```
#include "DBConnection.h"
#include <iostream>
#include <vector>
#include <string>
#include <iomanip>
#include <map>
#include <regex>
#include <iomanip>
#include <sstream>
#include <thread>
#include <chrono>
#include <conio.h>
#include <cmath>
#include <fstream>

using namespace std;

// Color codes for better UI
// Reset
#define RESET "\033[0m"

#define SUCCESS "\033[1;32m" // Bold Green
#define ERROR "\033[1;31m"   // Bold Red
#define PROMPT "\033[1;33m"  // Bold Yellow

// Standard Colors
#define RED "\033[31m"
#define GREEN "\033[32m"
#define YELLOW "\033[33m"
#define BLUE "\033[34m"
#define CYAN "\033[36m"

// Bright Colors
#define BRIGHT_RED "\033[91m"
#define BRIGHT_GREEN "\033[92m"
#define BRIGHT_YELLOW "\033[93m"
#define BRIGHT_BLUE "\033[94m"

#define GREEN_BG   "\033[42m"      // Green background

struct Cart {
   map<int, int> items; // Menu item ID -> Quantity

   void addItem(int itemID, int quantity, DBConnection& db) {
      // Add the item to the cart
      items[itemID] += quantity;

      // Fetch item details from the database
      db.prepareStatement("SELECT item_name, item_price FROM menu_item WHERE menu_item_id
= ?");
      db.stmt->setInt(1, itemID);
      db.QueryResult();

      // Check if the item exists in the database
      if (db.res->next()) {
         string itemName = db.res->getString("item_name");
         double itemPrice = db.res->getDouble("item_price");
```

```
        // Display the item details
        cout << BRIGHT_GREEN << "\nItem successfully added to the cart!\n" << RESET;
        cout << "-----------------------------------\n";
        cout << "Item ID    : " << itemID << "\n";
        cout << "Item Name  : " << itemName << "\n";
        cout << "Price (RM) : " << fixed << setprecision(2) << itemPrice << "\n";
        cout << "Quantity   : " << quantity << "\n";
        cout << "-----------------------------------\n";
      }
      else {
        // Handle the case where the item ID doesn't exist
        cout << "Error: Item ID " << itemID << " not found in the database.\n";
      }

      // Reminder for the user
      cout << CYAN << "**After you finish adding to the cart,\n" << RESET;
      cout << CYAN << "don't forget to confirm your order! Otherwise, we will not receive your
order.\n\n" << RESET;
    }

    void removeItem(int itemID, const string& quantityInput, DBConnection& db) {
    auto it = items.find(itemID);
    if (it != items.end()) {
      if (quantityInput == "all") {
        // Remove all quantities of the item
        items.erase(it);
        system("cls");  // Clear screen before showing updated cart
        cout << CYAN << "\n============================================================\n"
<< RESET;
        cout << CYAN << "                Remove Item from Cart               \n" << RESET;
        cout << CYAN << "============================================================\n" <<
RESET;
        cout << BRIGHT_GREEN << "\nItem ID " << itemID << " has been completely removed from
your cart.\n" << RESET;
      } else {
        // Parse quantity
        int quantity;
        try {
          quantity = stoi(quantityInput);
        } catch (...) {
          system("cls");
          cout << BRIGHT_RED << "\nInvalid input. Please enter a valid quantity or '0' to go back.\n" <<
RESET;
          system("pause");
          return;
        }

        if (quantity == 0) {
          // Return to the menu
          system("cls");
          cout << "\nReturning to the previous menu...\n";
          system("pause");
          return;
        }

        if (quantity > it->second) {
```

```
            // Quantity to remove exceeds available quantity
            system("cls");
            cout << CYAN <<
"\n============================================================\n" << RESET;
            cout << CYAN << "                Remove Item from Cart                \n" << RESET;
            cout << CYAN << "============================================================\n"
<< RESET;
            cout << BRIGHT_RED << "\nError: Not enough items in your cart to remove. "
                << "You have " << it->second << " of Item ID " << itemID << " in your cart.\n" << RESET;
         } else {
            // Reduce the quantity
            it->second -= quantity;
            system("cls");
            cout << CYAN <<
"\n============================================================\n" << RESET;
            cout << CYAN <<   "                Remove Item from Cart                \n" << RESET;
            cout << CYAN <<
"============================================================\n" << RESET;
            cout << BRIGHT_GREEN << "\nReduced quantity of Item ID " << itemID << " by " << quantity
<< ".\n" << RESET;
            cout << BRIGHT_GREEN << "Remaining quantity: " << it->second << ".\n" << RESET;

            // Remove the item if the quantity becomes zero
            if (it->second == 0) {
               items.erase(it);
               cout << BRIGHT_GREEN << "\nAll quantities of Item ID " << itemID << " have been removed
from your cart.\n" << RESET;
            }
         }
      }

      // Display updated cart
      cout << YELLOW << "\nYour updated cart:\n" << RESET;
      viewCart(db);
   } else {
      // Item not found
      system("cls");
      cout << CYAN << "\n============================================================\n" <<
RESET;
      cout << CYAN << "                Remove Item from Cart                \n" << RESET;
      cout << CYAN << "============================================================\n" <<
RESET;
      cout << BRIGHT_RED << "\nItem ID " << itemID << " not found in your cart.\n\n" << RESET;
      viewCart(db);
   }
}

   void viewCart(DBConnection& db) {
      if (items.empty()) {
         cout << RED << "Your cart is empty.\n" << RESET;
         return;
      }

      double total = 0.0;

      cout << YELLOW << "\nCurrent Cart:\n" << RESET;
```

```
      cout << CYAN << "+--------------------+----------------------------------+-----------+---------------+-----------
-+\n" << RESET;
      cout << GREEN << "| Menu_Item_ID       | Item_Name                        | Quantity  | Unit_Price    |
Subtotal    |\n" << RESET;
      cout << CYAN << "+--------------------+----------------------------------+-----------+---------------+-----------
-+\n" << RESET;

      for (const auto& item : items) {
         int itemID = item.first;
         int quantity = item.second;

         // Fetch the item name and price from the database
         db.prepareStatement("SELECT item_name, item_price FROM menu_item WHERE
menu_item_id = ?");
         db.stmt->setInt(1, itemID);
         db.QueryResult();

         string itemName = "Unknown";
         double itemPrice = 0.0;
         if (db.res->next()) {
            itemName = db.res->getString("item_name");
            itemPrice = db.res->getDouble("item_price");
         }

         double subtotal = itemPrice * quantity;
         total += subtotal;

         cout << "| " << setw(20) << left << itemID
            << "| " << setw(35) << left << itemName
            << "| " << setw(11) << quantity
            << "| " << fixed << setprecision(2) << setw(15) << itemPrice
            << "| " << fixed << setprecision(2) << setw(11) << subtotal << " |\n";
      }

      cout << CYAN << "+--------------------+----------------------------------+-----------+---------------+-----------
-+\n" << RESET;
      cout << GREEN << "Total: " << RESET << fixed << setprecision(2) << total << "\n";
   }



   void clearCart() {
      items.clear();
      cout << "Cart cleared.\n";
   }

   bool isEmpty() const {
      return items.empty();
   }
};
struct Recommendation {
   int menuItemID;
   int totalSales;
};

// Function prototypes
void loginRegisterMenu();
```

```
bool loginUser(DBConnection& db, int choice_login, string email, string password, int& customerID);
void registerCustomer(DBConnection& db);
void removeCustomer(DBConnection& db);
void customerMenu(int customerID);
void staffMenu(bool isManager, int staffID);
void viewMenu(DBConnection& db);
void viewRecommendations(DBConnection& db);
void placeOrder(DBConnection& db, int customerID, vector<int> menuItems, vector<int> quantities);
void viewOrderHistory(DBConnection& db, int customerID);
void addMenu(DBConnection& db);
void updateMenu(DBConnection& db);
void deleteMenu(DBConnection& db);
void generateSalesReport(DBConnection& db, const string& timeframe, const string& date);
int getOrCreateUnpaidOrder(DBConnection& db, int customerID);
void payOrder(DBConnection& db);
void generateSalesBarGraph(DBConnection& db);
void registerMembership(DBConnection& db);
void removeMembership(DBConnection& db);
double calculateDiscount(double totalAmount, bool isMember, const string& birthday);
void viewProfile(DBConnection& db, int customerID);
void managerMenu();
void registerStaff(DBConnection& db);
void updateStaff(DBConnection& db);
void viewStaffList(DBConnection& db);
void viewCustomerList(DBConnection& db);
void updateCustomerInformation(DBConnection& db);
bool rateFood(DBConnection& db, int customerID, int menuItemID, int rating);
string getMonthName(int month);
void displayMessageBox(const string& message, const string& color);
void loadingAnimation();
double calculateBirthdayDiscount(double totalAmount, const string& birthday);
void removeStaff(DBConnection& db);
void removeCustomer(DBConnection& db);
void deleteSpecificItem(DBConnection& db);
void deleteEntireCategory(DBConnection& db);

// main
int main() {
   loginRegisterMenu();

   return 0;
}

//loginMenu
void loginRegisterMenu() {
   DBConnection db;
   string email, password;
   int customerID = -1, staffID = -1;

   while (true) {
      system("cls"); // Clear the screen (for Windows, use "clear" on Unix)
      cout << CYAN << "+-----------------------------------------------+\n" << RESET;
      cout << CYAN << "|          WELCOME TO SUSHI SYSTEM          |\n" << RESET;
      cout << CYAN << "+-----------------------------------------------+\n" << RESET;
      cout << BRIGHT_GREEN << "| 1. Login                                  |\n" << RESET;
      cout << BRIGHT_GREEN << "| 2. Register                               |\n" << RESET;
      cout << BRIGHT_RED << "| 0. Exit                                   |\n" << RESET;
```

```
    cout << CYAN << "+---------------------------------------------+\n" << RESET;
    cout << YELLOW << "Enter your choice: " << RESET;
    int choice;
    cin >> choice;

    if (cin.fail() || choice < 0 || choice > 2) { // Validate input
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << ERROR << " Invalid choice! Please enter 1, 2, or 0.\n" << RESET;
        system("pause");
        continue;
    }

    switch (choice) {
    case 1: { // Login
        system("cls");
        cout << CYAN << "+-------------------------------------------------------------------------------+\n" << RESET;
        cout << CYAN << "|                                LOGIN                                          |\n" << RESET;
        cout << CYAN << "+-------------------------------------------------------------------------------+\n" << RESET;
        cout << YELLOW << "Note: Type '0' at any time to cancel and return to the main menu.\n\n" <<
RESET;

        // Prompt for email
        cout << BRIGHT_GREEN << "Enter your email: " << RESET;
        cin >> email;
        if (email == "0") { // Allow user to cancel during email entry
            cout << YELLOW << "Cancelling login...\n" << RESET;
            system("pause");
            break;
        }

        // Prompt for password
        cout << BRIGHT_GREEN << "Enter your password: " << RESET;
        password = "";
        char ch;
        while ((ch = _getch()) != 13) { // 13 is the Enter key
            if (ch == 8) { // Backspace
                if (!password.empty()) {
                    password.pop_back(); // Remove the last character from password
                    cout << "\b \b"; // Erase the last asterisk
                }
            }
            else {
                password += ch;
                cout << "*"; // Display asterisks for password
            }
        }
        cout << "\n";

        if (password == "0") { // Allow user to cancel during password entry
            cout << YELLOW << "Cancelling login...\n" << RESET;
            system("pause");
            break;
        }

        cout << YELLOW << "\nProcessing, please wait..." << RESET;
        loadingAnimation();
```

```
        for (int i = 0; i < 3; i++) {
            this_thread::sleep_for(chrono::milliseconds(500));
            cout << ".";
        }
        cout << "\n";

        // Check Customer Table
        db.prepareStatement("SELECT customer_id, customer_name FROM customer WHERE
customer_email = ? AND customer_password = ?");
        db.stmt->setString(1, email);
        db.stmt->setString(2, password);
        db.QueryResult();

        if (db.res->next()) {
            customerID = db.res->getInt("customer_id");
            string customerName = db.res->getString("customer_name");
            system("cls");

            // Construct the welcome message
            stringstream welcomeMessage;
            welcomeMessage << "Login successful! Welcome Customer, " << customerName << "!";
            displayMessageBox(welcomeMessage.str(), BRIGHT_GREEN); // Reusable function for
message boxes
            system("pause");
            customerMenu(customerID);
            break;
        }

        // Check Staff Table
        db.prepareStatement("SELECT staff_id, staff_name, staff_role FROM staff WHERE staff_email
= ? AND staff_password = ?");
        db.stmt->setString(1, email);
        db.stmt->setString(2, password);
        db.QueryResult();

        if (db.res->next()) {
            staffID = db.res->getInt("staff_id");
            string staffName = db.res->getString("staff_name");
            string staffRole = db.res->getString("staff_role");
            system("cls");

            // Construct the welcome message
            stringstream welcomeMessage;
            welcomeMessage << "Login successful! Welcome " << staffRole << ", " << staffName << "!";
            displayMessageBox(welcomeMessage.str(), BRIGHT_GREEN);
            system("pause");
            staffMenu(staffRole == "manager", staffID);
            break;
        }

        displayMessageBox("Invalid email or password. Please try again.", RED);
        system("pause");
        break;
    }

    case 2: // Register
        registerCustomer(db);
```

```
            break;

        case 0: // Exit
            displayMessageBox("Thank you for using the Sushi System. Goodbye!", GREEN);
            exit(0);

        default:
            displayMessageBox("Invalid choice. Please try again.", RED);
            system("pause");
        }
    }
}

// Customer menu
void customerMenu(int customerID) {
    Cart cart;
    DBConnection db;
    int choice_customer_menu;

    // Fetch customer name for personalization
    string customerName;
    db.prepareStatement("SELECT customer_name FROM customer WHERE customer_id = ?");
    db.stmt->setInt(1, customerID);
    db.QueryResult();
    if (db.res->next()) {
        customerName = db.res->getString("customer_name");
    }
    else {
        customerName = "Unknown"; // Fallback if customer ID is invalid
    }

    do {
        system("cls"); // Clear screen for a refreshed menu view

        // Header with "Welcome customer"
        stringstream header;
        header << "Welcome customer, " << customerName << " (ID: " << customerID << ")";
        string headerStr = header.str();
        int totalWidth = 48; // Total width of the box, including borders
        int contentWidth = totalWidth - 2; // Space between the borders (| |)
        int padding = (contentWidth - headerStr.length()) / 2;
        int paddingLeft = padding;
        int paddingRight = contentWidth - headerStr.length() - paddingLeft; // Adjust right padding
dynamically

        cout << CYAN << "+------------------------------------------------+" << RESET << endl;
        cout << CYAN << "|              CUSTOMER MENU                     |" << RESET << endl;
        cout << "| " << string(paddingLeft, ' ') << headerStr << string(paddingRight, ' ') << " |\n" << RESET;
        cout << CYAN << "+------------------------------------------------+" << RESET << endl;

        // Menu Options
        cout << BRIGHT_GREEN << "| 1. View Menu                             |\n" << RESET;
        cout << BRIGHT_GREEN << "| 2. Add order to Cart                     |\n" << RESET;
        cout << BRIGHT_GREEN << "| 3. View Cart                            |\n" << RESET;
        cout << BRIGHT_GREEN << "| 4. Remove Item from Cart                 |\n" << RESET;
        cout << BRIGHT_GREEN << "| 5. View Recommendations                  |\n" << RESET;
        cout << BRIGHT_GREEN << "| 6. View Order History                    |\n" << RESET;
```

```
    cout << BRIGHT_GREEN << "| 7. Confirm and Place Order                |\n" << RESET;
    cout << BRIGHT_GREEN << "| 8. View Profile                          |\n" << RESET;
    cout << BRIGHT_GREEN << "| 9. Rate a Food                           |\n" << RESET;
    cout << BRIGHT_RED << "| 0. Logout                                  |\n" << RESET;
    cout << CYAN << "+-------------------------------------------------+" << RESET << endl;
    cout << YELLOW << "Enter your choice: " << RESET;
    cin >> choice_customer_menu;


    bool isValid = true;
    // Input validation
    if (cin.fail() || choice_customer_menu < 0 || choice_customer_menu > 9) {
      cin.clear();
      cin.ignore(numeric_limits<streamsize>::max(), '\n');
      cout << BRIGHT_RED << " Invalid choice! Please enter 1 - 9 only.\n" << RESET;
      choice_customer_menu = -1;
      system("pause");
      continue;
    }

    system("cls"); // Refresh screen after every menu option

    switch (choice_customer_menu) {
    case 1: // View Menu
      viewMenu(db);
      system("pause");
      break;

    case 2: { // Add order to Cart
      int itemID;
      do {
        system("cls");

        // Display the menu header
        cout << CYAN <<
"+=====================================================================+" << RESET <<
endl;
        cout << CYAN << "|                    ORDER MENU                     |" << RESET << endl;
        cout << CYAN <<
"+=====================================================================+" << RESET <<
endl;

        // Query to get all menu items
        string query = "SELECT category, menu_item_id, item_name, item_price FROM menu_item
ORDER BY category, item_price";
        db.prepareStatement(query);
        db.QueryResult();

        string currentCategory = "";
        bool hasItems = false;

        // Display all items by category
        while (db.res->next()) {
          hasItems = true;
          string category = db.res->getString("category");

          if (category != currentCategory) {
```

```
                if (!currentCategory.empty()) {
                    cout << CYAN << "+------------------+----------------------------------+---------+" << RESET <<
endl << endl;
                }
                currentCategory = category;
                cout << YELLOW << "| Category: " << left << setw(39) << category << "              |" <<
RESET << endl;
                cout << CYAN << "+------------------+----------------------------------+---------+" << RESET <<
endl;
                cout << GREEN << "| Item ID          | Item Name                        |Price(RM)|" << RESET <<
endl;
                cout << CYAN << "+------------------+----------------------------------+---------+" << RESET <<
endl;
            }

            int itemID = db.res->getInt("menu_item_id");
            string itemName = db.res->getString("item_name");
            double itemPrice = db.res->getDouble("item_price");

            // Format price to 2 decimal places
            stringstream priceStr;
            priceStr << fixed << setprecision(2) << itemPrice;

            cout << "| " << left << setw(18) << itemID
                << "| " << left << setw(35) << itemName
                << "| " << right << setw(7) << priceStr.str() << " |" << endl;
        }

        if (!hasItems) {
            displayMessageBox("No items available in the menu!", BRIGHT_RED);
            break;
        }

        cout << CYAN << "+------------------+----------------------------------+---------+" << RESET << endl;

        // Add item to cart section
        cout << "\n" << CYAN <<
"+=============================================================================+" <<
RESET << endl;
        cout << CYAN << "|                    ADD ITEM TO CART                         |" << RESET << endl;
        cout << CYAN <<
"+=============================================================================+" <<
RESET << endl;

        cout << YELLOW << "Enter Item ID (0 to cancel): " << RESET;
        cin >> itemID;

        // Validate input
        if (cin.fail()) {
            cin.clear(); // Clear the error flag
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
            displayMessageBox("Invalid input! Please enter a numeric value.", BRIGHT_RED);
            system("pause");
            continue;
        }

        if (itemID == 0) break;
```

```
            // Validate item ID exists
            db.prepareStatement("SELECT item_name FROM menu_item WHERE menu_item_id = ?");
            db.stmt->setInt(1, itemID);
            db.QueryResult();

            if (!db.res->next()) {
                displayMessageBox("Invalid Item ID! No item found with this ID.", BRIGHT_RED);
                system("pause");
                continue; // Loop back for another attempt
            }

            string itemName = db.res->getString("item_name");
            int quantity = -1;

            cout << YELLOW << "Enter Quantity for " << itemName << " (0 to cancel): " << RESET;
            cin >> quantity;

            // Validate input
            if (cin.fail()) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                displayMessageBox("Invalid input! Please enter a numeric value.", BRIGHT_RED);
                system("pause");
                continue;
            }

            if (quantity == 0) break;

            if (quantity < 0) {
                displayMessageBox("Invalid quantity! Please enter a positive number.", RED);
                system("pause");
                continue; // Loop back for another attempt
            }

            // Add to cart
            cart.addItem(itemID, quantity, db);

        cout << YELLOW << "Do you want to add another item to the cart?" << RESET << endl;
        cout << BRIGHT_GREEN << "1. Yes" << RESET << endl;
        cout << BRIGHT_RED << "0. No" << RESET << endl;
        cout << YELLOW << "Enter your choice: " << RESET;
        int continueAdding;
        cin >> continueAdding;

        if (continueAdding != 1) break;

    } while (true);

    break;
}

case 3: // View Cart
    cout << CYAN << "+---------------------------------------------------------+\n" << RESET;
    cout << CYAN << "|                    YOUR CART                    |\n" << RESET;
    cout << CYAN << "+---------------------------------------------------------+\n" << RESET;
    cart.viewCart(db);
```

```
        system("pause"); // Wait for user to review the cart
        break;

    case 4: { // Remove an Item from Cart
        if (cart.isEmpty()) {
            system("cls");
            cout << CYAN <<
"\n============================================================\n" << RESET;
            cout << CYAN << "                Remove Item from Cart               \n" << RESET;
            cout << CYAN << "============================================================\n"
<< RESET;
            cout << BRIGHT_RED << "\nYour cart is empty. Nothing to remove.\n" << RESET;
            system("pause");  // Add pause to keep the message visible
        }
        else {
            while (true) {
                // Clear the screen and display the current cart
                system("cls");
                cout << CYAN <<
"\n============================================================\n" << RESET;
                cout << CYAN << "                Remove Item from Cart               \n" << RESET;
                cout << CYAN <<
"============================================================\n" << RESET;

                // Display the cart
                cart.viewCart(db);

                cout << YELLOW << "Enter " << RESET;
                cout << BRIGHT_YELLOW << "'all' " << RESET;
                cout << YELLOW << "to remove all items / quantity, or " << RESET;
                cout << BRIGHT_RED "'0' " << RESET;
                cout << YELLOW << "to return to the menu / cancel. \n\n" << RESET;

