Starbucks Application Using JavaFX

Supriya kankati

016657771

1. Introduction:

The **Starbucks Application** is a **JavaFX**-based **Coffee Shop Management System**, focused on facilitating the creation and maintenance of a menu, offer customers the flexibility to place orders, customize their beverages, and process payments, based on the operational model of Starbucks.

Developed using Java, taking advantage of its Object-Oriented Programming (OOP) capabilities, it is ideal for structuring a complex coffee shop management system. I decided from the beginning to include JavaFX because I wanted to create an interactive, user-friendly interface that simulates a digital coffee shop environment.

To integrate **JavaFX** with minimal effort, I added it as a project dependency and chose to use **Maven** as a **build tool**. This eliminated the need for manually configuring JavaFX jar libraries and the setup required for running the application, simplifying this part of the process and allowing me to focus on UI development.

2. Objective:

The primary objective of my project is to develop a Coffee Shop Management System, based on Object-Oriented Programming (OOP) principles. My aim is to apply these principles to deliver the functionalities needed for an effective coffee shop management, including the creation, editing, and maintenance of a menu, as well as managing customer interactions and orders.

In terms of technical execution, my focus has been on applying OOP principles to make the codebase clean, readable, reusable, and efficient. This approach involves the strategic use of encapsulation, inheritance, association, and composition.

The application will include the following features:

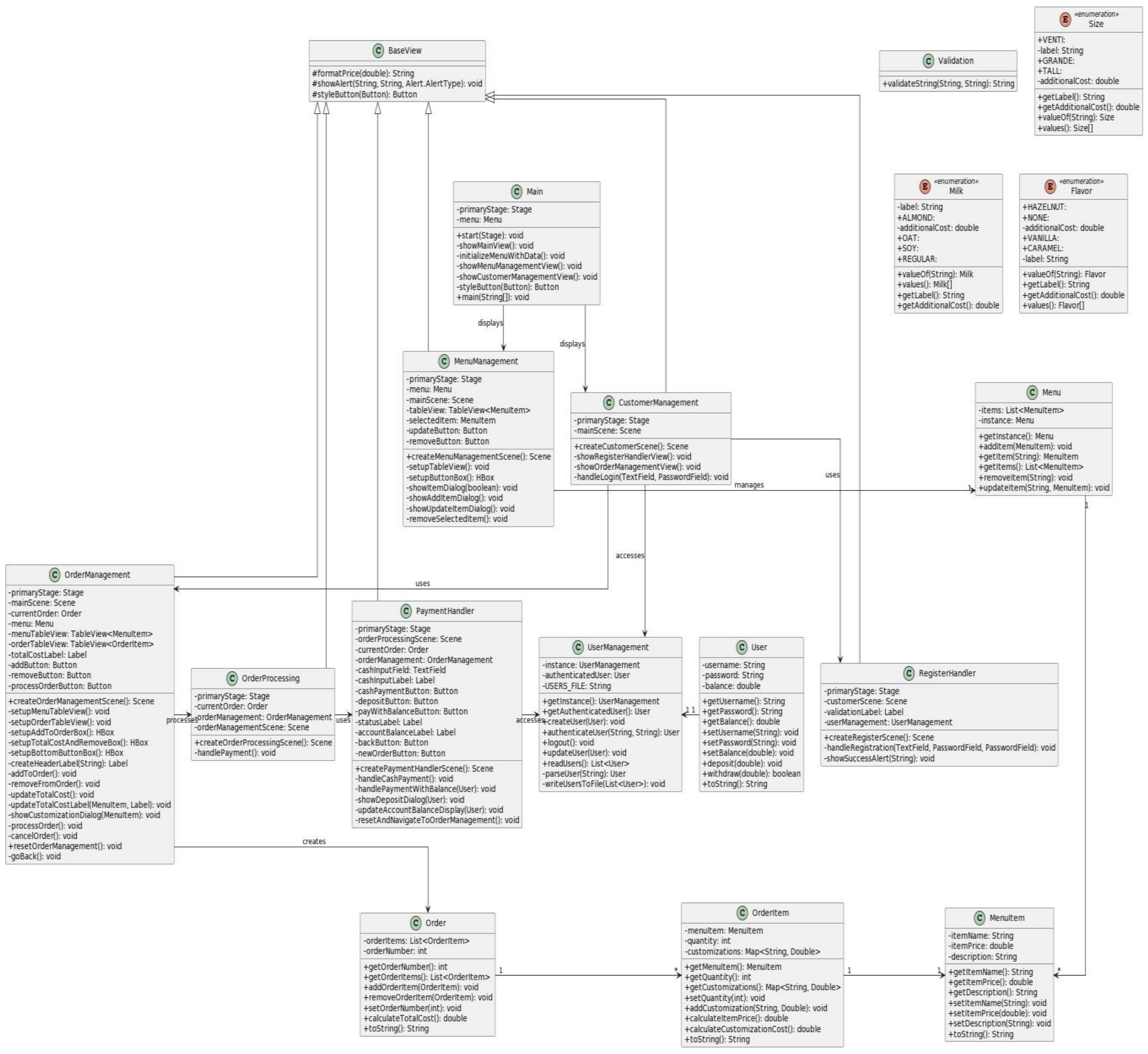
- **Menu Management**: The system will enable staff to add, edit, or delete items from the menu, ensuring flexibility and up-to-date offerings.
- **Order Management**: This will allow customers to view the menu, select items, and create customized orders.

