Ice Cream Parlor Management System!")
    user_role = input("Are you an 'Owner' or a 'Customer'? ").strip().lower()

    if user_role not in ['owner', 'customer']:
        print("Invalid role. Please restart and choose either 'Owner' or 'Customer'.")
        return

    while True:
        print("\nMain Menu:")

        if user_role == 'owner':
            print("1. Add New Flavor")
            print("2. Search & Filter Flavors")
            print("3. Add Ingredient")
            print("4. Update Ingredient Quantity")
            print("5. View Customer Suggestions")
            print("6. Add Allergen")
            print("7. View Allergen List")
            print("8. Exit")

            choice = input("Enter your choice (1-8): ").strip()

            if choice == '1':
                name = input("Enter flavor name: ").strip()
```

```
            description = input("Enter flavor description: ").strip()
            price = float(input("Enter price: ").strip())
            season = input("Enter season: ").strip()
            add_flavor(name, description, price, season)
        elif choice == '2':
            search_term = input("Enter search term (or press Enter to skip): ").strip()
            filter_season = input("Enter season to filter by (or press Enter to skip): ").strip()
            search_flavors(search_term, filter_season)
        elif choice == '3':
            name = input("Enter ingredient name: ").strip()
            quantity = float(input("Enter quantity: ").strip())
            unit = input("Enter unit (e.g., kg, liters): ").strip()
            add_ingredient(name, quantity, unit)
        elif choice == '4':
            name = input("Enter ingredient name to update: ").strip()
            quantity = float(input("Enter quantity to add/subtract: ").strip())
            operation = input("Enter operation ('add' or 'subtract'): ").strip().lower()
            update_ingredient_quantity(name, quantity, operation)
        elif choice == '5':
            view_customer_suggestions()
        elif choice == '6':
            allergen_name = input("Enter allergen name to add: ").strip()
            add_allergen(allergen_name)
        elif choice == '7':
            view_allergens()
        elif choice == '8':
            print("Exiting system. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")

    elif user_role == 'customer':
        print("1. Search & Filter Flavors")
```

```
        print("2. Add Customer Suggestion")
        print("3. Add to Cart")
        print("4. View Cart")
        print("5. Remove from Cart")
        print("6. Exit")

        choice = input("Enter your choice (1-6): ").strip()

        if choice == '1':
            search_term = input("Enter search term (or press Enter to skip): ").strip()
            filter_season = input("Enter season to filter by (or press Enter to skip): ").strip()
            search_flavors(search_term, filter_season)
        elif choice == '2':
            customer_name = input("Enter your name: ").strip()
            flavor_name = input("Enter flavor name suggestion: ").strip()
            allergy_concerns = input("Enter any allergy concerns (or press Enter to skip): ").strip()
            add_customer_suggestion(customer_name, flavor_name, allergy_concerns)
        elif choice == '3':
            add_to_cart()
        elif choice == '4':
            view_cart()
        elif choice == '5':
            cart_id = int(input("Enter Cart ID to remove: ").strip())
            remove_from_cart(cart_id)

        elif choice == '6':
            print("Exiting system. Goodbye!")
            break
        else:
            print("Invalid choice. Please try again.")
main_menu()
```

SQL

SQL Query or ORM abstraction Implementation

1. Creating Tables

These SQL queries create the tables needed for the application if they don’t already exist:

CREATE TABLE for seasonal_flavors:

sql

```
CREATE TABLE IF NOT EXISTS seasonal_flavors (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    description TEXT,
    price REAL NOT NULL,
    available INTEGER DEFAULT 1,
    season TEXT
);
```

One-line description : Creates a table for ice cream flavors with columns for the flavor's name, description, price, availability, and season.

CREATE TABLE for ingredients:

sql

```
CREATE TABLE IF NOT EXISTS ingredients (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    quantity REAL NOT NULL,
    unit TEXT NOT NULL
);
```

One-line description : Creates a table for ingredients with columns for the ingredient name, quantity, and unit of measurement.

CREATE TABLE for customer_suggestions:

sql