                // Prompt user for options
                cout << YELLOW << "\nEnter the Item ID to remove: " << RESET;
                string input;
                cin >> input;

                // Convert input to lowercase for case-insensitivity
                transform(input.begin(), input.end(), input.begin(), ::tolower);

                if (input == "0") {
                    cout << "Returning to Customer Menu...\n";
                    break;
                }

                if (input == "all") {
                    system("cls");  // Clear screen before showing final empty cart
                    cout << CYAN <<
"\n============================================================\n" << RESET;
                    cout << CYAN << "                Remove Item from Cart               \n" << RESET;
                    cout << CYAN <<
"============================================================\n" << RESET;
                    cart.clearCart();
                    cout << BRIGHT_GREEN << "\nAll items have been removed. Your cart is now empty.\n"
<< RESET;
                    system("pause");
```

```
                break;
            }

            int itemID;
            try {
                itemID = stoi(input);
            }
            catch (...) {
                system("cls");
                cout << BRIGHT_RED << "\nInvalid input. Please enter a valid Item ID, '0', or 'all'.\n" <<
RESET;
                system("pause");
                continue;
            }
            cout << YELLOW << "\nEnter the quantity to remove: " << RESET;
            string quantityInput;
            cin >> quantityInput;

            // Convert input to lowercase for case-insensitivity
            transform(input.begin(), input.end(), input.begin(), ::tolower);

            if (quantityInput == "0") {
                system("cls");
                cout << "\nReturning to the Customer menu...\n";
                system("pause");
                continue;
            }

            int quantity;
            try {
                quantity = stoi(quantityInput);
            }
            catch (...) {
                system("cls");
                cout << "\nInvalid input. Please enter a valid quantity or 'back'.\n";
                system("pause");
                continue;
            }

            // Remove the item and refresh display
            cart.removeItem(itemID, quantityInput, db);
            system("pause");
        }
    }
    break;
}

case 5: // View Recommendations
    viewRecommendations(db);
    break;

case 6: // View Order History
    viewOrderHistory(db, customerID);
    system("pause");
    break;

case 7: // Confirm and Place Order
```

```
        while (true) {
            system("cls"); // Clear the screen (refresh page)

            // Display confirmation menu
            cout << CYAN <<
"+=========================================================================+\n" << RESET;
            cout << CYAN << "|                      CONFIRM ORDER                      |\n" << RESET;
            cout << CYAN <<
"+=========================================================================+\n" << RESET;

            if (cart.isEmpty()) {
                cout << RED << "Your cart is empty. Add items before placing an order.\n" << RESET;
                system("pause");
                break; // Exit the loop since there's no action to take
            }

            // Show cart contents
            cart.viewCart(db);

            // Display options
            cout << YELLOW << "\nDo you want to confirm your order? \n" << RESET;
            cout << BRIGHT_GREEN << "1. Yes \n" << RESET;
            cout << BRIGHT_RED << "0. No \n" << RESET;

            int confirm;
            cout << YELLOW << "Enter your choice: " << RESET;
            cin >> confirm;

            // Input validation
            if (cin.fail() || confirm < 0 || confirm > 1) {
                cin.clear(); // Clear error state
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
                cout << ERROR << " Invalid choice! Please enter 1 to confirm or 0 to cancel.\n" << RESET;
                system("pause"); // Pause before refreshing
                continue; // Refresh the page
            }

            // Valid input received
            if (confirm == 1) {
                vector<int> menuItems, quantities;

                // Extract items and quantities from the cart
                for (const auto& item : cart.items) {
                    menuItems.push_back(item.first);  // Item ID
                    quantities.push_back(item.second); // Quantity
                }

                // Place the order
                placeOrder(db, customerID, menuItems, quantities);
                cart.clearCart(); // Clear the cart after placing the order

                cout << BRIGHT_GREEN << "\n\nOrder placed successfully! Thank you.\n" << RESET;
            }
            else {
                cout << BRIGHT_RED << "Order not confirmed. You can continue adding items.\n" <<
RESET;
            }
```

```
            system("pause");
            break; // Exit the loop after handling the confirmation
        }
        break;

    case 8: // View Profile
        viewProfile(db, customerID);
        system("pause");
        break;

    case 9: { // Rate Food
        int continueRating = 1;
        while (continueRating) {
            system("cls");
            cout << BLUE <<
"+=====================================================================+" << RESET <<
endl;
            cout << BLUE << "|                   YOUR ORDERED ITEMS                    |" << RESET << endl;
            cout << BLUE <<
"+=====================================================================+" << RESET <<
endl;

            string query =
                "SELECT DISTINCT m.category, m.menu_item_id, m.item_name, m.item_price, "
                "IFNULL(fr.rating, 0) as your_rating "
                "FROM menu_item m "
                "JOIN order_details od ON m.menu_item_id = od.menu_item_id "
                "JOIN `order` o ON od.order_id = o.order_id "
                "LEFT JOIN food_ratings fr ON m.menu_item_id = fr.menu_item_id "
                "AND fr.customer_id = ? "
                "WHERE o.customer_id = ? "
                "ORDER BY m.category, m.item_price";

            db.prepareStatement(query);
            db.stmt->setInt(1, customerID);  // Use customerID instead of currentCustomerID
            db.stmt->setInt(2, customerID);  // Use customerID instead of currentCustomerID
            db.QueryResult();

            string currentCategory = "";
            vector<int> validItemIDs;

            while (db.res->next()) {
                string category = db.res->getString("category");
                if (category != currentCategory) {
                    if (!currentCategory.empty()) {
                        cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET <<
endl << endl;
                    }
                    currentCategory = category;
                    cout << YELLOW << "| Category: " << left << setw(39) << category << "             |" <<
RESET << endl;
                    cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET <<
endl;
                    cout << GREEN << "| Item ID          | Item Name                         |Price(RM)|" << RESET <<
endl;
```

```
            cout << CYAN << "+-------------------+-----------------------------------+---------+" << RESET <<
endl;
        }

        int itemID = db.res->getInt("menu_item_id");
        validItemIDs.push_back(itemID);
        string itemName = db.res->getString("item_name");
        double itemPrice = db.res->getDouble("item_price");
        int yourRating = db.res->getInt("your_rating");

        stringstream priceStr;
        priceStr << fixed << setprecision(2) << itemPrice;

        cout << "| " << left << setw(18) << itemID
           << "| " << left << setw(35) << itemName
           << "| " << right << setw(7) << priceStr.str() << " |";

        if (yourRating > 0) {
           cout << YELLOW << " (Your rating: " << yourRating << "/5)" << RESET;
        }
        cout << endl;
      }
      cout << CYAN << "+-------------------+-----------------------------------+---------+" << RESET << endl;

      if (validItemIDs.empty()) {
        cout << RED << "\nYou haven't ordered any items yet. Please order something first before
rating.\n" << RESET;
        cout << GREEN << "Press Enter to return to the customer menu..." << RESET << endl;
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cin.get();
        break;
      }

      int itemID, rating;
      bool validInput = false;

      while (!validInput) {
        cout << YELLOW << "Enter the Menu Item ID that you want to rate (" << RESET;
        cout << BRIGHT_RED << "or 0 to cancel" << RESET;
        cout << YELLOW << "): " << RESET;
        if (cin >> itemID) {
          if (itemID == 0) {
            cout << GREEN << "Operation canceled. Returning to the customer menu.\n" << RESET;
            break;
          }
          if (find(validItemIDs.begin(), validItemIDs.end(), itemID) != validItemIDs.end()) {
            validInput = true;
          }
          else {
            cout << RED << "Invalid Item ID. Please select an ID from your ordered items above.\n"
<< RESET;
          }
        }
        else {
          cin.clear();
          cin.ignore(numeric_limits<streamsize>::max(), '\n');
          cout << RED << "Invalid input. Please enter a valid integer.\n" << RESET;
```

```
            }
        }

        if (!validInput || itemID == 0) break;

        validInput = false;
        while (!validInput) {
            cout << YELLOW << "Enter your rating (1-5) (" << RESET;
            cout << BRIGHT_RED << "or 0 to cancel" << RESET;
            cout << YELLOW << "): " << RESET;
            if (cin >> rating) {
                if (rating == 0) {
                    cout << GREEN << "Operation canceled. Returning to the customer menu.\n" << RESET;
                    break;
                }
                if (rating >= 1 && rating <= 5) {
                    validInput = true;
                }
                else {
                    cout << RED << "Invalid rating. Please provide a rating between 1 and 5.\n" << RESET;
                }
            }
            else {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << RED << "Invalid input. Please enter a valid integer.\n" << RESET;
            }
        }

        if (!validInput || rating == 0) break;

        if (rateFood(db, customerID, itemID, rating)) {
            db.prepareStatement(
                "SELECT m.item_name, m.total_rating, m.rating_count, "
                "CAST(m.total_rating * 1.0 / m.rating_count AS DECIMAL(5,2)) AS avg_rating, "
                "fr.rating as your_rating, fr.updated_at "
                "FROM menu_item m "
                "JOIN food_ratings fr ON m.menu_item_id = fr.menu_item_id "
                "WHERE m.menu_item_id = ? AND fr.customer_id = ?"
            );
            db.stmt->setInt(1, itemID);
            db.stmt->setInt(2, customerID);
            db.QueryResult();

            if (db.res->next()) {
                string itemName = db.res->getString("item_name");
                int totalRating = db.res->getInt("total_rating");
                int ratingCount = db.res->getInt("rating_count");
                double avgRating = db.res->getDouble("avg_rating");
                int yourRating = db.res->getInt("your_rating");
                string updateTime = db.res->getString("updated_at");

                cout << GREEN <<
"\n+=========================================================================+" << RESET
<< endl;
                cout << GREEN << "|                  UPDATED RATING INFORMATION                  |" <<
RESET << endl;
```

```
                cout << GREEN <<
"+=====================================================================+" << RESET
<< endl;
                cout << "Item Name      : " << itemName << endl;
                cout << "Your Rating    : " << yourRating << "/5" << endl;
                cout << "Last Updated   : " << updateTime << endl;
                cout << "Total Ratings  : " << totalRating << endl;
                cout << "Rating Count   : " << ratingCount << endl;
                cout << "Average Rating : " << avgRating << endl;
                cout << GREEN <<
"+=====================================================================+\n\n" <<
RESET;
              }

              cout << YELLOW << "Thank you for rating the food item!" << RESET << endl;
            }

            cout << YELLOW << "Would you like to rate another item? (" << RESET;
            cout << BRIGHT_GREEN << "1. Yes, " << RESET;
            cout << BRIGHT_RED << "0. No" << RESET;
            cout << YELLOW << "): " << RESET;
            cin >> continueRating;

            if (!continueRating) {
              cout << GREEN << "Thank you for your ratings! Returning to the customer menu.\n" <<
RESET;
            }
          }
          break;
      }

      case 0: // Logout
          cout << YELLOW << "Logging out...\n" << RESET;
          return;

      case -1:
          cout << RED << "Invalid choice. Please try again.\n" << RESET;
          system("pause");
          break;
      }

    } while (choice_customer_menu != 0);
}

// Staff Menu
void staffMenu(bool isManager, int staffID) {
    DBConnection db;
    int choice_staff_menu, sub_choice;

    // Fetch staff name and role for personalization
    string staffName, staffRole;
    db.prepareStatement("SELECT staff_name, staff_role FROM staff WHERE staff_id = ?");
    db.stmt->setInt(1, staffID);
    db.QueryResult();
    if (db.res->next()) {
        staffName = db.res->getString("staff_name");
        staffRole = db.res->getString("staff_role");
```

```
    }
    else {
        staffName = "Unknown";
        staffRole = isManager ? "Manager" : "Staff";
    }

    do {
        system("cls");

        // Header with staff welcome message
        stringstream header;
        header << "Welcome " << staffRole << ", " << staffName << " (ID: " << staffID << ")";
        string headerStr = header.str();
        int totalWidth = 47;
        int padding = (totalWidth - headerStr.length()) / 2;
        int paddingLeft = padding, paddingRight = padding;
        if (headerStr.length() % 2 != 0) paddingRight++;

        cout << CYAN << "+-----------------------------------------------+" << RESET << endl;
        cout << CYAN << "|                STAFF MENU                     |" << RESET << endl;
        cout << "| " << string(paddingLeft, ' ') << headerStr << string(paddingRight, ' ') << " |\n";
        cout << CYAN << "+-----------------------------------------------+" << RESET << endl;

        // Dynamic menu based on role
        int menuOption = 1;
        if (isManager) {
            cout << BRIGHT_GREEN << "| " << menuOption++ << ".  Staff Management                     |" <<
RESET << endl;
        }
        cout << BRIGHT_GREEN << "| " << menuOption++ << ".  Menu Management                      |" <<
RESET << endl;
        cout << BRIGHT_GREEN << "| " << menuOption++ << ".  Customer Management                  |"
<< RESET << endl;
        cout << BRIGHT_GREEN << "| " << menuOption++ << ".  Payment Processing                   |" <<
RESET << endl;
        cout << BRIGHT_GREEN << "| " << menuOption++ << ".  Reports & Analytics                  |" <<
RESET << endl;
        cout << BRIGHT_RED << "| 0.  Logout                                |" << RESET << endl;
        cout << CYAN << "+-----------------------------------------------+" << RESET << endl;
        cout << YELLOW << "Enter your choice: " << RESET;
        cin >> choice_staff_menu;

        // Adjust validation based on available options
        int maxChoice = isManager ? 5 : 4;
        if (cin.fail() || choice_staff_menu < 0 || choice_staff_menu > maxChoice) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            displayMessageBox("Invalid choice! Please enter a valid number.", RED);
            system("pause");
            continue;
        }

        // Handle logout before adjustment
        if (choice_staff_menu == 0) {
            cout << YELLOW << "Logging out...\n" << RESET;
            for (int i = 0; i < 3; i++) {
                this_thread::sleep_for(chrono::milliseconds(500));
```

```
            cout << ".";
        }
        cout << endl;
        displayMessageBox("Successfully logged out!", GREEN);
        return;
    }

    system("cls");

    // Adjust case numbers based on manager status
    int adjustedChoice = isManager ? choice_staff_menu : choice_staff_menu + 1;

    switch (adjustedChoice) {
    case 1: { // Staff Management (Manager Only)
        if (!isManager) {
            displayMessageBox("Access denied. Manager privileges required.", RED);
            system("pause");
            break;
        }
        do {
            system("cls");
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << CYAN << "|              STAFF MANAGEMENT                  |" << RESET << endl;
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << BRIGHT_GREEN << "| 1. Add New Staff                               |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 2. Update Staff Details                        |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 3. View Staff List                             |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 4. Remove Staff                                |" << RESET << endl;
            cout << BRIGHT_RED << "| 0. Back to Main Menu                           |" << RESET << endl;
            cout << CYAN << "+------------------------------------------------+" << RESET << endl;
            cout << YELLOW << "Enter your choice: " << RESET;
            cin >> sub_choice;

            if (cin.fail()) {
                cin.clear(); // Clear error flags
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
                displayMessageBox("Invalid input! Please enter a valid number.", BRIGHT_RED);
                continue;
            }

            switch (sub_choice) {
            case 1: registerStaff(db); break;
            case 2: updateStaff(db); break;
            case 3: viewStaffList(db); break;
            case 4: removeStaff(db); break;
            case 0: break;
            default: displayMessageBox("Invalid choice! Please try again.", BRIGHT_RED);
            }
            if (sub_choice != 0) system("pause");
        } while (sub_choice != 0);
        break;
    }
    case 2: { // Menu Management
        do {
            system("cls");
```

```
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << CYAN << "|              MENU MANAGEMENT               |" << RESET << endl;
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << BRIGHT_GREEN << "| 1. Add Menu Item                           |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 2. Update Menu Item                        |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 3. Delete Menu Item                        |" << RESET << endl;
            cout << BRIGHT_RED << "| 0. Back to Main Menu                       |" << RESET << endl;
            cout << CYAN << "+------------------------------------------------+" << RESET << endl;
            cout << YELLOW << "Enter your choice: " << RESET;
            cin >> sub_choice;

            switch (sub_choice) {
            case 1: addMenu(db); break;
            case 2: updateMenu(db); break;
            case 3: deleteMenu(db); break;
            case 0: break;
            default: displayMessageBox("Invalid choice! Please try again.", RED);
            }
            if (sub_choice != 0) system("pause");
        } while (sub_choice != 0);
        break;
    }
    case 3: { // Customer Management
        do {
            system("cls");
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << CYAN << "|            CUSTOMER MANAGEMENT             |" << RESET << endl;
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << BRIGHT_GREEN << "| 1. Register Customer Membership            |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 2. Update Customer Information             |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 3. View Customer List                      |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 4. Remove Customer Membership              |" << RESET <<
endl;
            cout << BRIGHT_GREEN << "| 5. Remove Customer                         |" << RESET << endl;
            cout << BRIGHT_RED << "| 0. Back to Main Menu                       |" << RESET << endl;
            cout << CYAN << "+------------------------------------------------+" << RESET << endl;
            cout << YELLOW << "Enter your choice: " << RESET;
            cin >> sub_choice;

            switch (sub_choice) {
            case 1: registerMembership(db); break;
            case 2: updateCustomerInformation(db); break;
            case 3: viewCustomerList(db); break;
            case 4: removeMembership(db); break;
            case 5: removeCustomer(db); break;
            case 0: break;
            default: displayMessageBox("Invalid choice! Please try again.", RED);
            }
            if (sub_choice != 0) system("pause");
        } while (sub_choice != 0);
        break;
    }
    case 4: { // Payment Processing
```

```
        do {
            system("cls");
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << CYAN << "|            PAYMENT PROCESSING              |" << RESET << endl;
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << BRIGHT_GREEN << "| 1. Process Order Payment                  |" << RESET << endl;
            cout << BRIGHT_RED << "| 0. Back to Main Menu                      |" << RESET << endl;
            cout << CYAN << "+------------------------------------------------+" << RESET << endl;
            cout << YELLOW << "Enter your choice: " << RESET;
            cin >> sub_choice;

            switch (sub_choice) {
            case 1:
                payOrder(db);
                break;
            case 0:
                break;
            default:
                displayMessageBox("Invalid choice! Please try again.", RED);
            }
            if (sub_choice != 0) system("pause");
        } while (sub_choice != 0);
        break;
    }
    case 5: { // Reports & Analytics
        do {
            system("cls");
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << CYAN << "|            REPORTS & ANALYTICS             |" << RESET << endl;
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << BRIGHT_GREEN << "| 1. Generate Sales Report                  |" << RESET << endl;
            cout << BRIGHT_GREEN << "| 2. View Sales Bar Graph                   |" << RESET << endl;
            cout << BRIGHT_RED << "| 0. Back to Main Menu                      |" << RESET << endl;
            cout << CYAN << "+------------------------------------------------+" << RESET << endl;
            cout << YELLOW << "Enter your choice: " << RESET;
            cin >> sub_choice;

            if (cin.fail() || sub_choice < 0 || sub_choice > 2) { // Validate input
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << ERROR << " Invalid choice! Please enter 1, 2, or 0.\n" << RESET;
                system("pause");
                continue;
            }

            switch (sub_choice) {
            case 1: {
                system("cls");
                cout << CYAN << "\n=== Generate Sales Report ===\n" << RESET;
                cout << BRIGHT_GREEN << "1. Daily\n";
                cout << "2. Monthly\n";
                cout << "3. Yearly\n";
                cout << "0. Go Back to Main Menu\n" << RESET;
```

```
        cout << YELLOW << "Enter your choice: " << RESET;

        int choice;
        cin >> choice;

        if (cin.fail() || choice < 0 || choice > 3) { // Validate input
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << ERROR << " Invalid choice! Please enter 1, 2, or 0.\n" << RESET;
            system("pause");
            continue;
        }

        if (choice == 0) break;

        string timeframe;
        string date;

        switch (choice) {
        case 1: {
            timeframe = "daily";
            cout << YELLOW << "Enter the date (YYYY-MM-DD): " << RESET;
            cin >> date;

            // Validate the date format for daily reports
            if (!regex_match(date, regex("^\\d{4}-\\d{2}-\\d{2}$"))) {
                cout << BRIGHT_RED << "Invalid date format! Please enter in YYYY-MM-DD format.\n"
<< RESET;
                system("pause");
                continue;
            }
            break;
        }
        case 2: {
            timeframe = "monthly";
            cout << YELLOW << "Enter the month and year (YYYY-MM): " << RESET;