- **Order Processing**: Customers will be able to view the total cost and a summary of their orders before proceeding to payment.
- **Order Payment**: The application will provide options for customers to pay using cash, with the system calculating and managing change effectively.

As part of the bonus features, I plan to incorporate:

- **User Authentication**: This will enable customers to place orders as either guests or registered users, with options to create an account or log in.
- **Payment with Account Balance**: This feature will offer an alternative to cash payments, allowing logged-in users to view their account balance, top it up, and use it for payments.

3. Project High-Level Design (UML Diagram):


BaseView
#formatPrice(double): String
#showAlert(String, String, Alert.AlertType): void
#styleButton(Button): Button
Validation
+validateString(String, String): String
«enumeration» Size
+VENTI:
-label: String
+GRANDE:
+TALL:
-additionalCost: double
+getLabel(): String
+getAdditionalCost(): double
+valueOf(String): Size
+values(): Size[]
Main
-primaryStage: Stage
-menu: Menu
+start(Stage): void
-showMainView(): void
-initializeMenuWithData(): void
-showMenuManagementView(): void
-showCustomerManagementView(): void
-styleButton(Button): Button
+main(String[]): void
«enumeration» Milk
-label: String
+ALMOND:
-additionalCost: double
+OAT:
+SOY:
+REGULAR:
+valueOf(String): Milk
+values(): Milk[]
+getLabel(): String
+getAdditionalCost(): double
«enumeration» Flavor
+HAZELNUT:
+NONE:
-additionalCost: double
+VANILLA:
+CARAMEL:
-label: String
+valueOf(String): Flavor
+getLabel(): String
+getAdditionalCost(): double
+values(): Flavor[]
displays
displays
MenuManagement
-primaryStage: Stage
-menu: Menu
-mainScene: Scene
-tableView: TableView<MenuItem>
-selectedItem: MenuItem
-updateButton: Button
-removeButton: Button
+createMenuManagementScene(): Scene
-setupTableView(): void
-setupButtonBox(): HBox
-showItemDialog(boolean): void
-showAddItemDialog(): void
-showUpdateItemDialog(): void
-removeSelectedItem(): void
CustomerManagement
-primaryStage: Stage
-mainScene: Scene
+createCustomerScene(): Scene
-showRegisterHandlerView(): void
-showOrderManagementView(): void
-handleLogin(TextField, PasswordField): void
uses
manages
1
Menu
-items: List<MenuItem>
-instance: Menu
+getInstance(): Menu
+addItem(MenuItem): void
+getItem(String): MenuItem
+getItems(): List<MenuItem>
+removeItem(String): void
+updateItem(String, MenuItem): void
1
accesses
uses
OrderManagement
-primaryStage: Stage
-mainScene: Scene
-currentOrder: Order
-menu: Menu
-menuTableView: TableView<MenuItem>
-orderTableView: TableView<OrderItem>
-totalCostLabel: Label
-addButton: Button
-removeButton: Button
-processOrderButton: Button
+createOrderManagementScene(): Scene
-setupMenuTableView(): void
-setupOrderTableView(): void
-setupAddToOrderBox(): HBox
-setupTotalCostAndRemoveBox(): HBox
-setupBottomButtonBox(): HBox
-createHeaderLabel(String): Label
-addToOrder(): void
-removeFromOrder(): void
-updateTotalCost(): void
-updateTotalCostLabel(MenuItem, Label): void
-showCustomizationDialog(MenuItem): void
-processOrder(): void
-cancelOrder(): void
+resetOrderManagement(): void