```
CREATE TABLE IF NOT EXISTS customer_suggestions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    customer_name TEXT NOT NULL,
    flavor_name TEXT NOT NULL,
    suggestion_date TEXT NOT NULL,
    allergy_concerns TEXT
);
```

One-line description : Creates a table for customer suggestions, storing the customer's name, flavor suggestion, date, and any allergy concerns.

CREATE TABLE for allergens:

sql

```
CREATE TABLE IF NOT EXISTS allergens (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE
);
```

One-line description : Creates a table to store allergens, with each allergen's name.

CREATE TABLE for cart:

sql

```
CREATE TABLE IF NOT EXISTS cart (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    flavor_id INTEGER NOT NULL,
    quantity INTEGER NOT NULL,
    FOREIGN KEY (flavor_id) REFERENCES seasonal_flavors (id)
);
```

One-line description : Creates a table to store items in a shopping cart, linking each item to a flavor in the seasonal_flavors table.

2. Inserting Data

These SQL queries add new data to the tables.

Add Flavor:

sql

INSERT INTO seasonal_flavors (name, description, price, season)

VALUES (?, ?, ?, ?)

One-line description : Adds a new flavor to the seasonal_flavors table.

Add Ingredient:

sql

INSERT INTO ingredients (name, quantity, unit)

VALUES (?, ?, ?)

One-line description : Adds a new ingredient to the ingredients table.

Add Customer Suggestion:

sql

INSERT INTO customer_suggestions (customer_name, flavor_name, suggestion_date, allergy_concerns)

VALUES (?, ?, ?, ?)

One-line description : Adds a customer's suggestion to the customer_suggestions table.

Add Allergen:

sql

INSERT INTO allergens (name)

VALUES (?)

One-line description : Adds a new allergen to the allergens table.

Add to Cart:

sql

INSERT INTO cart (flavor_id, quantity)

VALUES (?, ?)

One-line description : Adds an item to the shopping cart.

3. Selecting Data

These SQL queries retrieve data from the tables.

Get All Flavors:

Sql

SELECT * FROM seasonal_flavors WHERE 1

One-line description : Retrieves all flavors from the seasonal_flavors table.

Search Flavors by Name:

sql

SELECT * FROM seasonal_flavors WHERE name LIKE ?

One-line description : Retrieves flavors whose name contains a specific search term.

Get Flavors by Season:

sql

SELECT * FROM seasonal_flavors WHERE season = ?

One-line description : Retrieves all flavors for a specific season.

Get All Customer Suggestions:

sql

SELECT * FROM customer_suggestions

One-line description : Retrieves all customer suggestions.

Get All Allergens:

sql

SELECT * FROM allergens

One-line description : Retrieves all allergens.

Get Available Flavors:

sql

SELECT * FROM seasonal_flavors WHERE available = 1

One-line description : Retrieves all available flavors (where the available column is 1).

Get Ingredient Quantity:

sql

SELECT quantity FROM ingredients WHERE name = ?

One-line description : Retrieves the quantity of a specific ingredient.

View Cart with Flavor Details:

sql

SELECT c.id, s.name, c.quantity, s.price, (c.quantity * s.price) AS total_price

FROM cart c

JOIN seasonal_flavors s ON c.flavor_id = s.id

One-line description : Retrieves details of all items in the cart, including the flavor name, price, quantity, and total price for each item.

4. Updating Data

These queries update existing data in the tables.

Update Ingredient Quantity:

sql

UPDATE ingredients

SET quantity = ?

WHERE name = ?

One-line description : Updates the quantity of a specific ingredient.

5. Deleting Data

This query removes data from the cart.

Remove Item from Cart:

sql

DELETE FROM cart WHERE id = ?

One-line description : Deletes an item from the cart by its id.