            cin >> date;

            // Validate the date format for monthly reports
            if (!regex_match(date, regex("^\\d{4}-\\d{2}$"))) {
                cout << BRIGHT_RED << "Invalid date format! Please enter in YYYY-MM format.\n" <<
RESET;
                system("pause");
                continue;
            }
            break;
        }
        case 3: {
            timeframe = "yearly";
            cout << YELLOW << "Enter the year (YYYY): " << RESET;
            cin >> date;

            // Validate the date format for yearly reports
            if (!regex_match(date, regex("^\\d{4}$"))) {
                cout << BRIGHT_RED << "Invalid date format! Please enter in YYYY format.\n" <<
RESET;
                system("pause");
```

```
                continue;
            }
            break;
        }
        default:
            cout << BRIGHT_RED << "Invalid choice! Returning to the previous menu.\n" << RESET;
            system("pause");
            continue;
        }

        // Call the report generation function
        if (!date.empty()) {
            generateSalesReport(db, timeframe, date);
        }
        break;
      }

      case 2:
        generateSalesBarGraph(db);
        break;
      case 0:
        break;
      default:
        displayMessageBox("Invalid choice! Please try again.", RED);
      }
      if (sub_choice != 0) system("pause");
    } while (sub_choice != 0);
    break;
  }
  }
 } while (true);
}

// Function to login for customer or staff
bool loginUser(DBConnection& db, int choice_login, string email, string password, int& customerID) {
 try {
  if (choice_login == 1) {
    // Check credentials for customer
    db.prepareStatement("SELECT * FROM customer WHERE customer_email = ? AND
customer_password = ?");
    db.stmt->setString(1, email);
    db.stmt->setString(2, password);
    db.QueryResult();

    if (db.res->next()) {
      customerID = db.res->getInt("customer_id");  // Get the customer ID
      return true;  // Customer login successful
    }
  }
  else if (choice_login == 2) {
    // Check credentials for staff
    db.prepareStatement("SELECT * FROM staff WHERE staff_email = ? AND staff_password = ?");
    db.stmt->setString(1, email);
    db.stmt->setString(2, password);
    db.QueryResult();

    if (db.res->next()) {
```

```
            return true;  // Staff login successful
        }
      }
      cout << "Invalid login credentials." << endl;
      return false;  // Invalid login
   }
   catch (sql::SQLException& e) {
      system("cls");
      cerr << "Error logging in: " << e.what() << endl;
      return false;
   }
}

// Function to view recommendations
void viewRecommendations(DBConnection& db) {
   try {
      int recommendChoice;

      do {
         system("cls");
         cout << CYAN << "+================================================+\n" << RESET;
         cout << CYAN << "|              RECOMMENDATIONS              |\n" << RESET;
         cout << CYAN << "+================================================+\n" << RESET;

         cout << BRIGHT_GREEN << "| 1. Top Sales (by Timeframe)                |\n" << RESET;
         cout << BRIGHT_GREEN << "| 2. Food Ratings (Show All)                 |\n" << RESET;
         cout << BRIGHT_RED << "| 0. Back to Main Menu                       |\n" << RESET;
         cout << CYAN << "+================================================+\n" << RESET;

         cout << YELLOW << "Enter your choice: " << RESET;
         cin >> recommendChoice;

         if (cin.fail() || recommendChoice < 0 || recommendChoice > 2) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << ERROR << "Invalid choice! Please try again." << RESET << endl;
            system("pause");
            continue;
         }

         system("cls");

         switch (recommendChoice) {
         case 1: { // Top Sales
            cout << CYAN << "+================================================+\n" <<
RESET;
            cout << CYAN << "|                TOP SALES                |\n" << RESET;
            cout << CYAN << "+================================================+\n" <<
RESET;

            int timeFrame;
            cout << YELLOW << "Choose a Timeframe:\n" << RESET;
            cout << BRIGHT_GREEN << "1. Today\n" << RESET;
            cout << BRIGHT_GREEN << "2. Yesterday\n" << RESET;
            cout << BRIGHT_GREEN << "3. Last Month\n" << RESET;
            cout << BRIGHT_RED << "0. Go Back\n" << RESET;
            cout << YELLOW << "Enter your choice: " << RESET;
```

```
cin >> timeFrame;

// Input validation
if (cin.fail() || timeFrame < 0 || timeFrame > 3) {
   cin.clear();
   cin.ignore(numeric_limits<streamsize>::max(), '\n');
   cout << BRIGHT_RED << " Invalid choice! Please enter 1, 2, 3 or 0.\n" << RESET;
   system("pause");
   continue;
}

// Handle the "Go Back" option
if (timeFrame == 0) {
   cout << YELLOW << "Returning to Recommendations Menu..." << RESET << endl;
   system("pause");
   break; // Exit this case and return to the recommendations menu
}

string query;
if (timeFrame == 1) {
   query = "SELECT m.menu_item_id, m.item_name, SUM(s.daily_sales) AS total_sales "
      "FROM sales s JOIN menu_item m ON s.menu_item_id = m.menu_item_id "
      "WHERE s.sale_date = CURDATE() "
      "GROUP BY m.menu_item_id, m.item_name "
      "ORDER BY total_sales DESC LIMIT 5";
}
else if (timeFrame == 2) {
   query = "SELECT m.menu_item_id, m.item_name, SUM(s.daily_sales) AS total_sales "
      "FROM sales s JOIN menu_item m ON s.menu_item_id = m.menu_item_id "
      "WHERE s.sale_date = CURDATE() - INTERVAL 1 DAY "
      "GROUP BY m.menu_item_id, m.item_name "
      "ORDER BY total_sales DESC LIMIT 5";
}
else if (timeFrame == 3) {
   query = "SELECT m.menu_item_id, m.item_name, SUM(s.monthly_sales) AS total_sales "
      "FROM sales s JOIN menu_item m ON s.menu_item_id = m.menu_item_id "
      "WHERE YEAR(s.sale_date) = YEAR(CURDATE() - INTERVAL 1 MONTH) "
      "AND MONTH(s.sale_date) = MONTH(CURDATE() - INTERVAL 1 MONTH) "
      "GROUP BY m.menu_item_id, m.item_name "
      "ORDER BY total_sales DESC LIMIT 5";
}

try {
   db.prepareStatement(query);
   db.QueryResult();

   if (!db.res->next()) {
      cout << ERROR << "No sales data found for the selected timeframe!" << RESET << endl;
      system("pause");
      break;
   }

   cout << SUCCESS << "\nTop Recommended Foods by Sales:\n" << RESET;
   cout << CYAN << "+----------------+----------------------+----------------+\n" << RESET;
   cout << GREEN << "| Item ID        | Item Name            | Total Sales    |\n" << RESET;
   cout << CYAN << "+----------------+----------------------+----------------+\n" << RESET;
```

```
            do {
                int itemID = db.res->getInt("menu_item_id");
                string itemName = db.res->getString("item_name");
                int totalSales = db.res->getInt("total_sales");

                cout << "| " << setw(16) << left << itemID
                    << "| " << setw(20) << left << itemName
                    << "| " << setw(14) << totalSales << "|\n";
            } while (db.res->next());

            cout << CYAN << "+----------------+----------------------+----------------+\n" << RESET;
            system("pause");
        }
        catch (sql::SQLException& e) {
            cerr << ERROR << "SQL Error: " << e.what() << "\nError Code: " << e.getErrorCode() <<
RESET << endl;
        }
        break;
    }
    case 2: { // Food Ratings
        cout << CYAN <<
"+================================================================================
======+\n" << RESET;
        cout << CYAN << "|                          FOODS BY RATINGS                          |\n" << RESET;
        cout << CYAN <<
"+================================================================================
======+\n" << RESET;

        string query = "SELECT menu_item_id, item_name, total_rating, rating_count, "
            "(CASE WHEN rating_count > 0 THEN total_rating / rating_count ELSE 0 END) AS
average_rating "
            "FROM menu_item ORDER BY average_rating DESC, total_rating DESC";

        db.prepareStatement(query);
        db.QueryResult();

        cout << SUCCESS << "\nAll Foods by Ratings:\n" << RESET;
        cout << CYAN << "+-----------------+-----------------------------------+---------------+---------------+\n" <<
RESET;
        cout << GREEN << "| Item ID        | Item Name                    | Average Rating| Rating Count
|\n" << RESET;
        cout << CYAN << "+-----------------+-----------------------------------+---------------+---------------+\n" <<
RESET;

        while (db.res->next()) {
            int itemID = db.res->getInt("menu_item_id");
            string itemName = db.res->getString("item_name");
            double averageRating = db.res->getDouble("average_rating");
            int ratingCount = db.res->getInt("rating_count");

            cout << "| " << setw(16) << left << itemID
                << "| " << setw(35) << left << itemName
                << "| " << setw(14) << fixed << setprecision(2) << averageRating
                << "| " << setw(14) << ratingCount << "|\n";
        }
```

```
            cout << CYAN << "+-----------------+-----------------------------------+--------------+--------------+\n" <<
RESET;
            system("pause");
            break;
         }
         case 0: // Back to Main Menu
            break;
         default:
            cout << BRIGHT_RED << "Invalid choice. Please try again." << RESET << endl;
            system("pause");
         }
      } while (recommendChoice != 0);
   }
   catch (sql::SQLException& e) {
      cerr << ERROR << "Error fetching recommendations: " << e.what() << RESET << endl;
   }
}

//Add item into menu
void addMenu(DBConnection& db) {
   while (true) {
      try {
         // Clear screen and display header
         system("cls");
         cout << CYAN << "+================================================+" << RESET <<
endl;
         cout << CYAN << "|               *** ADD ITEM ***            |" << RESET << endl;
         cout << CYAN << "+================================================+" << RESET <<
endl;

         // Fetch categories
         string query = "SELECT DISTINCT category FROM menu_item ORDER BY category";
         db.prepareStatement(query);
         db.QueryResult();

         vector<string> categories;
         int index = 1;

         // Display categories in table format
         cout << CYAN << "+-------------------------------------------------+" << RESET << endl;
         cout << CYAN << "|  NO  |             CATEGORY               |" << RESET << endl;
         cout << CYAN << "+-------------------------------------------------+" << RESET << endl;

         while (db.res->next()) {
            string category = db.res->getString("category");
            categories.push_back(category);
            cout << BRIGHT_GREEN
               << "| " << setw(4) << right << index++ << " | "
               << setw(36) << left << category
               << RESET << CYAN << "    |\n" << RESET;
         }
         cout << YELLOW <<"| " << setw(4) << right << index << " | " << setw(36) << left << "Add New
Category" << RESET << YELLOW << "    |\n" << RESET;
         cout << CYAN << "+-------------------------------------------------+" << RESET << endl;
         cout << BRIGHT_RED << "0. Go back / cancel \n" << RESET;
         cout << YELLOW << "Enter the category number: " << RESET;
```

```
        // Get category choice
        string categoryInput;
        cin >> categoryInput;

        if (categoryInput == "0") {
            cout << BRIGHT_RED << "Operation canceled. Returning to Staff Menu...\n" << RESET;
            break;
        }

        int categoryChoice;
        try {
            categoryChoice = stoi(categoryInput);
        }
        catch (...) {
            cout << BRIGHT_RED << "Invalid input. Please enter a valid number.\n" << RESET;
            continue;
        }

        string selectedCategory;
        if (categoryChoice == index) {
            // Add new category
            cout << YELLOW << "Enter the new category name (" << RESET;
            cout << BRIGHT_RED << "or type '0' to cancel" << RESET;
            cout << YELLOW << "): " << RESET;
            cin.ignore();
            getline(cin, selectedCategory);

            if (selectedCategory == "0") {
                cout << BRIGHT_RED << "Operation canceled. Returning to Add Menu...\n" << RESET;
                continue;
            }

            // Check for duplicate category
            if (find(categories.begin(), categories.end(), selectedCategory) != categories.end()) {
                cout << BRIGHT_RED << "This category already exists. Please try again.\n" << RESET;
                continue;
            }

            // Add the new category to the database
            cout << BRIGHT_GREEN << "Adding new category: " << selectedCategory << RESET << endl;
            categories.push_back(selectedCategory); // Add locally for future iterations
        }
        else if (categoryChoice > 0 && categoryChoice < index) {
            selectedCategory = categories[categoryChoice - 1];
        }
        else {
            cout << BRIGHT_RED << "Invalid category choice. Please try again.\n" << RESET;
            continue;
        }

        // Continue with adding an item to the selected category
        system("cls");
        cout << CYAN << "+================================================+" << RESET <<
endl;
        cout << CYAN << "| Adding Item to Category: " << BRIGHT_GREEN << selectedCategory <<
RESET << CYAN << "       |" << endl;
```

```
        cout << CYAN << "+================================================+" << RESET <<
endl;

        // Get item details
        string name;
        cout << YELLOW << "Enter Item Name (" << RESET;
        cout << BRIGHT_RED << "or type '0' to cancel" << RESET;
        cout << YELLOW << "): " << RESET;
        cin.ignore();
        getline(cin, name);

        if (name == "0") {
            cout << BRIGHT_RED << "\u2716 Operation canceled. Returning to Add Menu...\n" << RESET;
            continue;
        }

        string priceInput;
        double price;
        cout << YELLOW << "Enter Price (" << RESET;
        cout << BRIGHT_RED << "or type '0' to cancel" << RESET;
        cout << YELLOW << "): " << RESET;
        cin >> priceInput;

        if (priceInput == "0") {
            cout << BRIGHT_RED << "\u2716 Operation canceled. Returning to Add Menu...\n" << RESET;
            continue;
        }

        try {
            price = stod(priceInput);
            if (price <= 0) throw invalid_argument("Price must be positive.");
        }
        catch (...) {
            cout << BRIGHT_RED << "\u2716 Invalid price. Please enter a positive number.\n" << RESET;
            continue;
        }

        // Insert item
        db.prepareStatement("SELECT MAX(menu_item_id) AS max_id FROM menu_item");
        db.QueryResult();
        db.res->next();
        int menu_item_id = db.res->getInt("max_id") + 1;

        db.prepareStatement("INSERT INTO menu_item (menu_item_id, item_name, item_price,
category) VALUES (?, ?, ?, ?)");
        db.stmt->setInt(1, menu_item_id);
        db.stmt->setString(2, name);
        db.stmt->setDouble(3, price);
        db.stmt->setString(4, selectedCategory);
        db.QueryStatement();

        // Display the item details
        cout << BRIGHT_GREEN << "\nItem successfully added to the menu!\n" << RESET;
        cout << "-------------------------------------\n";
        cout << "Item ID    : " << menu_item_id << "\n";
        cout << "Item Name  : " << name << "\n";
        cout << "Price (RM) : " << fixed << setprecision(2) << price << "\n";
```

```
        cout << "Category   : " << selectedCategory << "\n";
        cout << "-----------------------------------\n";

        cout << YELLOW << "\nDo you want to add another item?\n" << RESET;
        cout << CYAN << "1. Yes\n" << RESET;
        cout << BRIGHT_RED << "0. No\n" << RESET;
        cout << YELLOW << "Enter your choice: " << RESET;
        string choice;
        cin >> choice;
        if (choice == "0") {
          cout << BRIGHT_RED << "\u2716 Returning to Staff Menu...\n" << RESET;
          break;
        }

      }
      catch (exception& e) {
        cerr << BRIGHT_RED << "\u2716 Error: " << e.what() << RESET << endl;
      }

      // Centralized single pause for all outcomes
      system("pause");
    }
}

// Function to update a menu item
void updateMenu(DBConnection& db) {
  while (true) {
    try {
      // Step 1: Display Update Menu
      system("cls");
      cout << CYAN << "+==================================================+" << RESET <<
endl;
      cout << CYAN << "|                UPDATE MENU                       |" << RESET << endl;
      cout << CYAN << "+==================================================+" << RESET <<
endl;

      string query = "SELECT DISTINCT category FROM menu_item ORDER BY category";
      db.prepareStatement(query);
      db.QueryResult();

      vector<string> categories;
      cout << YELLOW << "\nSelect a category to update items:" << RESET << endl;
      cout << GREEN << "1. Overall (View All Items)\n" << RESET;
      int index = 2;

      while (db.res->next()) {
        string category = db.res->getString("category");
        categories.push_back(category);
        cout << GREEN << index++ << ". " << category << RESET << endl;
      }

      if (categories.empty()) {
        cout << RED << "\nNo categories found! Please add items first." << RESET << endl;
        system("pause");
        return;
      }
```

```
        cout << RED << "\n0. Cancel and Return to Staff Menu\n" << RESET;
        cout << YELLOW << "\nEnter category number: " << RESET;
        string categoryInput;
        cin >> categoryInput;

        if (categoryInput == "0") {
            cout << GREEN << "Operation canceled. Returning to Staff Menu..." << RESET << endl;
            return;
        }

        int categoryChoice = stoi(categoryInput);
        string selectedCategory;
        bool isOverall = false;

        if (categoryChoice == 1) {
            isOverall = true;
        }
        else if (categoryChoice < 2 || categoryChoice > categories.size() + 1) {
            cout << RED << "\nInvalid category choice. Please try again." << RESET << endl;
            system("pause");
            continue;
        }
        else {
            selectedCategory = categories[categoryChoice - 2];
        }

        while (true) {
            system("cls");
            cout << CYAN <<
"\n+=====================================================================+" << RESET <<
endl;
            cout << CYAN << "|                    ITEMS IN SELECTED CATEGORY               |" << RESET << endl;
            cout << CYAN <<
"+=====================================================================+" << RESET <<
endl;

            query = isOverall
                ? "SELECT menu_item_id, item_name, item_price, category FROM menu_item ORDER BY
category, menu_item_id"
                : "SELECT menu_item_id, item_name, item_price, category FROM menu_item WHERE
category = ? ORDER BY menu_item_id";
            db.prepareStatement(query);

            if (!isOverall) {
                db.stmt->setString(1, selectedCategory);
            }

            db.QueryResult();

            vector<int> itemIDs;
            string currentCategory = "";
            bool hasItems = false;

            while (db.res->next()) {
                hasItems = true;
                string category = db.res->getString("category");
```

```
            if (category != currentCategory) {
                if (!currentCategory.empty()) {
                    cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET <<
endl << endl;
                }
                currentCategory = category;
                cout << YELLOW << "| Category: " << left << setw(39) << category << "              |" <<
RESET << endl;
                cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET <<
endl;
                cout << GREEN << "| Item ID         | Item Name                      |Price(RM)|" << RESET <<
endl;
                cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET <<
endl;
            }

            int itemID = db.res->getInt("menu_item_id");
            string itemName = db.res->getString("item_name");
            double itemPrice = db.res->getDouble("item_price");

            // Format price to 2 decimal places
            stringstream priceStr;
            priceStr << fixed << setprecision(2) << itemPrice;

            itemIDs.push_back(itemID);
            cout << "| " << left << setw(18) << itemID
                << "| " << left << setw(35) << itemName
                << "| " << right << setw(7) << priceStr.str() << " |" << endl;
        }

        if (!hasItems) {
            cout << RED << "\nNo items available in the menu!" << RESET << endl;
            system("pause");
            break;
        }

        cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET << endl;

        cout << YELLOW << "\nEnter the Item ID to update (or type '0' to cancel): " << RESET;
        string itemInput;
        cin >> itemInput;

        if (itemInput == "0") {
            cout << GREEN << "Operation canceled. Returning to Update Menu..." << RESET << endl;
            system("pause");
            break;
        }

        int itemID = stoi(itemInput);
        if (find(itemIDs.begin(), itemIDs.end(), itemID) == itemIDs.end()) {
            cout << RED << "\nInvalid Item ID. Please try again." << RESET << endl;
            system("pause");
            continue;
        }

        while (true) {
```

```
            query = "SELECT item_name, item_price, category FROM menu_item WHERE
menu_item_id = ?";
            db.prepareStatement(query);
            db.stmt->setInt(1, itemID);
            db.QueryResult();

            if (!db.res->next()) {
               cout << RED << "\nFailed to fetch item details. Returning to Update Menu." << RESET <<
endl;
               system("pause");
               break;
            }

            string itemName = db.res->getString("item_name");
            double itemPrice = db.res->getDouble("item_price");
            string itemCategory = db.res->getString("category");

            system("cls");
            cout << CYAN <<
"\n+=====================================================================+" << RESET <<
endl;
            cout << CYAN << "|                  UPDATE ITEM DETAILS                  |" << RESET << endl;
            cout << CYAN <<
"+=====================================================================+" << RESET <<
endl;

            cout << "Item ID     : " << itemID << endl;
            cout << "Old Name    : " << itemName << endl;
            cout << "Old Price   : RM" << fixed << setprecision(2) << itemPrice << endl;
            cout << "Old Category : " << itemCategory << endl;

            cout << YELLOW << "\nWhat would you like to update?" << RESET << endl;
            cout << GREEN << "1. Item Name\n" << RESET;
            cout << GREEN << "2. Item Price\n" << RESET;
            cout << GREEN << "3. Item Category\n" << RESET;
            cout << GREEN << "4. All\n" << RESET;
            cout << RED << "0. Cancel and Return\n" << RESET;
            cout << YELLOW << "\nEnter your choice: " << RESET;

            string updateChoice;
            cin >> updateChoice;

            if (updateChoice == "0") {
               cout << GREEN << "Operation canceled. Returning to Update Menu..." << RESET << endl;
               system("pause");
               break;
            }

            string newItemName = itemName;
            double newItemPrice = itemPrice;
            string newItemCategory = itemCategory;

            if (updateChoice == "1" || updateChoice == "4") {
               cin.ignore();
               cout << YELLOW << "Enter new item name: " << RESET;
               getline(cin, newItemName);
            }
```

```
            if (updateChoice == "2" || updateChoice == "4") {
                cout << YELLOW << "Enter new item price: " << RESET;
                cin >> newItemPrice;
                if (newItemPrice < 0) {
                    cout << RED << "Invalid price. Please enter a positive number." << RESET << endl;
                    system("pause");
                    continue;
                }
            }

            if (updateChoice == "3" || updateChoice == "4") {
                cin.ignore();
                cout << YELLOW << "Enter new category: " << RESET;
                getline(cin, newItemCategory);
            }

            db.prepareStatement("UPDATE menu_item SET item_name = ?, item_price = ?, category = ?