-goBack(): void
processes
OrderProcessing
-primaryStage: Stage
-currentOrder: Order
-orderManagement: OrderManagement
-orderManagementScene: Scene
+createOrderProcessingScene(): Scene
-handlePayment(): void
uses
PaymentHandler
-primaryStage: Stage
-orderProcessingScene: Scene
-currentOrder: Order
-orderManagement: OrderManagement
-cashInputField: TextField
-cashInputLabel: Label
-cashPaymentButton: Button
-depositButton: Button
-payWithBalanceButton: Button
-statusLabel: Label
-accountBalanceLabel: Label
-backButton: Button
-newOrderButton: Button
+createPaymentHandlerScene(): Scene
-handleCashPayment(): void
-handlePaymentWithBalance(User): void
-showDepositDialog(User): void
-updateAccountBalanceDisplay(User): void
-resetAndNavigateToOrderManagement(): void
accesses
UserManagement
-instance: UserManagement
-authenticatedUser: User
-USERS_FILE: String
+getInstance(): UserManagement
+getAuthenticatedUser(): User
+createUser(User): void
+authenticateUser(String, String): User
+logout(): void
+updateUser(User): void
+readUsers(): List<User>
-parseUser(String): User
-writeUsersToFile(List<User>): void
1 1
User
-username: String
-password: String
-balance: double
+getUsername(): String
+getPassword(): String
+getBalance(): double
+setUsername(String): void
+setPassword(String): void
+setBalance(double): void
+deposit(double): void
+withdraw(double): boolean
+toString(): String
RegisterHandler
-primaryStage: Stage
-customerScene: Scene
-validationLabel: Label
-userManagement: UserManagement
+createRegisterScene(): Scene
-handleRegistration(TextField, PasswordField, PasswordField): void
-showSuccessAlert(String): void
creates
Order
-orderItems: List<OrderItem>
-orderNumber: int
+getOrderNumber(): int
+getOrderItems(): List<OrderItem>
+addOrderItem(OrderItem): void
+removeOrderItem(OrderItem): void
+setOrderNumber(int): void
+calculateTotalCost(): double
+toString(): String
1
*
OrderItem
-menuItem: MenuItem
-quantity: int
-customizations: Map<String, Double>
+getMenuItem(): MenuItem
+getQuantity(): int
+getCustomizations(): Map<String, Double>
+setQuantity(int): void
+addCustomization(String, Double): void
+calculateItemPrice(): double
+calculateCustomizationCost(): double
+toString(): String
1
1
MenuItem
-itemName: String
-itemPrice: double
-description: String
+getItemName(): String
+getItemPrice(): double
+getDescription(): String
+setItemName(String): void
+setItemPrice(double): void
+setDescription(String): void
+toString(): String
*

The Starbucks Application, realized through JavaFX, adopts an architectural design closely aligned with the Model-View-ViewModel (MVVM) pattern. However, in my implementation, the View and ViewModel functionalities are combined into single classes due to the absence of SceneBuilder or FXML files for separate view definitions. Instead, the UI components and their associated logic are programmatically defined within Java classes.

Starting from the previously defined UML Class diagram, during the midterm phase, and throughout the implementation process, it required numerous modifications and additions to concretize all the requested features, including the UI and bonus features.