WHERE menu_item_id = ?");
            db.stmt->setString(1, newItemName);
            db.stmt->setDouble(2, newItemPrice);
            db.stmt->setString(3, newItemCategory);
            db.stmt->setInt(4, itemID);
            db.QueryStatement();

            cout << GREEN << "\nItem updated successfully!" << RESET << endl;
            cout << "New Name     : " << newItemName << endl;
            cout << "New Price    : RM" << fixed << setprecision(2) << newItemPrice << endl;
            cout << "New Category : " << newItemCategory << endl;

            system("pause");
            break;
          }
        }
    }
    catch (sql::SQLException& e) {
        cerr << RED << "Error updating menu item: " << e.what() << RESET << endl;
        system("pause");
    }
    catch (exception& e) {
        cerr << RED << "Error: " << e.what() << RESET << endl;
        system("pause");
    }
  }
}

// Function to delete an inventory item
void deleteMenu(DBConnection& db) {
  while (true) {
    try {
        // Clear screen and display Delete Menu
        system("cls");
        cout << CYAN << "+================================================+" << RESET <<
endl;
        cout << CYAN << "|               DELETE MENU                |" << RESET << endl;
        cout << CYAN << "+================================================+" << RESET <<
endl;
```

```
        cout << YELLOW << "\nSelect an option:" << RESET << endl;
        cout << GREEN << "1. Delete Specific Item\n" << RESET;
        cout << GREEN << "2. Delete Entire Category\n" << RESET;
        cout << RED << "0. Cancel and Return to Staff Menu\n" << RESET;

        cout << YELLOW << "\nEnter your choice: " << RESET;
        string choiceInput;
        cin >> choiceInput;

        if (cin.fail() || choiceInput < "0" || choiceInput > "2") { // Validate input
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << ERROR << " Invalid choice! Please enter 1, 2, or 0.\n" << RESET;
            system("pause");
            continue;
        }

        if (choiceInput == "0") {
            // Single confirmation prompt before exiting
            cout << GREEN << "Operation canceled. Returning to Staff Menu..." << RESET << endl;
            return;
        }

        int choice = stoi(choiceInput);

        if (choice == 1) {
            // Call a function to delete a specific item
            deleteSpecificItem(db);
        }
        else if (choice == 2) {
            // Call a function to delete an entire category
            deleteEntireCategory(db);
        }
        else {
            cout << RED << "\nInvalid choice. Please try again." << RESET << endl;
            system("pause");
        }
    }
    catch (exception& e) {
        cerr << RED << "\nError: " << e.what() << RESET << endl;
        system("pause");
    }
  }
}

void deleteSpecificItem(DBConnection& db) {
  while (true) {
    try {
        // Display the complete menu
        system("cls");
        cout << BLUE <<
"+============================================================+" << RESET <<
endl;
        cout << BLUE << "|                    COMPLETE MENU                    |" << RESET << endl;
```

```
        cout << BLUE <<
"+=================================================================+" << RESET <<
endl;

        string query = "SELECT category, menu_item_id, item_name, item_price FROM menu_item
ORDER BY category, item_price";
        db.prepareStatement(query);
        db.QueryResult();

        string currentCategory = "";
        while (db.res->next()) {
           string category = db.res->getString("category");
           if (category != currentCategory) {
              if (!currentCategory.empty()) {
                 cout << CYAN << "+------------------+----------------------------------+---------+" << RESET <<
endl << endl;
              }
              currentCategory = category;
              cout << YELLOW << "| Category: " << left << setw(39) << category << "             |" << RESET
<< endl;
              cout << CYAN << "+------------------+----------------------------------+---------+" << RESET << endl;
              cout << GREEN << "| Item ID          | Item Name                        |Price(RM)|" << RESET <<
endl;
              cout << CYAN << "+------------------+----------------------------------+---------+" << RESET << endl;
           }

           int itemID = db.res->getInt("menu_item_id");
           string itemName = db.res->getString("item_name");
           double itemPrice = db.res->getDouble("item_price");

           // Format price to 2 decimal places
           stringstream priceStr;
           priceStr << fixed << setprecision(2) << itemPrice;

           cout << "| " << left << setw(18) << itemID
              << "| " << left << setw(35) << itemName
              << "| " << right << setw(7) << priceStr.str() << " |" << endl;
        }
        cout << CYAN << "+------------------+----------------------------------+---------+" << RESET << endl;

        // If no items were displayed
        if (currentCategory.empty()) {
           cout << RED << "\nNo menu items found! Please add items first." << RESET << endl;
           system("pause");
           return;
        }

        // Prompt user for Item ID to delete
        cout << YELLOW << "\nEnter the Item ID to delete or '0' to cancel: " << RESET;
        string input;
        cin >> input;

        if (input == "0") {
           cout << GREEN << "\nOperation canceled. Returning to Delete Menu..." << RESET << endl;
           system("pause");
           return;
        }
```

```
        int itemIDToDelete = stoi(input);

        // Confirm deletion
        cout << YELLOW << "\nAre you sure you want to delete the item with ID: " << itemIDToDelete
<< "?" << RESET << endl;
        cout << RED << "Confirm deletion? (y/n): " << RESET;
        char confirm;
        cin >> confirm;

        if (tolower(confirm) != 'y') {
            cout << GREEN << "\nOperation canceled. Returning to Delete Menu..." << RESET << endl;
            system("pause");
            return;
        }

        // Execute delete query
        query = "DELETE FROM menu_item WHERE menu_item_id = ?";
        db.prepareStatement(query);
        db.stmt->setInt(1, itemIDToDelete);
        db.QueryStatement();

        cout << GREEN << "\nItem deleted successfully!" << RESET << endl;
        system("pause");
        return;

      }
      catch (exception& e) {
        cerr << RED << "\nError: " << e.what() << RESET << endl;
        system("pause");
        return;
      }
    }
}

void deleteEntireCategory(DBConnection& db) {
    while (true) {
      try {
        system("cls");
        cout << CYAN << "+==================================================+" << RESET <<
endl;
        cout << CYAN << "|              DELETE ENTIRE CATEGORY            |" << RESET << endl;
        cout << CYAN << "+==================================================+" << RESET <<
endl;

        string query = "SELECT DISTINCT category FROM menu_item ORDER BY category";
        db.prepareStatement(query);
        db.QueryResult();

        vector<string> categories;
        cout << YELLOW << "\nSelect a category to delete:" << RESET << endl;
        int index = 1;

        while (db.res->next()) {
            string category = db.res->getString("category");
            categories.push_back(category);
            cout << GREEN << index++ << ". " << category << RESET << endl;
```

```
        }

        if (categories.empty()) {
            cout << RED << "\nNo categories found! Please add items first." << RESET << endl;
            system("pause");
            return;
        }

        cout << RED << "\n0. Cancel and Return to Delete Menu\n" << RESET;
        cout << YELLOW << "\nEnter category number: " << RESET;
        string categoryInput;
        cin >> categoryInput;

        if (categoryInput == "0") {
            return; // Go back to the main menu without pause
        }

        int categoryChoice = stoi(categoryInput);
        if (categoryChoice < 1 || categoryChoice > categories.size()) {
            cout << RED << "\nInvalid category choice. Please try again." << RESET << endl;
            system("pause");
            continue;
        }

        string selectedCategory = categories[categoryChoice - 1];
        cout << YELLOW << "\nAre you sure you want to delete the entire category: " <<
selectedCategory << "?" << RESET << endl;
        cout << RED << "Confirm deletion? (y/n): " << RESET;
        char confirm;
        cin >> confirm;

        if (tolower(confirm) != 'y') {
            cout << GREEN << "Operation canceled. Returning to Delete Menu..." << RESET << endl;
            continue;
        }

        // Execute delete query
        query = "DELETE FROM menu_item WHERE category = ?";
        db.prepareStatement(query);
        db.stmt->setString(1, selectedCategory);
        db.QueryStatement();

        cout << GREEN << "\nCategory deleted successfully!" << RESET << endl;
        system("pause");
        return;
      }
      catch (exception& e) {
        cerr << RED << "\nError: " << e.what() << RESET << endl;
        system("pause");
        return; // Exit on error
      }
    }
}

// Function to register a new customer
void registerCustomer(DBConnection& db) {
    string name, email, password, phone_number, birthday;
```

```
    int customerID;

    while (true) {
        system("cls"); // Clear the screen
        cout << CYAN << "+--------------------------------------------------------------------------------+\n" << RESET;
        cout << CYAN << "|                              REGISTER                                          |\n" << RESET;
        cout << CYAN << "+--------------------------------------------------------------------------------+\n" << RESET;
        cout << YELLOW << "Note: You can type '0' at any time to cancel and return to the main
menu.\n\n" << RESET;

        try {
            // Collect name
            cout << BRIGHT_GREEN << "Enter your name: " << RESET;
            cin.ignore();
            getline(cin, name);

            if (name == "0") {
                cout << YELLOW << "Cancelling registration...\n" << RESET;
                system("pause");
                return;
            }

            // Validate email
            while (true) {
                cout << BRIGHT_GREEN << "Enter your email: " << RESET;
                cin >> email;

                if (email == "0") {
                    cout << YELLOW << "Cancelling registration...\n" << RESET;
                    system("pause");
                    return;
                }

                regex emailRegex(R"((\w+)(\.{1}\w+)*@(\w+)(\.\w{2,3})+)");
                if (regex_match(email, emailRegex)) {
                    break;
                }
                else {
                    cout << RED << "Invalid email format. Please try again.\n" << RESET;
                }
            }

            // Validate phone number
            while (true) {
                cout << BRIGHT_GREEN << "Enter your phone number: " << RESET;
                cin >> phone_number;

                if (phone_number == "0") {
                    cout << YELLOW << "Cancelling registration...\n" << RESET;
                    system("pause");
                    return;
                }

                regex phoneRegex(R"(^\d{7,15}$)");
                if (regex_match(phone_number, phoneRegex)) {
                    break;
                }
```

```
        else {
            cout << RED << "Invalid phone number. Please enter 7-15 digits.\n" << RESET;
        }
    }

    // Validate birthday
    while (true) {
        cout << BRIGHT_GREEN << "Enter your birthday (YYYY-MM-DD): " << RESET;
        cin >> birthday;

        if (birthday == "0") {
            cout << YELLOW << "Cancelling registration...\n" << RESET;
            system("pause");
            return;
        }

        regex birthdayRegex(R"(^\d{4}-(0[1-9]|1[0-2])-(0[1-9]|[12]\d|3[01])$)");
        if (regex_match(birthday, birthdayRegex)) {
            break;
        }
        else {
            cout << RED << "Invalid birthday format. Please use YYYY-MM-DD.\n" << RESET;
        }
    }

    // Validate password
    while (true) {
        cout << BRIGHT_GREEN << "Enter your password: " << RESET;
        password = "";
        char ch;
        while ((ch = _getch()) != 13) { // 13 is Enter key
            if (ch == 8 && !password.empty()) { // Backspace
                password.pop_back();
                cout << "\b \b";
            }
            else if (ch != 8) {
                password += ch;
                cout << "*";
            }
        }
        cout << "\n";

        if (password == "0") {
            cout << YELLOW << "Cancelling registration...\n" << RESET;
            system("pause");
            return;
        }

        regex passwordRegex(R"(^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{8,}$)");
        if (regex_match(password, passwordRegex)) {
            break;
        }
        else {
            cout << RED << "Password must be at least 8 characters with an uppercase, digit, and
special character.\n" << RESET;
        }
```

```
        }

        // Check for duplicate email
        db.prepareStatement("SELECT COUNT(*) AS count FROM customer WHERE customer_email
= ?");
        db.stmt->setString(1, email);
        db.QueryResult();

        if (db.res->next() && db.res->getInt("count") > 0) {
            cout << RED << "This email is already registered. Please use a different email.\n" << RESET;
            system("pause");
            return;
        }

        // Retrieve next available customer ID
        db.prepareStatement("SELECT MAX(customer_id) AS max_id FROM customer");
        db.QueryResult();

        if (db.res->next()) {
            customerID = db.res->getInt("max_id") + 1;
        }
        else {
            customerID = 1;
        }

        // Insert new customer
        db.prepareStatement("INSERT INTO customer (customer_id, customer_name, customer_email,
customer_password, customer_phone_number, customer_birthday, customer_registration_date)
VALUES (?, ?, ?, ?, ?, ?, NOW())");
        db.stmt->setInt(1, customerID);
        db.stmt->setString(2, name);
        db.stmt->setString(3, email);
        db.stmt->setString(4, password);
        db.stmt->setString(5, phone_number);
        db.stmt->setString(6, birthday);

        db.QueryStatement();

        cout << BRIGHT_GREEN << "Registration successful! Your customer ID is: " << customerID <<
RESET << endl;
        system("pause");
        return;
    }
    catch (sql::SQLException& e) {
        cerr << RED << "Database error: " << e.what() << RESET << endl;
    }
    catch (exception& e) {
        cerr << RED << "Error: " << e.what() << RESET << endl;
    }

    system("pause");
  }
}

// Function to remove a customer
void removeCustomer(DBConnection& db) {
  try {
```

```
    string customerID;
    char confirm;

    while (true) {
        system("cls"); // Clear screen for a clean interface

        cout << CYAN
            << "+-----------------------------------------------------------------------------------------------+\n"
            << "|                                REMOVE CUSTOMER                                                |\n"
            << "+-----------------------------------------------------------------------------------------------+\n"
            << RESET;
        cout << "Note: Type '0' at any time to return to the main menu.\n\n";

        // View and refresh the customer list
        viewCustomerList(db);

        // Prompt for customer ID
        cout << GREEN << "\nEnter the Customer ID to remove: " << RESET;
        cin >> customerID;

        if (customerID == "0") {
            cout << YELLOW << "\nReturning to the main menu...\n" << RESET;
            return;
        }

        // Validate Customer ID (numeric check)
        regex idRegex(R"(^\d+$)");
        if (!regex_match(customerID, idRegex)) {
            cout << RED << "\n[Error] Invalid Customer ID. Please enter a valid numeric ID.\n" << RESET;
            system("pause");
            continue;
        }

        // Check if the customer exists in the database
        db.prepareStatement("SELECT COUNT(*) AS count FROM customer WHERE customer_id = ?");
        db.stmt->setString(1, customerID);
        db.QueryResult();

        if (db.res->next() && db.res->getInt("count") == 0) {
            cout << RED << "\n[Error] Customer not found. Please check the ID and try again.\n" <<
RESET;
            system("pause");
            continue;
        }

        // Confirm removal
        cout << YELLOW << "\nAre you sure you want to remove this customer? This action cannot be
undone. (y/n): " << RESET;
        cin >> confirm;
        if (tolower(confirm) != 'y') {
            cout << YELLOW << "\nCustomer removal canceled.\n" << RESET;
            system("pause");
            continue;
        }

        // Delete the customer from the database
        db.prepareStatement("DELETE FROM customer WHERE customer_id = ?");
```

```
            db.stmt->setString(1, customerID);
            db.QueryStatement();

            cout << GREEN
                 << "\n+-------------------------------------+\n"
                 << "|  Customer removed successfully!  |\n"
                 << "+-------------------------------------+\n"
                 << RESET;

            system("pause"); // Pause to show success message
        }
    }
    catch (const sql::SQLException& e) {
        cerr << RED << "\n[Error] SQL Error: " << e.what() << RESET << endl;
    }
    catch (const exception& e) {
        cerr << RED << "\n[Error] General Error: " << e.what() << RESET << endl;
    }
}

// Function to view menu
void viewMenu(DBConnection& db) {
    try {
        while (true) {
            system("cls");

            // Header for category selection
            cout << CYAN << "+================================================+" << RESET <<
endl;
            cout << CYAN << "|              MENU CATEGORIES               |" << RESET << endl;
            cout << CYAN << "+================================================+" << RESET <<
endl;

            // Step 1: Fetch categories
            string query = "SELECT DISTINCT category FROM menu_item ORDER BY category";
            db.prepareStatement(query);
            db.QueryResult();

            vector<string> categories;

            // Display options with color coding
            cout << BRIGHT_GREEN << "| " << left << setw(46) << "1 . Overall (View All Menu Items)" << "
|" << RESET << endl;

            int index = 2;
            while (db.res->next()) {
                string category = db.res->getString("category");
                categories.push_back(category);
                cout << BRIGHT_GREEN << "| " << left << setw(2) << index << ". "
                     << left << setw(43) << category << " |" << RESET << endl;
                index++;
            }

            cout << BRIGHT_RED << "| " << left << setw(46) << "0 . Go Back" << "  |" << RESET << endl;
            cout << CYAN << "+================================================+" << RESET <<
endl;
```

```
        if (categories.empty()) {
            displayMessageBox("No categories available!", RED);
            return;
        }

        cout << YELLOW << "Enter your choice: " << RESET;
        int choice;
        cin >> choice;

        if (cin.fail() || choice < 0 || choice > categories.size() + 1) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            displayMessageBox("Invalid choice! Please select a valid option.", RED);
            continue;
        }

        if (choice == 0) return;

        system("cls");

        if (choice == 1) {
            // View all items
            cout << BLUE <<
"+=====================================================================+" << RESET <<
endl;
            cout << BLUE << "|                    COMPLETE MENU                    |" << RESET << endl;
            cout << BLUE <<
"+=====================================================================+" << RESET <<
endl;

            query = "SELECT category, menu_item_id, item_name, item_price FROM menu_item ORDER
BY category, item_price";
            db.prepareStatement(query);
            db.QueryResult();

            string currentCategory = "";
            while (db.res->next()) {
                string category = db.res->getString("category");
                if (category != currentCategory) {
                    if (!currentCategory.empty()) {
                        cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET <<
endl << endl;
                    }
                    currentCategory = category;
                    cout << YELLOW << "| Category: " << left << setw(39) << category << "             |" <<
RESET << endl;
                    cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET <<
endl;
                    cout << GREEN << "| Item ID          | Item Name                         |Price(RM)|" << RESET <<
endl;
                    cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET <<
endl;
                }

                int itemID = db.res->getInt("menu_item_id");
                string itemName = db.res->getString("item_name");
                double itemPrice = db.res->getDouble("item_price");
```

```
            // Format price to 2 decimal places
            stringstream priceStr;
            priceStr << fixed << setprecision(2) << itemPrice;

            cout << "| " << left << setw(18) << itemID
               << "| " << left << setw(35) << itemName
               << "| " << right << setw(7) << priceStr.str() << " |" << endl;
         }
         cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET << endl;
      }
      else {
         // View specific category
         string selectedCategory = categories[choice - 2];

         cout << BLUE <<
"+=================================================================+" << RESET <<
endl;
         cout << BLUE << "|                    COMPLETE MENU                        |" << RESET << endl;
         cout << BLUE <<
"+=================================================================+" << RESET <<
endl;
         cout << YELLOW << "| Category: " << left << setw(39) << selectedCategory << "              |" <<
RESET << endl;
         cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET << endl;
         cout << GREEN << "| Item ID          | Item Name                         |Price(RM)|" << RESET << endl;
         cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET << endl;

         query = "SELECT menu_item_id, item_name, item_price FROM menu_item WHERE category
= ? ORDER BY item_price";
         db.prepareStatement(query);
         db.stmt->setString(1, selectedCategory);
         db.QueryResult();

         bool hasItems = false;
         while (db.res->next()) {
            hasItems = true;
            int itemID = db.res->getInt("menu_item_id");
            string itemName = db.res->getString("item_name");
            double price = db.res->getDouble("item_price");

            // Format price to 2 decimal places
            stringstream priceStr;
            priceStr << fixed << setprecision(2) << price;

            cout << "| " << left << setw(18) << itemID
               << "| " << left << setw(35) << itemName
               << "| " << right << setw(7) << priceStr.str() << " |" << endl;
         }

         if (!hasItems) {
            cout << RED << "| No items found in this category.                  |" << RESET << endl;
         }

         cout << CYAN << "+------------------+-----------------------------------+---------+" << RESET << endl;
      }
```

```
        cout << "\n" << YELLOW << "Press any key to return to the menu selection..." << RESET << endl;
        system("pause");
      }
    }
    catch (sql::SQLException& e) {
      displayMessageBox("Error fetching menu: " + string(e.what()), RED);
    }
}

// Function to view Order History
void viewOrderHistory(DBConnection& db, int customerID) {
    try {
      // Retrieve all orders made by the customer, sorted by `order_id`
      db.prepareStatement("SELECT order_id, order_date, total_amount, is_paid FROM `order` WHERE
customer_id = ? ORDER BY order_id ASC");
      db.stmt->setInt(1, customerID);
      db.QueryResult();
      sql::ResultSet* orderResult = db.res;

      // Check if no rows are returned
      if (!orderResult->rowsCount()) {
        cout << "\n=========================================\n";
        cout << BRIGHT_RED << "     No Order History Found          \n" << RESET;
        cout << "=========================================\n";
        return;
      }

      cout << BRIGHT_BLUE <<
"\n=====================================================================\n" << RESET;
      cout << BRIGHT_BLUE << "              Customer Order History              \n" << RESET;
      cout << BRIGHT_BLUE <<
"=====================================================================\n" << RESET;

      // Iterate through all orders
      int orderCount = 0; // To display `Order #` in sequence
      while (orderResult->next()) {
        orderCount++;
        int orderID = orderResult->getInt("order_id");
        string orderDate = orderResult->getString("order_date");
        double totalAmount = orderResult->getDouble("total_amount");
        bool isPaid = orderResult->getInt("is_paid") == 1;

        // Display order summary
        cout << YELLOW << "\n-----------------------------------------\n" << RESET;
        cout << YELLOW << " Order #" << orderCount << " (Order ID: " << orderID << ")\n" << RESET;
        cout << YELLOW << "-----------------------------------------\n" << RESET;
        cout << " Date         : " << orderDate << "\n";
        cout << " Payment Status: " << (isPaid ? "Paid" : "Unpaid") << "\n";

        // Fetch and display items for this order
        DBConnection detailsDb; // Create a separate connection for details
        detailsDb.prepareStatement(
          "SELECT menu_item.item_name, order_details.quantity, order_details.item_price, "
          "(order_details.quantity * order_details.item_price) AS subtotal "
          "FROM order_details "
          "INNER JOIN menu_item ON order_details.menu_item_id = menu_item.menu_item_id "
          "WHERE order_details.order_id = ?"
```

```
        );
        detailsDb.stmt->setInt(1, orderID);
        detailsDb.QueryResult();
        sql::ResultSet* detailsResult = detailsDb.res;

        cout << "\n Items in Order:\n";
        cout << CYAN << "+------------------------------+----------+-------------+-------------+\n" << RESET;
        cout << GREEN << "| Item Name                    | Quantity | Unit Price  | Subtotal    |\n" << RESET;
        cout << CYAN << "+------------------------------+----------+-------------+-------------+\n" << RESET;

        while (detailsResult->next()) {
            string itemName = detailsResult->getString("item_name");
            int quantity = detailsResult->getInt("quantity");
            double unitPrice = detailsResult->getDouble("item_price");
            double subtotal = detailsResult->getDouble("subtotal");

            cout << "| " << setw(28) << left << itemName
                 << "| " << setw(9) << right << quantity
                 << "| RM" << setw(10) << fixed << setprecision(2) << unitPrice
                 << "| RM" << setw(9) << fixed << setprecision(2) << subtotal << " |\n";
        }
        cout << CYAN << "+------------------------------+----------+-------------+-------------+\n" << RESET;

        // Display total amount for the order
        cout << "| " << setw(28) << left << "Total Amount"
             << "| " << setw(9) << right << " "
             << "| " << setw(12) << " "
             << "| RM" << setw(9) << fixed << setprecision(2) << totalAmount << " |\n";
        cout << CYAN << "+------------------------------+----------+-------------+-------------+\n" << RESET;
    }

    // Summary
    cout << CYAN << "\n==================================\n" << RESET;
    cout << " Total Orders: " << orderCount << "\n";
    cout << CYAN << "==================================\n" << RESET;

  }
  catch (sql::SQLException& e) {
    cerr << "Error fetching order history: " << e.what() << endl;
  }
}

// Function to place an order
void placeOrder(DBConnection& db, int customerID, vector<int> menuItems, vector<int> quantities) {
  if (menuItems.size() != quantities.size()) {
    cerr << "Error: Menu items and quantities mismatch." << endl;
    return;
  }

  try {
    // Step 1: Get or create an unpaid order
    int orderID = getOrCreateUnpaidOrder(db, customerID);
    double currentTotal = 0.0;

    // Step 2: Fetch current total amount from the order
    db.prepareStatement("SELECT total_amount FROM `order` WHERE order_id = ?");
    db.stmt->setInt(1, orderID);
```

```
        db.QueryResult();

        if (db.res->next()) {
            currentTotal = db.res->getDouble("total_amount");
        }

        double newItemsTotal = 0.0;

        // Step 3: Insert order details and calculate new total
        for (size_t i = 0; i < menuItems.size(); ++i) {
            int menuItemID = menuItems[i];
            int quantity = quantities[i];

            // Fetch item price
            db.prepareStatement("SELECT item_price FROM menu_item WHERE menu_item_id = ?");
            db.stmt->setInt(1, menuItemID);
            db.QueryResult();

            if (db.res->next()) {
                double itemPrice = db.res->getDouble("item_price");
                newItemsTotal += itemPrice * quantity;

                // Insert into order_details
                db.prepareStatement("INSERT INTO order_details (order_id, menu_item_id, quantity,
item_price) VALUES (?, ?, ?, ?)");
                db.stmt->setInt(1, orderID);
                db.stmt->setInt(2, menuItemID);
                db.stmt->setInt(3, quantity);
                db.stmt->setDouble(4, itemPrice);
                db.QueryStatement();
            }
            else {
                cerr << "Error: Menu item ID " << menuItemID << " not found." << endl;
            }
        }

        // Step 4: Update order total without applying a discount
        double finalTotal = currentTotal + newItemsTotal;

        db.prepareStatement("UPDATE `order` SET total_amount = ? WHERE order_id = ?");
        db.stmt->setDouble(1, finalTotal);
        db.stmt->setInt(2, orderID);
        db.QueryStatement();

        // Step 5: Display order summary
        cout << "Order placed successfully! Your order ID is: " << orderID << endl;
        cout << "Total Amount: RM" << fixed << setprecision(2) << finalTotal << endl;

    }
    catch (sql::SQLException& e) {
        cerr << "Error placing order: " << e.what() << endl;
    }
}

// Function to checks if there's an unpaid order for the user.
int getOrCreateUnpaidOrder(DBConnection& db, int customerID) {
    // Step 1: Check for an unpaid order
```

```
   db.prepareStatement("SELECT order_id FROM `order` WHERE customer_id = ? AND is_paid = 0");
   db.stmt->setInt(1, customerID);
   db.QueryResult();

   if (db.res->next()) {
      return db.res->getInt("order_id"); // Return the ID of the unpaid order
   }
   else {
      // Step 2: Create a new order if no unpaid order exists
      db.prepareStatement("INSERT INTO `order` (customer_id, order_date, is_paid, total_amount)
VALUES (?, NOW(), 0, 0)");
      db.stmt->setInt(1, customerID);
      db.QueryStatement();

      // Step 3: Retrieve the ID of the newly created order
      db.prepareStatement("SELECT LAST_INSERT_ID() AS order_id");
      db.QueryResult();
      db.res->next();
      return db.res->getInt("order_id");
   }
}

// Function payment
void payOrder(DBConnection& db) {
   while (true) {
      try {
         system("cls"); // Clear the console
         cout << "\nFetching updated Customer List...\n";

         // Fetch customer list with unpaid orders
         db.prepareStatement(
            "SELECT c.customer_id, c.customer_name, c.customer_email, c.customer_phone_number, "
            "c.customer_birthday, c.customer_registration_date, c.is_member, c.membership_points, "
            "SUM(CASE WHEN o.is_paid = 0 THEN 1 ELSE 0 END) AS unpaid_orders "
            "FROM customer c "
            "LEFT JOIN `order` o ON c.customer_id = o.customer_id "
            "GROUP BY c.customer_id, c.customer_name, c.customer_email,
c.customer_phone_number, "
            "c.customer_birthday, c.customer_registration_date, c.is_member, c.membership_points "
            "ORDER BY c.customer_id");
         db.QueryResult();

         // Display updated customer list
         cout << YELLOW << "\nCustomer List:\n" << RESET;
         cout << CYAN << "+------------+-----------------+----------------------------+----------------+------------+-----
----------------+--------+--------+----------------+\n" << RESET;
         cout << GREEN << "| CustomerID | Name            | Email                      | Phone Number    |
Birthday   | Registration       | Member | Points | Unpaid Orders  |\n" << RESET;
         cout << CYAN << "+------------+-----------------+----------------------------+----------------+------------+-----
----------------+--------+--------+----------------+\n" << RESET;

         while (db.res->next()) {
            int customerID = db.res->getInt("customer_id");
            string name = db.res->getString("customer_name");
            string email = db.res->getString("customer_email");
            string phoneNumber = db.res->getString("customer_phone_number");
            string birthday = db.res->getString("customer_birthday");
```

```
            string registrationDateTime = db.res->getString("customer_registration_date");
            bool isMember = db.res->getInt("is_member") == 1;
            int points = db.res->getInt("membership_points");
            int unpaidOrders = db.res->getInt("unpaid_orders");

            cout << "| " << setw(10) << customerID << " | "
                 << setw(16) << name << " | "
                 << setw(27) << email << " | "
                 << setw(15) << phoneNumber << " | "
                 << setw(10) << birthday << " | "
                 << setw(19) << registrationDateTime << " | "
                 << setw(6) << (isMember ? "Yes" : "No") << " | "
                 << setw(6) << points << " | "
                 << setw(14) << unpaidOrders << " |\n";
        }
        cout << CYAN << "+------------+-----------------+----------------------------+-----------------+------------+-----
----------------+--------+--------+----------------+\n" << RESET;

        // Prompt for customer ID
        int customerID;
        while (true) {
            cout << GREEN << "\nEnter customer ID (or 0 to go back): " << RESET;
            cin >> customerID;

            // Validate input
            if (cin.fail()) {
                cin.clear(); // Clear the error state
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
                cout << RED << "Invalid input! Please enter a valid numeric Customer ID.\n" << RESET;
            }
            else {
                break; // Valid input, exit loop
            }
        }

        if (customerID == 0) {
            cout << GREEN << "\nReturning to the main menu...\n" << RESET;
            return; // Exit function to go back
        }

        // Fetch unpaid orders for the selected customer
        db.prepareStatement("SELECT order_id, total_amount, is_paid FROM `order` WHERE
customer_id = ? AND is_paid = 0");
        db.stmt->setInt(1, customerID);
        db.QueryResult();

        if (db.res->next()) {
            int orderID = db.res->getInt("order_id");
            double totalAmount = db.res->getDouble("total_amount");

            // Fetch customer details
            bool isMember = false;
            int currentPoints = 0;
            string birthday;

            db.prepareStatement("SELECT is_member, membership_points, customer_birthday FROM
customer WHERE customer_id = ?");
```

```
db.stmt->setInt(1, customerID);
db.QueryResult();

if (db.res->next()) {
   isMember = db.res->getInt("is_member") == 1;
   currentPoints = db.res->getInt("membership_points");
   birthday = db.res->getString("customer_birthday");
}

// Calculate discounts and final amount
double memberDiscount = isMember ? 0.05 * totalAmount : 0.0;
double highAmountDiscount = totalAmount > 100 ? 0.10 * totalAmount : 0.0;
double birthdayDiscount = 0.0;
time_t t = time(nullptr);
struct tm now;
if (localtime_s(&now, &t) == 0) {
   string birthdayMonthDay = birthday.substr(5, 5); // MM-DD
   char todayMonthDay[6];
   strftime(todayMonthDay, sizeof(todayMonthDay), "%m-%d", &now);
   if (birthdayMonthDay == todayMonthDay) {
      birthdayDiscount = 0.15 * totalAmount;
   }
}

double totalDiscount = memberDiscount + highAmountDiscount + birthdayDiscount;
double finalAmount = totalAmount - totalDiscount;

// Display order details and discounts
cout << CYAN << "\n=========================================" << RESET << endl;
cout << CYAN << "              Order Details            " << RESET << endl;
cout << CYAN << "=========================================\n" << RESET;

cout << YELLOW << "Total amount before discount: " << RESET
   << "RM" << fixed << setprecision(2) << totalAmount << "\n\n";

if (memberDiscount > 0) {
   cout << GREEN << "Member discount applied (5%): " << RESET
      << "RM" << fixed << setprecision(2) << memberDiscount << "\n";
}
if (highAmountDiscount > 0) {
   cout << GREEN << "over RM100 discount applied (10%): " << RESET
      << "RM" << fixed << setprecision(2) << highAmountDiscount << "\n";
}
if (birthdayDiscount > 0) {
   cout << GREEN << "Birthday discount applied (15%): " << RESET
      << "RM" << fixed << setprecision(2) << birthdayDiscount << "\n";
}

cout << CYAN << "-----------------------------------------" << RESET << endl;

cout << BRIGHT_GREEN << "Total discount: " << RESET
   << "RM" << fixed << setprecision(2) << totalDiscount << "\n";
cout << YELLOW << "Final amount after discount: " << RESET
   << "RM" << fixed << setprecision(2) << finalAmount << "\n";

cout << CYAN << "=========================================\n" << RESET;
```

```
        // Prompt for point redemption
        int pointsToRedeem = 0;
        if (isMember && currentPoints > 0) {
            cout << GREEN << "\nCurrent membership points: " << RESET << currentPoints << "\n";
            cout << YELLOW << "Do you want to redeem points? (1 for Yes, 0 for No): " << RESET;

            int redeemChoice;
            cin >> redeemChoice;

            // Validate input for redeemChoice
            while (cin.fail() || (redeemChoice != 0 && redeemChoice != 1)) {
                cin.clear(); // Clear error state
                cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore invalid input
                cout << RED << "Invalid input! Please enter 1 for Yes or 0 for No: " << RESET;
                cin >> redeemChoice;
            }

            if (redeemChoice == 1) {
                cout << "Enter points to redeem (max " << currentPoints << "): ";
                cin >> pointsToRedeem;

                // Validate input for pointsToRedeem
                while (cin.fail() || pointsToRedeem < 0) {
                    cin.clear();
                    cin.ignore(numeric_limits<streamsize>::max(), '\n');
                    cout << RED << "Invalid input! Please enter a positive number (max " << currentPoints
<< "): " << RESET;
                    cin >> pointsToRedeem;
                }

                if (pointsToRedeem > currentPoints) {
                    pointsToRedeem = currentPoints;
                }
                if (pointsToRedeem > finalAmount) {
                    pointsToRedeem = static_cast<int>(finalAmount); // Cap points to the final amount
                }

                finalAmount -= pointsToRedeem;
                cout << "Points redeemed: " << pointsToRedeem << "\n";
                cout << "New final amount to pay: RM" << fixed << setprecision(2) << finalAmount <<
"\n";
            }
        }

        // Payment process
        cout << CYAN << "\nEnter payment amount: " << RESET;
        double paymentAmount;
        cin >> paymentAmount;

        // Validate input for paymentAmount
        while (cin.fail() || paymentAmount < 0) {
            cin.clear(); // Clear error state
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Ignore invalid input
            cout << RED << "Invalid input! Please enter a valid payment amount: " << RESET;
            cin >> paymentAmount;
        }
```

```
        // Define a tolerance for floating-point comparisons
        const double epsilon = 1e-2; // Adjust this as necessary for your currency precision (e.g., 0.01
for two decimal places)
        double change = round((paymentAmount - finalAmount) * 100) / 100;
        // Check if the payment amount is sufficient
        if (paymentAmount >= finalAmount - epsilon) {
          change = max(0.0, change);
          cout << GREEN << "\nPayment successful! Change to return: RM" << fixed <<
setprecision(2) << change << RESET << "\n";

          // Proceed with marking the order as paid and updating points
          db.prepareStatement("UPDATE `order` SET is_paid = 1 WHERE order_id = ?");
          db.stmt->setInt(1, orderID);
          db.QueryStatement();

          // Deduct redeemed points (if any)
          if (pointsToRedeem > 0) {
            db.prepareStatement("UPDATE customer SET membership_points = membership_points
- ? WHERE customer_id = ?");
            db.stmt->setInt(1, pointsToRedeem);
            db.stmt->setInt(2, customerID);
            db.QueryStatement();
          }

          // Add earned points
          int earnedPoints = static_cast<int>(finalAmount / 10); // RM 10 = 1 point
          if (earnedPoints > 0) {
            db.prepareStatement("UPDATE customer SET membership_points = membership_points
+ ? WHERE customer_id = ?");
            db.stmt->setInt(1, earnedPoints);
            db.stmt->setInt(2, customerID);
            db.QueryStatement();
            cout << "You earned " << earnedPoints << " points for this payment.\n";
          }
          // Prompt for receipt
          cout << YELLOW << "Do you want a receipt? (1 for Yes, 0 for No): " << RESET;
          int receiptChoice;
          cin >> receiptChoice;

          // Validate input for receiptChoice
          while (cin.fail() || (receiptChoice != 0 && receiptChoice != 1)) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << RED << "Invalid input! Please enter 1 for Yes or 0 for No: " << RESET;
            cin >> receiptChoice;
          }

          if (receiptChoice == 1) {
            // Generate receipt content
            string receiptContent = "============================\n";
            receiptContent += "        Receipt\n";
            receiptContent += "============================\n";
            receiptContent += "Customer ID: " + to_string(customerID) + "\n";
            receiptContent += "Order ID: " + to_string(orderID) + "\n";
            receiptContent += "Total Amount: RM" + to_string(totalAmount) + "\n";
            receiptContent += "Discounts Applied: RM" + to_string(totalDiscount) + "\n";
            receiptContent += "Final Amount Paid: RM" + to_string(finalAmount) + "\n";
```

```
            receiptContent += "Points Redeemed: " + to_string(pointsToRedeem) + "\n";
            receiptContent += "Points Earned: " + to_string(static_cast<int>(finalAmount / 10)) +
"\n";
            receiptContent += "Change Returned: RM" + to_string(change) + "\n";
            receiptContent += "============================\n";
            receiptContent += "     Thank you for your payment!\n";

            // Save receipt to a file
            string filename = "receipt_" + to_string(orderID) + ".txt";
            ofstream receiptFile(filename);

            if (receiptFile.is_open()) {
                receiptFile << "============================\n";
                receiptFile << "         Receipt         \n";
                receiptFile << "============================\n";
                receiptFile << "Customer ID: " << customerID << "\n";
                receiptFile << "Order ID: " << orderID << "\n";
                receiptFile << fixed << setprecision(2); // Ensure two decimal places
                receiptFile << "Total Amount: RM" << totalAmount << "\n";

                // Display discounts
                receiptFile << "Discounts Applied:\n";
                if (memberDiscount > 0) {
                    receiptFile << " - Member Discount (5%): RM" << memberDiscount << "\n";
                }
                if (highAmountDiscount > 0) {
                    receiptFile << " - Over RM100 Discount (10%): RM" << highAmountDiscount << "\n";
                }
                if (birthdayDiscount > 0) {
                    receiptFile << " - Birthday Discount (15%): RM" << birthdayDiscount << "\n";
                }
                receiptFile << "Total Discounts: RM" << totalDiscount << "\n";

                receiptFile << "Final Amount Paid: RM" << finalAmount << "\n";
                receiptFile << "Points Redeemed: " << pointsToRedeem << "\n";
                receiptFile << "Points Earned: " << earnedPoints << "\n";

                // Ensure no negative change is displayed
                double finalChange = max(0.0, change);
                receiptFile << "Change Returned: RM" << finalChange << "\n";

                receiptFile << "============================\n";
                receiptFile << "     Thank you for your payment!\n";
                receiptFile.close();
                cout << GREEN << "\nReceipt saved as: " << filename << RESET << endl;
            }
            else {
                cerr << RED << "\nError saving receipt to file!" << RESET << endl;
            }
        }
        else if (receiptChoice == 0) {
            cout << GREEN << "\nNo receipt will be generated.\n" << RESET;
        }
        else {
            cout << "ok";
        }
```

```
            // Fetch sales details and update the sales table
            db.prepareStatement("SELECT menu_item_id, quantity FROM order_details WHERE
order_id = ?");
            db.stmt->setInt(1, orderID);
            db.QueryResult();

            while (db.res->next()) {
               int menuItemID = db.res->getInt("menu_item_id");
               int quantity = db.res->getInt("quantity");

               // Fetch price per menu item
               db.prepareStatement("SELECT item_price FROM menu_item WHERE menu_item_id
= ?");
               db.stmt->setInt(1, menuItemID);
               db.QueryResult();

               double pricePerItem = 0.0;
               if (db.res->next()) {
                  pricePerItem = db.res->getDouble("item_price");
               }

               double revenue = pricePerItem * quantity; // Calculate revenue for this menu item

               // Update the sales table
               db.prepareStatement(
                  "INSERT INTO sales (menu_item_id, quantity_sales, revenue, sale_date, daily_sales,
monthly_sales, yearly_sales) "
                  "VALUES (?, ?, ?, NOW(), ?, ?, ?) "
                  "ON DUPLICATE KEY UPDATE "
                  "quantity_sales = quantity_sales + VALUES(quantity_sales), "
                  "revenue = revenue + VALUES(revenue), "
                  "daily_sales = CASE WHEN DATE(sale_date) = CURDATE() THEN daily_sales +
VALUES(quantity_sales) ELSE VALUES(quantity_sales) END, "
                  "monthly_sales = CASE WHEN MONTH(sale_date) = MONTH(CURDATE()) AND
YEAR(sale_date) = YEAR(CURDATE()) THEN monthly_sales + VALUES(quantity_sales) ELSE
VALUES(quantity_sales) END, "
                  "yearly_sales = CASE WHEN YEAR(sale_date) = YEAR(CURDATE()) THEN yearly_sales +
VALUES(quantity_sales) ELSE VALUES(quantity_sales) END"
               );
               db.stmt->setInt(1, menuItemID);
               db.stmt->setInt(2, quantity);
               db.stmt->setDouble(3, revenue);

               // Calculate daily, monthly, and yearly sales values for the current sale
               db.stmt->setInt(4, quantity); // Daily sales
               db.stmt->setInt(5, quantity); // Monthly sales
               db.stmt->setInt(6, quantity); // Yearly sales

               db.QueryStatement();
            }

         }
         else {
            cout << RED << "\nInsufficient payment amount. Please try again.\n" << RESET;
            system("pause");
         }
```

```
        }
        else {
            cout << RED << "\nNo unpaid orders found for this customer.\n" << RESET;
            system("pause");
        }

        // Ask user whether to process another payment
        cout << YELLOW << "\nDo you want to process another payment? (1 for Yes, 0 to go back): " <<
RESET;
        int choice;
        while (true) {
            cin >> choice;