Since the purpose of this project is to apply **OOP** principles, I organized and structured the code into distinct **packages** and **layers**, thus improving **maintainability**, **scalability**, and project **clarity**. This involved introducing **encapsulation** and **abstraction** by dividing the application into multiple classes, with their implementation hidden, exposing only what is necessary. Additionally, the Single Responsibility Principle (**SRP**) was applied, where each class in each package is responsible for a specific part of the application, making it more robust and easier to maintain.

The **architectural structure** of the application is as follows:

- **Model** Layer: This layer includes the initial classes **Menu**, **MenuItem**, **Order**, **OrderItem**, and later the **User** class for the bonus authentication feature. These classes serve as the foundation of the application, representing essential data structures and business logic.

  Notably, the **Menu** class, implemented as a **singleton**, manages a collection of **MenuItem** instances, ensuring consistent and centralized menu data throughout the application.

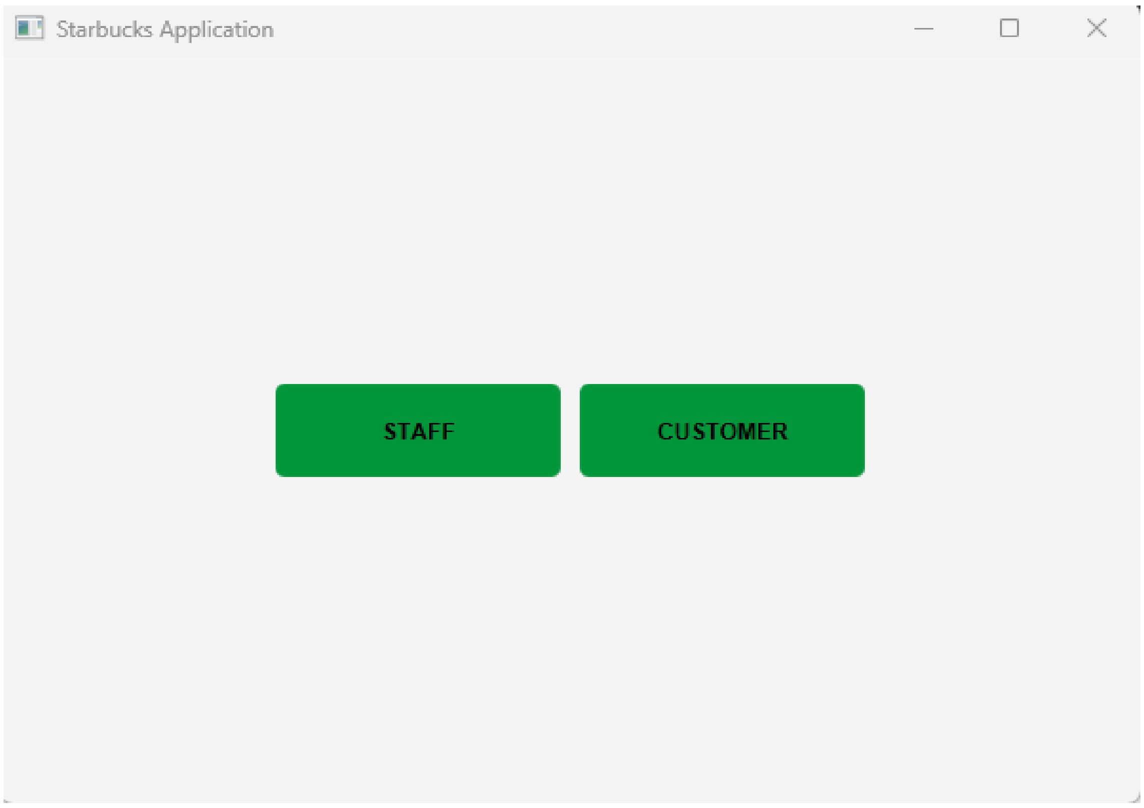
- **View-ViewModel** Layer: This layer contains the view classes (**CustomerManagement**, **MenuManagement**, **OrderManagement**, **OrderProcessing**, **PaymentHandler**, **RegisterHandler**), all extending from **BaseView**. They are the core components of the user interface, encapsulating both the UI elements (View) and the logic to handle user interactions (ViewModel). This approach merges responsibilities typically separated in a standard MVVM approach.
- **Utilities**: Utility classes such as **UserManagement** and **Validation** provide support functions, including user authentication and input validation, showcasing the application of OOP concepts like abstraction and encapsulation.
- **Enumerations**: The **enums** package defines constant values (**Flavor**, **Milk**, **Size**) used for menu item customizations, ensuring standardization and ease of use across the application.