            if (cin.fail() || (choice != 0 && choice != 1)) {
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << RED << "Invalid input! Please enter 1 for Yes or 0 to go back: " << RESET;
            }
            else {
                break;
            }
        }

        if (choice == 0) {
            cout << GREEN << "\nReturning to the main menu...\n" << RESET;
            return; // Exit function to go back
        }
    }
    catch (sql::SQLException& e) {
        cerr << RED << "\nDatabase error: " << e.what() << RESET << endl;
        system("pause");
    }
    catch (exception& e) {
        cerr << RED << "\nError: " << e.what() << RESET << endl;
        system("pause");
    }
  }
}

// Helper function for birthday discount
double calculateBirthdayDiscount(double totalAmount, const string& birthday) {
  time_t t = time(nullptr);
  struct tm now;
  if (localtime_s(&now, &t) != 0) {
    cerr << "Error: Failed to retrieve local time.\n";
    return 0.0;
  }

  string birthdayMonthDay = birthday.substr(5, 5); // "MM-DD"
  char todayMonthDay[6];
  strftime(todayMonthDay, sizeof(todayMonthDay), "%m-%d", &now);

  if (birthdayMonthDay == todayMonthDay) {
    cout << "\nHappy Birthday! You received an extra 15% discount!\n";
    return 0.15 * totalAmount;
  }

  return 0.0;
```

```
}

// Function to generate a sales report
void generateSalesReport(DBConnection& db, const string& timeframe, const string& date) {
   try {
      system("cls"); // Clear the console screen

      // Header for the sales report
      cout << CYAN <<
"+=================================================================================
===================================+\n" << RESET;
      cout << CYAN << "|                                    SALES REPORT                                    |\n"
<< RESET;
      cout << CYAN <<
"+=================================================================================
===================================+\n" << RESET;

      // Shop details (replace with your shop's information)
      cout << GREEN << "Shop Name: Sushi CHOOI JUN XIANG\n" << RESET;
      cout << GREEN << "Address  : 91,jalan indah 1/1\n" << RESET;
      cout << GREEN << "Contact  : 011-16182812\n" << RESET;

      // Display the timeframe and selected date
      cout << YELLOW << "Sales Report (" << timeframe << ")\n" << RESET;
      cout << YELLOW << "Date: " << date << "\n" << RESET;

      string query;
      if (timeframe == "daily") {
         query = "SELECT menu_item.menu_item_id, menu_item.item_name, SUM(sales.quantity_sales)
AS total_quantity, "
            "menu_item.item_price, SUM(sales.quantity_sales * menu_item.item_price) AS subtotal "
            "FROM sales "
            "JOIN menu_item ON sales.menu_item_id = menu_item.menu_item_id "
            "WHERE DATE(sales.sale_date) = ? "
            "GROUP BY menu_item.menu_item_id, menu_item.item_name, menu_item.item_price "
            "ORDER BY menu_item.item_name";
      }
      else if (timeframe == "monthly") {
         query = "SELECT menu_item.menu_item_id, menu_item.item_name, SUM(sales.quantity_sales)
AS total_quantity, "
            "menu_item.item_price, SUM(sales.quantity_sales * menu_item.item_price) AS subtotal "
            "FROM sales "
            "JOIN menu_item ON sales.menu_item_id = menu_item.menu_item_id "
            "WHERE YEAR(sales.sale_date) = YEAR(?) AND MONTH(sales.sale_date) = MONTH(?) "
            "GROUP BY menu_item.menu_item_id, menu_item.item_name, menu_item.item_price "
            "ORDER BY menu_item.item_name";
      }
      else if (timeframe == "yearly") {
         query = "SELECT menu_item.menu_item_id, menu_item.item_name, SUM(sales.quantity_sales)
AS total_quantity, "
            "menu_item.item_price, SUM(sales.quantity_sales * menu_item.item_price) AS subtotal "
            "FROM sales "
            "JOIN menu_item ON sales.menu_item_id = menu_item.menu_item_id "
            "WHERE YEAR(sales.sale_date) = YEAR(?) "
            "GROUP BY menu_item.menu_item_id, menu_item.item_name, menu_item.item_price "
            "ORDER BY menu_item.item_name";
      }
```

```
    db.prepareStatement(query);

    if (timeframe == "daily") {
        db.stmt->setString(1, date); // Bind the full date
    }
    else if (timeframe == "monthly") {
        db.stmt->setString(1, date + "-01"); // Bind the first day of the month (e.g., 2025-01-01)
        db.stmt->setString(2, date + "-01"); // Bind the same date for MONTH(?)
    }
    else if (timeframe == "yearly") {
        db.stmt->setString(1, date + "-01-01"); // Bind the first day of the year (e.g., 2025-01-01)
    }

    db.QueryResult();

    // Display table header
    cout << CYAN <<  "+----+--------------+-------------------------------------------------------------------------------+------------+------------+\n" << RESET;
    cout << GREEN << "| No | Menu Item ID | Item                                                                          | Quantity | Unit Price |  Subtotal  |\n" << RESET;
    cout << CYAN <<  "+----+--------------+-------------------------------------------------------------------------------+------------+------------+\n" << RESET;

    int no = 1;
    double total = 0.0;
    stringstream reportContent;

    // Add headers to the report content
    reportContent << "No,Menu Item ID,Item,Quantity,Unit Price,Subtotal\n";

    while (db.res->next()) {
        int menuItemId = db.res->getInt("menu_item_id");
        string itemName = db.res->getString("item_name");
        int totalQuantity = db.res->getInt("total_quantity");
        double unitPrice = db.res->getDouble("item_price");
        double subtotal = db.res->getDouble("subtotal");

        // Display each row
        cout << "| " << setw(2) << left << no << " | "
            << setw(12) << left << menuItemId << " | "
            << setw(70) << left << itemName << " | "
            << setw(10) << right << totalQuantity << " | "
            << setw(10) << fixed << setprecision(2) << unitPrice << " | "
            << setw(10) << fixed << setprecision(2) << subtotal << " |\n";

        // Add row to report content
        reportContent << no << "," << menuItemId << "," << itemName << "," << totalQuantity << ","
            << fixed << setprecision(2) << unitPrice << "," << fixed << setprecision(2) << subtotal << "\n";

        total += subtotal;
        no++;
    }

    // Table footer and total
```

```
        cout << CYAN << "+----+--------------+-------------------------------------------------------------------------------+---------
---+------------+------------+\n" << RESET;
        cout << GREEN <<"| TOTAL:                                                                                        | "
             << setw(11) << fixed << setprecision(2) << total << "|\n" << RESET;
        cout << CYAN << "+----+--------------+-------------------------------------------------------------------------------+---------
---+------------+------------+\n" << RESET;

        // Add total to the report content
        reportContent << ",,,,,Total," << total << "\n";

        // Ask user if they want to save the report
        int saveReport;
        cout << YELLOW << "Do you want to save this report? (1 for Yes, 0 for No): " << RESET;
        cin >> saveReport;

        if (saveReport == 1) {
            string filename;
            cout << YELLOW << "Enter filename to save (e.g., sales_report.csv): " << RESET;
            cin >> filename;

            ofstream outFile(filename);
            if (outFile.is_open()) {
                outFile << reportContent.str(); // Write report content to file
                outFile.close();
                cout << GREEN << "Sales report saved as: " << filename << RESET << endl;
            }
            else {
                cerr << RED << "Error: Unable to save report to file." << RESET << endl;
            }
        }
    }
    catch (sql::SQLException& e) {
        cerr << BRIGHT_RED << "\nError generating sales report: " << e.what() << RESET << endl;
    }
    catch (exception& e) {
        cerr << BRIGHT_RED << "\nAn error occurred: " << e.what() << RESET << endl;
    }
}

// Function to generate bar graph for sales
void generateSalesBarGraph(DBConnection& db) {
    try {
        while (true) {
            system("cls");
            cout << CYAN << "\n============================================\n";
            cout << "            SALES BAR GRAPH MENU         \n";
            cout << "============================================\n" << RESET;
            cout << BRIGHT_GREEN << "1. Revenue Comparison\n";
            cout << "2. Item Sales\n";
            cout << "0. Go Back to Main Menu\n" << RESET;
            cout << YELLOW << "\nEnter your choice: " << RESET;

            int graphType;
            cin >> graphType;

            if (cin.fail() || graphType < 0 || graphType > 2) { // Validate input
                cin.clear();
```

```
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    cout << ERROR << " Invalid choice! Please enter 1, 2, or 0.\n" << RESET;
    system("pause");
    continue;
}

if (graphType == 0) return;

// Common variables for query and data
string query;
vector<pair<string, double>> data;
double totalValue = 0.0;

if (graphType == 1) { // Revenue Comparison
    while (true) {
        system("cls");
        cout << CYAN << "\n=============================================\n";
        cout << "            REVENUE COMPARISON            \n";
        cout << "=============================================\n" << RESET;
        cout << BRIGHT_GREEN << "1. annually (2024 - Today)\n";
        cout << "2. Monthly for a Specific Year\n";
        cout << "3. Daily for a Specific Month\n";
        cout << "0. Go Back\n" << RESET;
        cout << YELLOW << "\nEnter your choice: " << RESET;

        int timeframe;
        cin >> timeframe;

        if (cin.fail() || timeframe < 0 || timeframe > 3) { // Validate input
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << ERROR << " Invalid choice! Please enter 1, 2, or 0.\n" << RESET;
            system("pause");
            continue;
        }

        if (timeframe == 0) break;

        if (timeframe == 1) {
            // Yearly Comparison
            query = "SELECT YEAR(sale_date) AS label, SUM(revenue) AS total_value "
                "FROM sales GROUP BY YEAR(sale_date) ORDER BY YEAR(sale_date)";
        }
        else if (timeframe == 2) {
            // Monthly for Specific Year
            cout << YELLOW << "\nEnter Year: " << RESET;
            int year;
            cin >> year;
            if (year < 2000 || year > 2100) {
                cout << BRIGHT_RED << "Invalid year! Please enter a valid year.\n" << RESET;
                system("pause");
                continue;
            }
            query = "SELECT MONTH(sale_date) AS label, SUM(revenue) AS total_value "
                "FROM sales WHERE YEAR(sale_date) = " + to_string(year) +
                " GROUP BY MONTH(sale_date) ORDER BY MONTH(sale_date)";
        }
```

```
            else if (timeframe == 3) {
                // Daily for Specific Month
                cout << YELLOW << "\nEnter Year: " << RESET;
                int year;
                cin >> year;
                cout << YELLOW << "Enter Month (1-12): " << RESET;
                int month;
                cin >> month;
                if (year < 2000 || year > 2100 || month < 1 || month > 12) {
                    cout << BRIGHT_RED << "Invalid input! Please enter valid year and month.\n" <<
RESET;
                    system("pause");
                    continue;
                }
                query = "SELECT DAY(sale_date) AS label, SUM(revenue) AS total_value "
                    "FROM sales WHERE YEAR(sale_date) = " + to_string(year) +
                    " AND MONTH(sale_date) = " + to_string(month) +
                    " GROUP BY DAY(sale_date) ORDER BY DAY(sale_date)";
            }
            else {
                cout << BRIGHT_RED << "\nInvalid choice!\n" << RESET;
                continue;
            }

            // Execute query and process results
            db.prepareStatement(query);
            db.QueryResult();
            data.clear();
            totalValue = 0.0;

            while (db.res->next()) {
                string label = to_string(db.res->getInt("label"));
                double value = db.res->getDouble("total_value");
                totalValue += value;
                data.emplace_back(label, value);
            }

            if (data.empty()) {
                cout << BRIGHT_RED << "\nNo data available for the selected timeframe.\n" << RESET;
            }
            else {
                // Display bar graph
                cout << CYAN <<
"\n================================================================================
================================================\n";
                cout <<       "                                    REVENUE BAR GRAPH
\n";
                cout <<
"================================================================================
================================================\n" << RESET;

                int maxBarLength = 50;
                double maxValue = max_element(data.begin(), data.end(), [](const auto& a, const auto&
b) {
                    return a.second < b.second;
                    })->second;
```

```
                cout << CYAN << "+----------------------------------------------------------------------------------------------------
-----------------------------------+\n" << RESET;
                for (const auto& entry : data) {
                    int barLength = static_cast<int>((entry.second / maxValue) * maxBarLength);
                    cout << YELLOW << setw(10) << entry.first << " | " << RESET;
                    // Draw the bar with colored blocks
                    for (int i = 0; i < barLength; i++) {
                        cout << GREEN_BG << " " << RESET; // Colored block for each unit
                    }
                    cout << " " << fixed << setprecision(2) << entry.second << "\n\n";
                }
                cout << CYAN << "+----------------------------------------------------------------------------------------------------
-----------------------------------+\n" << RESET;
                cout <<GREEN << "Total Revenue: " << RESET << fixed << setprecision(2) << totalValue
<< "\n";
            }
            system("pause");
        }
    }
    else if (graphType == 2) { // Item Sales
        while (true) {
            system("cls");
            cout << CYAN << "\n=============================================\n";
            cout << "              ITEM SALES               \n";
            cout << "=============================================\n" << RESET;
            cout << BRIGHT_GREEN << "1. annualy\n";
            cout << "2. Monthly\n";
            cout << "3. Daily\n";
            cout << "0. Go Back\n" << RESET;
            cout << YELLOW << "\nEnter your choice: " << RESET;

            int timeframe;
            cin >> timeframe;

            if (cin.fail() || timeframe < 0 || timeframe > 3) { // Validate input
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
                cout << ERROR << " Invalid choice! Please enter 1, 2, or 0.\n" << RESET;
                system("pause");
                continue;
            }

            if (timeframe == 0) break;

            if (timeframe == 1) {
                cout << YELLOW << "\nEnter Year: " << RESET;
                int year;
                cin >> year;
                query = "SELECT menu_item.item_name AS label, SUM(sales.quantity_sales) AS
total_value "
                    "FROM sales JOIN menu_item ON sales.menu_item_id = menu_item.menu_item_id "
                    "WHERE YEAR(sales.sale_date) = " + to_string(year) +
                    " GROUP BY menu_item.item_name ORDER BY total_value DESC";
            }
            else if (timeframe == 2) {
                cout << YELLOW << "\nEnter Year: " << RESET;
                int year;
```

```
            cin >> year;
            cout << YELLOW << "Enter Month (1-12): " << RESET;
            int month;
            cin >> month;
            query = "SELECT menu_item.item_name AS label, SUM(sales.quantity_sales) AS
total_value "
                "FROM sales JOIN menu_item ON sales.menu_item_id = menu_item.menu_item_id "
                "WHERE YEAR(sales.sale_date) = " + to_string(year) +
                " AND MONTH(sales.sale_date) = " + to_string(month) +
                " GROUP BY menu_item.item_name ORDER BY total_value DESC";
        }
        else if (timeframe == 3) {
            cout << YELLOW << "\nEnter Date (YYYY-MM-DD): " << RESET;
            string date;
            cin >> date;
            query = "SELECT menu_item.item_name AS label, SUM(sales.quantity_sales) AS
total_value "
                "FROM sales JOIN menu_item ON sales.menu_item_id = menu_item.menu_item_id "
                "WHERE DATE(sales.sale_date) = '" + date + "' "
                "GROUP BY menu_item.item_name ORDER BY total_value DESC";
        }
        else {
            cout << BRIGHT_RED << "\nInvalid choice!\n" << RESET;
            continue;
        }

        // Execute query and process results
        db.prepareStatement(query);
        db.QueryResult();
        data.clear();
        totalValue = 0.0;

        while (db.res->next()) {
            string label = db.res->getString("label");
            double value = db.res->getDouble("total_value");
            totalValue += value;
            data.emplace_back(label, value);
        }

        if (data.empty()) {
            cout << BRIGHT_RED << "\nNo data available for the selected timeframe.\n" << RESET;
        }
        else {
            // Display bar graph
            cout << CYAN <<
"\n================================================================================
================================================\n";
            cout <<       "                                    ITEM SALES BAR GRAPH
\n";
            cout <<
"================================================================================
================================================\n" << RESET;

            int maxBarLength = 50;
            double maxValue = max_element(data.begin(), data.end(), [](const auto& a, const auto&
b) {
                return a.second < b.second;
```

```
                    })->second;

                cout << CYAN << "+----------------------------------------------------------------------------------------------------
----------------------------------+\n" << RESET;
                for (const auto& entry : data) {
                    int barLength = static_cast<int>((entry.second / maxValue) * maxBarLength);
                    cout << YELLOW << setw(35) << entry.first << " | " << RESET;
                    // Draw the bar with colored blocks
                    for (int i = 0; i < barLength; i++) {
                        cout << GREEN_BG << " " << RESET; // Colored block for each unit
                    }
                    cout << " " << fixed << setprecision(2) << entry.second << "\n\n";
                }
                cout << CYAN << "+----------------------------------------------------------------------------------------------------
----------------------------------+\n" << RESET;
                cout << GREEN << "Total Sales: " << RESET << fixed << setprecision(2) << totalValue <<
"\n\n";
            }
            system("pause");
        }
    }
    else {
        cout << BRIGHT_RED << "\nInvalid choice!\n" << RESET;
    }
  }
  catch (const sql::SQLException& e) {
    cerr << BRIGHT_RED << "\nError generating sales bar graph: " << e.what() << RESET << endl;
  }
  catch (const exception& e) {
    cerr << BRIGHT_RED << "\nAn error occurred: " << e.what() << RESET << endl;
  }
}

// Helper function to convert month number to name
string getMonthName(int month) {
  const string months[] = { "January", "February", "March", "April", "May", "June",
                    "July", "August", "September", "October", "November", "December" };
  return months[month - 1];
}

// Function for staff to help customer register membership
void registerMembership(DBConnection& db) {
  bool continueRegister = true; // Control variable for the loop

  while (continueRegister) {
    system("cls"); // Clear the screen for a refreshed page
    viewCustomerList(db); // Display the current customer list

    try {
      int customerID;

      while (true) { // Input loop for Customer ID
        cout << YELLOW << "Enter the Customer ID to register as a member (-1 to cancel): " << RESET;
        if (cin >> customerID) {
          if (customerID == -1) { // Cancel option
            cout << YELLOW << "Registration process cancelled by the staff.\n" << RESET;
```

```
            return; // Exit the function
        }

        // Query to check if the customer exists
        db.prepareStatement("SELECT customer_name, is_member,
COALESCE(membership_points, 0) AS membership_points "
            "FROM customer WHERE customer_id = ?");
        db.stmt->setInt(1, customerID);
        db.QueryResult();

        if (db.res->next()) {
            break; // Valid Customer ID found
        }
        else {
            // Invalid Customer ID
            system("cls"); // Refresh the screen
            viewCustomerList(db); // Redisplay the customer list
            cout << RED << "Customer ID " << customerID << " not found. Please try again.\n" <<
RESET;
        }
      }
      else {
        cin.clear(); // Clear the error flag
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
        system("cls"); // Refresh the screen
        viewCustomerList(db); // Redisplay the customer list
        cout << RED << "Invalid input! Please enter a valid Customer ID (-1 to cancel).\n" << RESET;
      }
    }

    // Process membership registration
    string customerName = db.res->getString("customer_name");
    int isMember = db.res->getInt("is_member");
    int membershipPoints = db.res->getInt("membership_points");

    if (isMember == 1) {
      cout << RED << "Customer \"" << customerName << "\" is already a member with " <<
membershipPoints
          << " membership points.\n" << RESET;
    }
    else {
      char confirm;
      cout << GREEN << "Customer \"" << customerName << "\" is not yet a member. Register as a
member? (y/n): " << RESET;
      cin >> confirm;

      if (tolower(confirm) == 'y') {
        db.prepareStatement("UPDATE customer SET is_member = 1, membership_points = ?