4. Results:

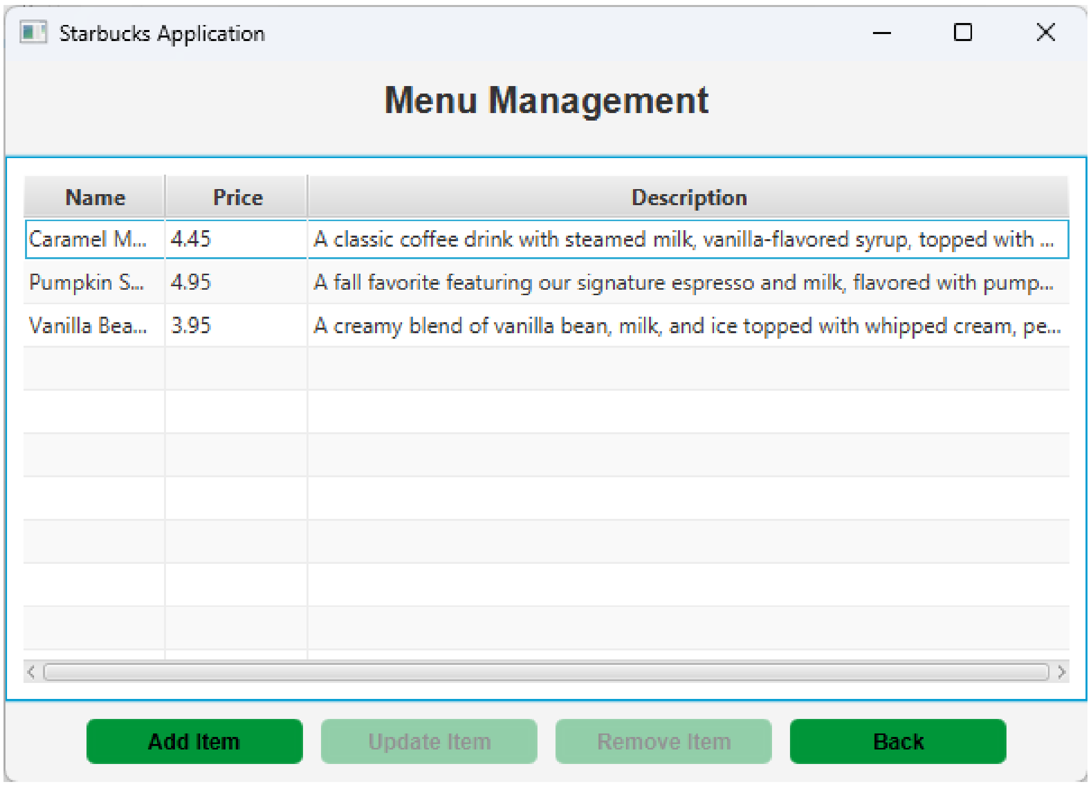
- The Starbucks Application begins with the **Main** class, the entry point that presents a main view where users choose between **Staff** and **Customer** roles. Selecting Staff directs to **MenuManagement** where staff can modify **MenuItems** in the **Menu**, a singleton shared across the application. Here, staff can **add**, **update**, or **remove** items, and **navigate** back to the main view.
- For customers, the **CustomerManagement** view offers the option to continue as a **Guest** or to **log in**/**register**. Proceeding as either leads to **OrderManagement**, where customers **view menu items** and **create** their **orders**, including **customization** through dialogs for **quantity** and additions like size, flavors or milk types.
- After finalizing the order, **OrderProcessing** displays a **detailed summary** and **total cost**, leading to **PaymentHandler** for transaction completion. This view varies based on user status. Guests only see the **total cost** and **cash payment input**, with the application handling **payment confirmation** and **change calculation**. Logged-in **users** see their **username**, **account balance**, and have the option to **pay with their account balance** or **deposit funds**, facilitated by **UserManagement**, which handles user **authentication** and **account** operations.
- So, as mentioned above, we can see that **all the core features have been successfully implemented**, including **bonus features** like **User Authentication** (**Register** / **Login**, or continue as a **Guest**) and the dynamic view of Payment Handling based on the user's status, which adds an extra feature, If the user is logged in, they can pay with their account balance.
- I have **not used a real database** for data persistence, such as storing user information like username-password or account balance. However, to achieve this feature, I had to find an alternative, which was simulating a basic database table in a **text file** named 'users.txt.' The application automatically creates this file if it doesn't exist when the first user registers. The responsibility for this lies with the **UserManagement** class, which handles **writing** and **reading** data from the file, mapping them as User objects, and updating a user's information when their account balance changes.