WHERE customer_id = ?");
        db.stmt->setInt(1, membershipPoints);
        db.stmt->setInt(2, customerID);
        db.QueryStatement();

        cout << GREEN << "Customer \"" << customerName << "\" has been successfully registered
as a member with "
            << membershipPoints << " membership points.\n" << RESET;
      }
```

```
            else {
                cout << YELLOW << "Membership registration cancelled for \"" << customerName <<
"\".\n" << RESET;
            }
        }
    }
    catch (const sql::SQLException& e) {
        cerr << RED << "Error registering membership: " << e.what() << RESET << endl;
    }

    // Continuation prompt with invalid choice handling
    int choice;
    while (true) {
        cout << YELLOW << "Do you want to continue registering memberships? (1 = Yes, 0 = No): " <<
RESET;
        if (cin >> choice && (choice == 1 || choice == 0)) {
            continueRegister = (choice == 1);
            break; // Valid input
        }
        else {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << RED << "Invalid choice! Please enter 1 for Yes or 0 for No.\n" << RESET;
        }
    }
  }
}

// Function to remove membership from a customer
void removeMembership(DBConnection& db) {
  bool continueRemove = true; // Control variable for the loop

  while (continueRemove) {
    system("cls"); // Clear the screen for a refreshed page
    viewCustomerList(db); // Display the current customer list

    try {
        int customerID;

        while (true) { // Input loop for Customer ID
            cout << YELLOW << "Enter the Customer ID to remove membership (-1 to cancel): " << RESET;
            if (cin >> customerID) {
                if (customerID == -1) { // Cancel option
                    cout << YELLOW << "Membership removal process cancelled by the staff.\n" << RESET;
                    return; // Exit the function
                }

                // Query to check if the customer exists
                db.prepareStatement("SELECT customer_name, is_member FROM customer WHERE
customer_id = ?");
                db.stmt->setInt(1, customerID);
                db.QueryResult();

                if (db.res->next()) {
                    break; // Valid Customer ID found
                }
                else {
```

```
                // Invalid Customer ID
                system("cls"); // Refresh the screen
                viewCustomerList(db); // Redisplay the customer list
                cout << RED << "Customer ID " << customerID << " not found. Please try again.\n" <<
RESET;
            }
          }
          else {
            cin.clear(); // Clear the error flag
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Discard invalid input
            system("cls"); // Refresh the screen
            viewCustomerList(db); // Redisplay the customer list
            cout << RED << "Invalid input! Please enter a valid Customer ID (-1 to cancel).\n" << RESET;
          }
        }

        // Process membership removal
        string customerName = db.res->getString("customer_name");
        int isMember = db.res->getInt("is_member");

        if (isMember == 0) {
          cout << RED << "Customer \"" << customerName << "\" is not a member.\n" << RESET;
        }
        else {
          char confirm;
          cout << GREEN << "Customer \"" << customerName << "\" is currently a member. Remove
membership? (y/n): " << RESET;
          cin >> confirm;

          if (tolower(confirm) == 'y') {
            db.prepareStatement("UPDATE customer SET is_member = 0, membership_points = NULL
WHERE customer_id = ?");
            db.stmt->setInt(1, customerID);
            db.QueryStatement();

            cout << GREEN << "Membership for customer \"" << customerName << "\" has been
successfully removed.\n" << RESET;
          }
          else {
            cout << YELLOW << "Membership removal cancelled for \"" << customerName << "\".\n"
<< RESET;
          }
        }
      }
      catch (const sql::SQLException& e) {
        cerr << RED << "Error removing membership: " << e.what() << RESET << endl;
      }

      // Continuation prompt with invalid choice handling
      int choice;
      while (true) {
        cout << YELLOW << "Do you want to continue removing memberships? (1 = Yes, 0 = No): " <<
RESET;
        if (cin >> choice && (choice == 1 || choice == 0)) {
          continueRemove = (choice == 1);
          break; // Valid input
        }
```

```
        else {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << RED << "Invalid choice! Please enter 1 for Yes or 0 for No.\n" << RESET;
        }
    }
  }
}

// Function to calculate discount with birthday check
double calculateDiscount(double totalAmount, bool isMember, const string& birthday) {
  double discount = 0.0;

  // Apply 10% discount if the total amount is greater than 100
  if (totalAmount > 100) {
    discount += 0.10 * totalAmount;
  }

  // Add 5% discount for members
  if (isMember) {
    discount += 0.05 * totalAmount;
  }

  // Birthday discount logic
  time_t t = time(nullptr);
  struct tm now;
  if (localtime_s(&now, &t) != 0) {
    cerr << "Error: Failed to retrieve local time.\n";
    return discount;
  }

  // Extract month and day from birthday string (assuming YYYY-MM-DD format)
  string birthdayMonthDay = birthday.substr(5, 5);  // Gets "MM-DD"

  // Get today's month and day
  char todayMonthDay[6];
  strftime(todayMonthDay, sizeof(todayMonthDay), "%m-%d", &now);

  // Compare only month and day
  if (birthdayMonthDay == todayMonthDay) {
    cout << "\nHappy Birthday! You received an extra 15% discount!\n";
    discount += 0.15 * totalAmount;
  }

  return discount;
}

// Function for customer to check their profile
void viewProfile(DBConnection& db, int customerID) {
  try {
    // Query to fetch customer profile details including birthday
    db.prepareStatement(
      "SELECT customer_name, customer_email, customer_phone_number, is_member, "
      "membership_points, customer_registration_date, customer_birthday FROM customer WHERE
customer_id = ?"
    );
    db.stmt->setInt(1, customerID);
```

```
        db.QueryResult();

        if (db.res->next()) {
            string name = db.res->getString("customer_name");
            string email = db.res->getString("customer_email");
            string phone = db.res->getString("customer_phone_number");
            bool isMember = db.res->getInt("is_member") == 1;
            int points = db.res->getInt("membership_points");
            string regDate = db.res->getString("customer_registration_date");
            string birthday = db.res->getString("customer_birthday");

            // Query to fetch total orders and total spent
            db.prepareStatement(
                "SELECT COUNT(order_id) AS total_orders, IFNULL(SUM(total_amount), 0) AS total_spent "
                "FROM `order` WHERE customer_id = ?"
            );
            db.stmt->setInt(1, customerID);
            db.QueryResult();

            int totalOrders = 0;
            double totalSpent = 0.0;
            if (db.res->next()) {
                totalOrders = db.res->getInt("total_orders");
                totalSpent = db.res->getDouble("total_spent");
            }

            // Display the profile information
            cout << CYAN << "\n===== Customer Profile =====\n" << RESET;
            cout << "Name           : " << name << "\n";
            cout << "Email          : " << email << "\n";
            cout << "Phone Number    : " << phone << "\n";
            cout << "Membership      : " << (isMember ? "Yes" : "No") << "\n";
            cout << "Membership Points: " << points << "\n";
            cout << "Registration Date: " << regDate << "\n";
            cout << "Birthday        : " << birthday << "\n";
            cout << "Total Orders    : " << totalOrders << "\n";
            cout << "Total Spent (RM): RM " << fixed << setprecision(2) << totalSpent << "\n";
            cout << CYAN << "============================\n" << RESET;
        }
        else {
            cout << "Error: Customer profile not found.\n";
        }
    }
    catch (sql::SQLException& e) {
        cerr << "Error fetching profile: " << e.what() << endl;
    }
}

// Function manager menu
void managerMenu() {
    DBConnection db;
    int choice_manager_menu;

    do {
        cout << "\nManager Menu:\n"
            << "1. Add Staff\n"
            << "2. Update Staff Details\n"
```

```
            << "3. View Staff List\n"
            << "4. Logout\n"
            << "Enter your choice: ";
        cin >> choice_manager_menu;
        system("cls");

        switch (choice_manager_menu) {
        case 1:
            registerStaff(db);
            break;
        case 2:
            updateStaff(db);
            break;
        case 3:
            viewStaffList(db); // Optional: Add this function to view staff
            break;
        case 4:
            cout << "Logging out...\n";
            return; // Exit to main menu
        default:
            cout << "Invalid choice. Try again.\n";
        }
    } while (choice_manager_menu != 4);
}

//Add staff
void registerStaff(DBConnection& db) {
    try {
        string name, email, password, role, phoneNumber;
        int roleChoice;

        system("cls"); // Clear screen to provide a clean interface

        cout << CYAN
            << "+-----------------------------------------------------------------------------------------------+\n"
            << "|                                REGISTER NEW STAFF                                             |\n"
            << "+-----------------------------------------------------------------------------------------------+\n"
            << RESET;
        cout << "Note: You can type '0' at any time to return to the main menu.\n\n";

        // Collect staff details
        while (true) {
            cout << YELLOW << "Enter staff name: " << RESET;
            cin.ignore(); // Clear input buffer
            getline(cin, name);

            if (name == "0") {
                cout << YELLOW << "\nReturning to the main menu...\n" RESET;
                return;
            }

            if (!name.empty()) break;
            else cout << RED << "\n[Error] Name cannot be empty. Please try again.\n" RESET;
        }

        while (true) {
            cout << YELLOW << "Enter staff email: " << RESET;
```

```
        cin >> email;

        if (email == "back") {
            cout << YELLOW << "\nReturning to the main menu...\n" RESET;
            system("pause");
            return;
        }

        // Validate email format using regex
        regex emailRegex(R"((\w+)(\.{1}\w+)*@(\w+)(\.\w{2,3})+)");
        if (regex_match(email, emailRegex)) {
            // Check if email already exists in the database
            db.prepareStatement("SELECT COUNT(*) AS count FROM staff WHERE staff_email = ?");
            db.stmt->setString(1, email);
            db.QueryResult();

            if (db.res->next() && db.res->getInt("count") > 0) {
                cout << RED << "\n[Error] This email is already registered. Please use a different email.\n"
RESET;
            }
            else {
                break; // Valid and unique email
            }
        }
        else {
            cout << RED << "\n[Error] Invalid email format. Please use a valid email (e.g.,
user@example.com).\n" RESET;
        }
    }

    while (true) {
        cout << YELLOW << "Enter staff password (at least 8 characters, includes upper/lowercase,
digits, and special characters): " << RESET;
        cin >> password;

        if (password == "back") {
            cout << YELLOW << "\nReturning to the main menu...\n" RESET;
            system("pause");
            return;
        }

        // Password strength validation using regex
        regex passwordRegex(R"(^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@$!%*?&])[A-Za-
z\d@$!%*?&]{8,}$)");
        if (regex_match(password, passwordRegex)) {
            break; // Valid password
        }
        else {
            cout << RED << "\n[Error] Password must be at least 8 characters long, with at least one
uppercase letter, one digit, and one special character.\n" RESET;
        }
    }

    while (true) {
        cout << YELLOW << "Enter staff phone number (7 to 15 digits): " << RESET;
        cin >> phoneNumber;
```

```
        if (phoneNumber == "back") {
            cout << YELLOW << "\nReturning to the main menu...\n" RESET;
            system("pause");
            return;
        }

        // Phone number validation using regex
        regex phoneRegex(R"(^\d{7,15}$)");
        if (regex_match(phoneNumber, phoneRegex)) {
            break; // Valid phone number
        }
        else {
            cout << RED << "\n[Error] Invalid phone number. Please enter a number with 7 to 15
digits.\n" RESET;
        }
    }

    while (true) {
        cout << YELLOW
            << "\nSelect staff role:\n" << RESET
            <<CYAN<< "[1] Staff\n"
            << "[2] Manager\n"
            << RESET;
        cout << YELLOW << "Enter choice: " << RESET;
        cin >> roleChoice;

        if (roleChoice == 1) {
            role = "staff";
            break;
        }
        else if (roleChoice == 2) {
            role = "manager";
            break;
        }
        else {
            cout << RED << "\n[Error] Invalid role choice. Please enter 1 for Staff or 2 for Manager.\n"
RESET;
        }
    }

    // Insert the new staff record into the database
    db.prepareStatement("INSERT INTO staff (staff_name, staff_email, staff_password, staff_role,
staff_phone_number, staff_hire_date) "
        "VALUES (?, ?, ?, ?, ?, NOW())");
    db.stmt->setString(1, name);
    db.stmt->setString(2, email);
    db.stmt->setString(3, password); // Store plain password (consider hashing in production)
    db.stmt->setString(4, role);
    db.stmt->setString(5, phoneNumber);
    db.QueryStatement();

    cout << GREEN
        << "\n+--------------------------------------+\n"
        << "|  Staff member registered SUCCESS!   |\n"
        << "+--------------------------------------+\n"
        << RESET;
```

```
        // Retrieve and display the details of the newly added staff
        db.prepareStatement("SELECT staff_id, staff_name, staff_email, staff_role, staff_phone_number,
staff_hire_date "
            "FROM staff WHERE staff_id = LAST_INSERT_ID()");
        db.QueryResult();

        if (db.res->next()) {
            cout << YELLOW << "\nDetails of the Newly Added Staff:\n" << RESET;
            cout << CYAN << "+---------+----------------------+-------------------------+-----------+---------------------+--
-----------------+\n" << RESET;
            cout << GREEN << "| StaffID | Name                 | Email                   | Role      | Phone Number
| Hire Date        |\n" << RESET;
            cout << CYAN << "+---------+----------------------+-------------------------+-----------+---------------------+--
-----------------+\n" << RESET;

            int staffID = db.res->getInt("staff_id");
            string staffName = db.res->getString("staff_name");
            string staffEmail = db.res->getString("staff_email");
            string staffRole = db.res->getString("staff_role");
            string staffPhone = db.res->getString("staff_phone_number");
            string hireDate = db.res->getString("staff_hire_date");

            cout << "| " << setw(7) << staffID << " | "
                << setw(21) << staffName << " | "
                << setw(24) << staffEmail << " | "
                << setw(9) << staffRole << " | "
                << setw(20) << staffPhone << " | "
                << setw(17) << hireDate << " |\n";

            cout << CYAN << "+---------+----------------------+-------------------------+-----------+---------------------+--
-----------------+\n" << RESET;
        }
        else {
            cerr << RED << "\n[Error] Failed to retrieve details of the newly added staff.\n" RESET;
        }
    }
    catch (const sql::SQLException& e) {
        cerr << RED << "\n[Error] SQL Error: " << e.what() << RESET << endl;
    }
    catch (const exception& e) {
        cerr << RED << "\n[Error] General Error: " << e.what() << RESET << endl;
    }
}

//Remove Staff
void removeStaff(DBConnection& db) {
    try {
        string identifier;

        system("cls"); // Clear screen to provide a clean interface

        cout << CYAN
            << "+-----------------------------------------------------------------------------------------+\n"
            << "|                                  REMOVE STAFF                                           |\n"
            << "+-----------------------------------------------------------------------------------------+\n"
            << RESET;
        cout << "Note: Enter '0' at any time to return to the main menu.\n\n";
```

```
    while (true) {
        // View staff list for user reference
        viewStaffList(db);

        cout << GREEN << "Enter staff ID to remove (0 to return): " << RESET;
        cin >> identifier;

        if (identifier == "0") {
            cout << YELLOW << "Returning to the main menu...\n" << RESET;
            return; // No redundant pause
        }

        // Validate staff ID (numeric check)
        regex idRegex(R"(^\d+$)");
        if (regex_match(identifier, idRegex)) {
            // Check if the staff member exists in the database
            db.prepareStatement("SELECT COUNT(*) AS count FROM staff WHERE staff_id = ?");
            db.stmt->setString(1, identifier);
            db.QueryResult();

            if (db.res->next() && db.res->getInt("count") == 0) {
                cout << RED << "\n[Error] Staff member not found. Please check the staff ID and try
again.\n" << RESET;
            }
            else {
                // Confirm removal
                cout << YELLOW << "\nAre you sure you want to remove this staff member? This action
cannot be undone. (y/n): " << RESET;
                char confirm;
                cin >> confirm;
                if (tolower(confirm) == 'y') {
                    // Delete the staff member from the database
                    db.prepareStatement("DELETE FROM staff WHERE staff_id = ?");
                    db.stmt->setString(1, identifier);
                    db.QueryStatement();

                    cout << GREEN
                        << "\n+----------------------------------+\n"
                        << "|  Staff member removed SUCCESS! |\n"
                        << "+----------------------------------+\n"
                        << RESET;
                }
                else {
                    cout << YELLOW << "\nStaff removal canceled.\n" << RESET;
                }
            }
        }
        else {
            cout << RED << "\n[Error] Invalid staff ID. Please enter a valid numeric ID or '0' to return.\n"
<< RESET;
        }
    }
  }
  catch (const sql::SQLException& e) {
    cerr << RED << "\n[Error] SQL Error: " << e.what() << RESET << endl;
  }
```

```
    catch (const exception& e) {
        cerr << RED << "\n[Error] General Error: " << e.what() << RESET << endl;
    }
}

//Modify Staff
void updateStaff(DBConnection& db) {
    while (true) {
        try {
            // Step 1: Display the staff list
            system("cls");
            cout << CYAN << "+=================================================+" << RESET <<
endl;
            cout << CYAN << "|                UPDATE STAFF                |" << RESET << endl;
            cout << CYAN << "+=================================================+" << RESET <<
endl;

            cout << YELLOW << "\nCurrent Staff List:" << RESET << endl;
            cout << CYAN << "+---------+----------------------+-----------------------+---------+----------------+" <<
RESET << endl;
            cout << GREEN << "| StaffID | Name                 | Email                 | Role    | Phone Number   |"
<< RESET << endl;
            cout << CYAN << "+---------+----------------------+-----------------------+---------+----------------+" <<
RESET << endl;

            db.prepareStatement("SELECT staff_id, staff_name, staff_email, staff_role,
staff_phone_number FROM staff ORDER BY staff_id");
            db.QueryResult();

            while (db.res->next()) {
                int staffID = db.res->getInt("staff_id");
                string name = db.res->getString("staff_name");
                string email = db.res->getString("staff_email");
                string role = db.res->getString("staff_role");
                string phoneNumber = db.res->getString("staff_phone_number");

                cout << "| " << setw(7) << staffID << " | "
                    << setw(21) << name << " | "
                    << setw(23) << email << " | "
                    << setw(7) << role << " | "
                    << setw(14) << phoneNumber << " |" << endl;
            }
            cout << CYAN << "+---------+----------------------+-----------------------+---------+----------------+" <<
RESET << endl;

            cout << RED << "\n0. Cancel and Return to Staff Menu\n" << RESET;
            cout << YELLOW << "\nEnter the Staff ID to update: " << RESET;
            string input;
            cin >> input;

            if (input == "0") {
                return;
            }

            int staffID = stoi(input);

            // Validate staff ID
```

```
        db.prepareStatement("SELECT * FROM staff WHERE staff_id = ?");
        db.stmt->setInt(1, staffID);
        db.QueryResult();

        if (!db.res->next()) {
            cout << RED << "\nInvalid Staff ID. Please try again." << RESET << endl;
            system("pause");
            continue;
        }

        string currentName = db.res->getString("staff_name");
        string currentEmail = db.res->getString("staff_email");
        string currentRole = db.res->getString("staff_role");
        string currentPhone = db.res->getString("staff_phone_number");

        while (true) {
            system("cls");
            cout << CYAN << "+=================================================+" << RESET
<< endl;
            cout << CYAN << "|              UPDATE STAFF DETAILS              |" << RESET << endl;
            cout << CYAN << "+=================================================+" << RESET
<< endl;

            cout << YELLOW << "Staff ID: " << staffID << RESET << endl;
            cout << "1. Name: " << currentName << endl;
            cout << "2. Email: " << currentEmail << endl;
            cout << "3. Phone Number: " << currentPhone << endl;
            cout << "4. Role: " << currentRole << " (staff/manager)" << endl;
            cout << "5. Update All Details" << endl;
            cout << RED << "\n0. Cancel and Return\n" << RESET;

            cout << YELLOW << "\nWhat would you like to update? (1-5): " << RESET;
            string choice;
            cin >> choice;

            if (choice == "0") {
                return;
            }

            string newName = currentName, newEmail = currentEmail, newPhone = currentPhone,
newRole = currentRole;

            if (choice == "1") {
                cin.ignore();
                cout << YELLOW << "Enter new name (leave blank to keep current): " << RESET;
                getline(cin, newName);
            }
            else if (choice == "2") {
                cin.ignore();
                while (true) {
                    cout << YELLOW << "Enter new email (leave blank to keep current): " << RESET;
                    getline(cin, newEmail);

                    // Validate email format using regex
                    regex emailRegex(R"((\w+)(\.{1}\w+)*@(\w+)(\.\w{2,3})+)");
                    if (newEmail.empty() || regex_match(newEmail, emailRegex)) {
                        break; // Valid email
```

```
                }
                else {
                    cout << RED << "\n[Error] Invalid email format. Please use a valid email (e.g.,
user@example.com).\n" RESET;
                }
            }
        }
        else if (choice == "3") {
            cin.ignore();
            while (true) {
                cout << YELLOW << "Enter new phone number (leave blank to keep current): " << RESET;
                getline(cin, newPhone);

                // Validate phone number format using regex
                regex phoneRegex(R"(^\d{7,15}$)");
                if (newPhone.empty() || regex_match(newPhone, phoneRegex)) {
                    break; // Valid phone number
                }
                else {
                    cout << RED << "\n[Error] Invalid phone number. Please enter a number with 7 to 15
digits.\n" RESET;
                }
            }
        }
        else if (choice == "4") {
            cin.ignore();
            while (true) {
                cout << YELLOW << "Enter new role (1. Staff, 2. Manager, leave blank to keep current): "
<< RESET;
                string roleInput;
                getline(cin, roleInput);

                if (roleInput == "1") {
                    newRole = "staff";
                    break;
                }
                else if (roleInput == "2") {
                    newRole = "manager";
                    break;
                }
                else if (roleInput.empty()) {
                    break; // Keep current role
                }
                else {
                    cout << RED << "Invalid role choice. Please enter 1 for Staff or 2 for Manager.\n"
RESET;
                }
            }
        }
        else if (choice == "5") {
            cin.ignore();
            while (true) {
                cout << YELLOW << "Enter new name (leave blank to keep current): " << RESET;
                getline(cin, newName);

                cout << YELLOW << "Enter new email (leave blank to keep current): " << RESET;
                getline(cin, newEmail);
```

```
            // Validate email format
            regex emailRegex(R"((\w+)(\.{1}\w+)*@(\w+)(\.\w{2,3})+)");
            if (!newEmail.empty() && !regex_match(newEmail, emailRegex)) {
                cout << RED << "[Error] Invalid email format.\n" RESET;
                continue;
            }

            cout << YELLOW << "Enter new phone number (leave blank to keep current): " << RESET;
            getline(cin, newPhone);

            // Validate phone number format
            regex phoneRegex(R"(^\d{7,15}$)");
            if (!newPhone.empty() && !regex_match(newPhone, phoneRegex)) {
                cout << RED << "[Error] Invalid phone number.\n" RESET;
                continue;
            }

            cout << YELLOW << "Enter new role (1. Staff, 2. Manager, leave blank to keep current): "
<< RESET;
            string roleInput;
            getline(cin, roleInput);

            if (roleInput == "1") {
                newRole = "staff";
            }
            else if (roleInput == "2") {
                newRole = "manager";
            }

            break;
        }
    }
    else {
        cout << RED << "\nInvalid choice. Please try again." << RESET << endl;
        system("pause");
        continue;
    }

    // Update staff details in the database
    db.prepareStatement(
        "UPDATE staff SET "
        "staff_name = COALESCE(NULLIF(?, ''), staff_name), "
        "staff_email = COALESCE(NULLIF(?, ''), staff_email), "
        "staff_phone_number = COALESCE(NULLIF(?, ''), staff_phone_number), "
        "staff_role = COALESCE(NULLIF(?, ''), staff_role) "
        "WHERE staff_id = ?"
    );

    db.stmt->setString(1, newName);
    db.stmt->setString(2, newEmail);
    db.stmt->setString(3, newPhone);
    db.stmt->setString(4, newRole);
    db.stmt->setInt(5, staffID);

    db.QueryStatement();
```

```
            cout << GREEN << "\nStaff details updated successfully!" << RESET << endl;
            break;
          }
        }
        catch (sql::SQLException& e) {
          cerr << RED << "\nSQL Error: " << e.what() << RESET << endl;
        }
        catch (exception& e) {
          cerr << RED << "\nGeneral Error: " << e.what() << RESET << endl;
        }
    }
}

// Function to view staff list
void viewStaffList(DBConnection& db) {
    try {
        // Update the query to fetch phone number and password
        db.prepareStatement("SELECT staff_id, staff_name, staff_email, staff_role, staff_phone_number,
staff_password FROM staff ORDER BY staff_id");
        db.QueryResult();

        cout << YELLOW << "\nStaff List:\n" << RESET;
        cout << CYAN << "+---------+-----------------------+--------------------------+-----------+----------------------+----
----------------------+\n" << RESET;
        cout << GREEN << "| StaffID | Name                  | Email                    | Role      | Phone Number         |
Password              |\n" << RESET;
        cout << CYAN << "+---------+-----------------------+--------------------------+-----------+----------------------+----
----------------------+\n" << RESET;

        while (db.res->next()) {
            int staffID = db.res->getInt("staff_id");
            string name = db.res->getString("staff_name");
            string email = db.res->getString("staff_email");
            string role = db.res->getString("staff_role");
            string phoneNumber = db.res->getString("staff_phone_number");
            string password = db.res->getString("staff_password");

            cout << "| " << setw(7) << staffID << " | "
                 << setw(21) << name << " | "
                 << setw(24) << email << " | "
                 << setw(9) << role << " | "
                 << setw(20) << phoneNumber << " | "
                 << setw(24) << password << " |\n";
        }

        cout << CYAN << "+---------+-----------------------+--------------------------+-----------+----------------------+----
----------------------+\n" << RESET;
    }
    catch (sql::SQLException& e) {
        cerr << "Error fetching staff list: " << e.what() << endl;
    }
}

// Function to view customer list
void viewCustomerList(DBConnection& db) {
    try {
        db.prepareStatement(
```

```
        "SELECT customer_id, customer_name, customer_email, customer_phone_number, "
        "customer_birthday, customer_registration_date, is_member, membership_points "
        "FROM customer ORDER BY customer_id");
    db.QueryResult();

    cout << YELLOW << "\nCustomer List:\n" << RESET;
    cout << CYAN << "+------------+----------------------+----------------------------+-----------------+------------+--
--------------------+--------+--------+\n" << RESET;
    cout << GREEN << "| CustomerID | Name                 | Email                      | Phone Number    |
Birthday   | Registration       | Member | Points |\n" << RESET;
    cout << CYAN << "+------------+----------------------+----------------------------+-----------------+------------+--
--------------------+--------+--------+\n" << RESET;

    while (db.res->next()) {
        int customerID = db.res->getInt("customer_id");
        string name = db.res->getString("customer_name");
        string email = db.res->getString("customer_email");
        string phoneNumber = db.res->getString("customer_phone_number");
        string birthday = db.res->getString("customer_birthday");
        string registrationDateTime = db.res->getString("customer_registration_date");
        bool isMember = db.res->getInt("is_member") == 1;
        int points = db.res->getInt("membership_points");

        cout << "| " << setw(10) << customerID << " | "
            << setw(21) << name << " | "
            << setw(27) << email << " | "
            << setw(15) << phoneNumber << " | "
            << setw(10) << birthday << " | "
            << setw(19) << registrationDateTime << " | "
            << setw(6) << (isMember ? "Yes" : "No") << " | "
            << setw(6) << points << " |\n";
    }

    cout << CYAN << "+------------+----------------------+----------------------------+-----------------+------------+--
--------------------+--------+--------+\n" << RESET;
  }
  catch (sql::SQLException& e) {
    cerr << "Error fetching customer list: " << e.what() << endl;
  }
}

// Function to edit customer information
void updateCustomerInformation(DBConnection& db) {
  while (true) {
    try {
      // Step 1: Display the customer list
      system("cls");
      cout << CYAN << "+==================================================+" << RESET <<
endl;
      cout << CYAN << "|                UPDATE CUSTOMER                |" << RESET << endl;
      cout << CYAN << "+==================================================+" << RESET <<
endl;

      cout << YELLOW << "\nCurrent Customer List:" << RESET << endl;
      cout << CYAN <<  "+------------+----------------------+----------------------+--------------------+-----------
-------+" << RESET << endl;
```

```
        cout << GREEN << "| CustomerID | Name               | Email               | Phone Number       |
Membership Status |" << RESET << endl;
        cout << CYAN <<  "+------------+----------------------+-----------------------+--------------------+-----------
-------+" << RESET << endl;

        db.prepareStatement("SELECT customer_id, customer_name, customer_email,
customer_phone_number, is_member FROM customer ORDER BY customer_id");
        db.QueryResult();

        while (db.res->next()) {
           int customerID = db.res->getInt("customer_id");
           string name = db.res->getString("customer_name");
           string email = db.res->getString("customer_email");
           string phoneNumber = db.res->getString("customer_phone_number");
           int membershipStatus = db.res->getInt("is_member");

           cout << "| " << setw(10) << customerID << " | "
              << setw(21) << name << " | "
              << setw(23) << email << " | "
              << setw(19) << phoneNumber << " | "
              << setw(17) <<(membershipStatus ? "Yes" : "No") << " |" << endl;
        }
        cout << CYAN << "+------------+----------------------+-----------------------+--------------------+-----------
------+" << RESET << endl;

        cout << RED << "\n0. Cancel and Return to Customer Menu\n" << RESET;
        cout << YELLOW << "\nEnter the Customer ID to update: " << RESET;
        string input;
        cin >> input;

        if (input == "0") {
           return;
        }

        int customerID = stoi(input);

        // Validate customer ID
        db.prepareStatement("SELECT * FROM customer WHERE customer_id = ?");
        db.stmt->setInt(1, customerID);
        db.QueryResult();

        if (!db.res->next()) {
           cout << RED << "\nInvalid Customer ID. Please try again." << RESET << endl;
           system("pause");
           continue;
        }

        string currentName = db.res->getString("customer_name");
        string currentEmail = db.res->getString("customer_email");
        string currentPhone = db.res->getString("customer_phone_number");
        int currentMembershipStatus = db.res->getInt("is_member");

        while (true) {
           system("cls");
           cout << CYAN << "+================================================+" << RESET
<< endl;
           cout << CYAN << "|            UPDATE CUSTOMER DETAILS           |" << RESET << endl;
```

```
        cout << CYAN << "+================================================+" << RESET
<< endl;

        cout << YELLOW << "Customer ID: " << customerID << RESET << endl;
        cout << "1. Name: " << currentName << endl;
        cout << "2. Email: " << currentEmail << endl;
        cout << "3. Phone Number: " << currentPhone << endl;
        cout << "4. Membership Status: " << (currentMembershipStatus ? "Yes" : "No") << endl;
        cout << "5. Update All Details" << endl;
        cout << RED << "\n0. Cancel and Return\n" << RESET;

        cout << YELLOW << "\nWhat would you like to update? (1-5): " << RESET;
        string choice;
        cin >> choice;

        if (choice == "0") {
            return;
        }

        string newName = currentName, newEmail = currentEmail, newPhone = currentPhone;
        int newMembershipStatus = currentMembershipStatus;

        if (choice == "1") {
            cin.ignore();
            cout << YELLOW << "Enter new name (leave blank to keep current): " << RESET;
            getline(cin, newName);
        }
        else if (choice == "2") {
            cin.ignore();
            while (true) {
                cout << YELLOW << "Enter new email (leave blank to keep current): " << RESET;
                getline(cin, newEmail);

                // Validate email format using regex
                regex emailRegex(R"((\w+)(\.{1}\w+)*@(\w+)(\.\w{2,3})+)");
                if (newEmail.empty() || regex_match(newEmail, emailRegex)) {
                    break; // Valid email
                }
                else {
                    cout << RED << "\n[Error] Invalid email format. Please use a valid email (e.g.,
user@example.com).\n" RESET;
                }
            }
        }
        else if (choice == "3") {
            cin.ignore();
            while (true) {
                cout << YELLOW << "Enter new phone number (leave blank to keep current): " << RESET;
                getline(cin, newPhone);

                // Validate phone number format using regex
                regex phoneRegex(R"(^\d{7,15}$)");
                if (newPhone.empty() || regex_match(newPhone, phoneRegex)) {
                    break; // Valid phone number
                }
                else {
```

```
                cout << RED << "\n[Error] Invalid phone number. Please enter a number with 7 to 15
digits.\n" RESET;
            }
          }
        }
        else if (choice == "4") {
          cin.ignore();
          while (true) {
            cout << YELLOW << "Enter new membership status (1. Yes, 0. No, leave blank to keep
current): " << RESET;
            string membershipInput;
            getline(cin, membershipInput);

            if (membershipInput == "1") {
              newMembershipStatus = 1;
              break;
            }
            else if (membershipInput == "0") {
              newMembershipStatus = 0;
              break;
            }
            else if (membershipInput.empty()) {
              break; // Keep current membership status
            }
            else {
              cout << RED << "Invalid membership choice. Please enter 1 for Yes or 0 for No.\n"
RESET;
            }
          }
        }
        else if (choice == "5") {
          cin.ignore();
          while (true) {
            cout << YELLOW << "Enter new name (leave blank to keep current): " << RESET;
            getline(cin, newName);

            cout << YELLOW << "Enter new email (leave blank to keep current): " << RESET;
            getline(cin, newEmail);

            // Validate email format
            regex emailRegex(R"((\w+)(\.{1}\w+)*@(\w+)(\.\w{2,3})+)");
            if (!newEmail.empty() && !regex_match(newEmail, emailRegex)) {
              cout << RED << "[Error] Invalid email format.\n" RESET;
              continue;
            }

            cout << YELLOW << "Enter new phone number (leave blank to keep current): " << RESET;
            getline(cin, newPhone);

            // Validate phone number format
            regex phoneRegex(R"(^\d{7,15}$)");
            if (!newPhone.empty() && !regex_match(newPhone, phoneRegex)) {
              cout << RED << "[Error] Invalid phone number.\n" RESET;
              continue;
            }
```

```
                cout << YELLOW << "Enter new membership status (1. Yes, 0. No, leave blank to keep
current): " << RESET;
                string membershipInput;
                getline(cin, membershipInput);

                if (membershipInput == "1") {
                    newMembershipStatus = 1;
                }
                else if (membershipInput == "0") {
                    newMembershipStatus = 0;
                }

                break;
            }
        }
        else {
            cout << RED << "\nInvalid choice. Please try again." << RESET << endl;
            system("pause");
            continue;
        }

        // Update customer details in the database
        db.prepareStatement(
            "UPDATE customer SET "
            "customer_name = COALESCE(NULLIF(?, ''), customer_name), "
            "customer_email = COALESCE(NULLIF(?, ''), customer_email), "
            "customer_phone_number = COALESCE(NULLIF(?, ''), customer_phone_number), "
            "is_member = COALESCE(?, is_member) "
            "WHERE customer_id = ?"
        );

        db.stmt->setString(1, newName);
        db.stmt->setString(2, newEmail);
        db.stmt->setString(3, newPhone);
        db.stmt->setInt(4, newMembershipStatus);
        db.stmt->setInt(5, customerID);

        db.QueryStatement();

        cout << GREEN << "\nCustomer details updated successfully!" << RESET << endl;
        break;
      }
    }
    catch (sql::SQLException& e) {
      cerr << RED << "\nSQL Error: " << e.what() << RESET << endl;
    }
    catch (exception& e) {
      cerr << RED << "\nGeneral Error: " << e.what() << RESET << endl;
    }
  }
}

// Function for customer to rate food
bool rateFood(DBConnection& db, int customerID, int menuItemID, int rating) {
  try {
    if (rating < 1 || rating > 5) {
      cout << RED << "Invalid rating. Please provide a rating between 1 and 5.\n" << RESET;
```

```
    return false;
}

// First check if the customer has ordered this item
db.prepareStatement(
    "SELECT DISTINCT od.menu_item_id "
    "FROM order_details od "
    "JOIN `order` o ON od.order_id = o.order_id "
    "WHERE o.customer_id = ? AND od.menu_item_id = ?"
);
db.stmt->setInt(1, customerID);
db.stmt->setInt(2, menuItemID);
db.QueryResult();

if (!db.res->next()) {
    cout << RED << "You can only rate items you have ordered.\n" << RESET;
    return false;
}

// Check if the customer has already rated this item
db.prepareStatement(
    "SELECT rating_id, rating FROM food_ratings "
    "WHERE customer_id = ? AND menu_item_id = ?"
);
db.stmt->setInt(1, customerID);
db.stmt->setInt(2, menuItemID);
db.QueryResult();

if (db.res->next()) {
    // Update existing rating
    int oldRating = db.res->getInt("rating");

    db.prepareStatement(
        "UPDATE food_ratings SET rating = ?, updated_at = CURRENT_TIMESTAMP "
        "WHERE customer_id = ? AND menu_item_id = ?"
    );
    db.stmt->setInt(1, rating);
    db.stmt->setInt(2, customerID);
    db.stmt->setInt(3, menuItemID);
    db.QueryStatement();

    // Update menu_item table
    db.prepareStatement(
        "UPDATE menu_item "
        "SET total_rating = total_rating - ? + ? "
        "WHERE menu_item_id = ?"
    );
    db.stmt->setInt(1, oldRating);
    db.stmt->setInt(2, rating);
    db.stmt->setInt(3, menuItemID);
    db.QueryStatement();
}
else {
    // Insert new rating
    db.prepareStatement(
        "INSERT INTO food_ratings (customer_id, menu_item_id, rating) "
        "VALUES (?, ?, ?)"
```

```
            );
            db.stmt->setInt(1, customerID);
            db.stmt->setInt(2, menuItemID);
            db.stmt->setInt(3, rating);
            db.QueryStatement();

            // Update menu_item table
            db.prepareStatement(
               "UPDATE menu_item "
               "SET total_rating = total_rating + ?, "
               "rating_count = rating_count + 1 "
               "WHERE menu_item_id = ?"
            );
            db.stmt->setInt(1, rating);
            db.stmt->setInt(2, menuItemID);
            db.QueryStatement();
         }

         return true;
      }
      catch (sql::SQLException& e) {
         cerr << RED << "Error while rating the food: " << e.what() << RESET << endl;
         return false;
      }
   }

   //displayMessage
   void displayMessageBox(const string& message, const string& color) {
      const int boxWidth = 47;
      int padding = (boxWidth - message.length()) / 2;
      int paddingRight = boxWidth - message.length() - padding;

      cout << color << "+-----------------------------------------------+\n";
      cout << "| " << string(padding, ' ') << message << string(paddingRight, ' ') << " |\n";
      cout << "+-----------------------------------------------+" << RESET << "\n";
   }

   // loading animation
   void loadingAnimation() {
      const char* spinner = "|/-\\";
      cout << PROMPT << "Processing " << RESET;
      for (int i = 0; i < 10; ++i) {
         cout << spinner[i % 4] << flush;
         this_thread::sleep_for(chrono::milliseconds(150));
         cout << "\b";
      }
      cout << " Done!\n";
   }
```