5. Screenshots of the Application with All the Views:

Main view :

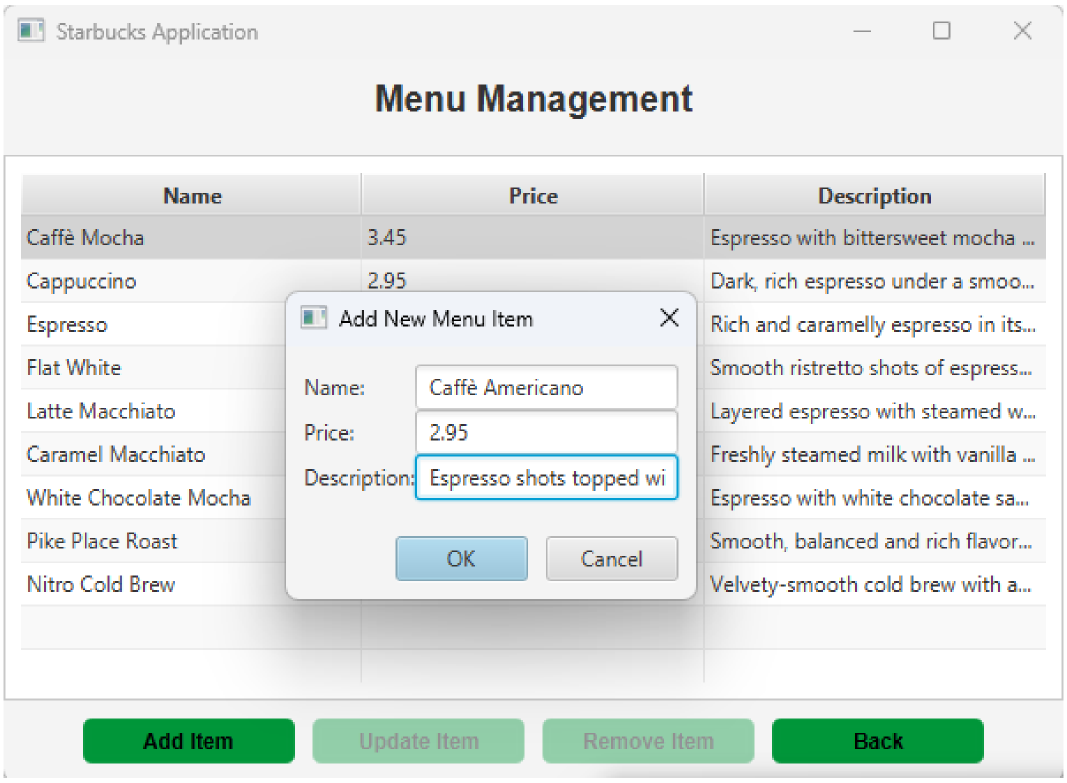


MenuManagement view (STAFF) :



Name	Price	Description
Caramel M...	4.45	A classic coffee drink with steamed milk, vanilla-flavored syrup, topped with ...
Pumpkin S...	4.95	A fall favorite featuring our signature espresso and milk, flavored with pump...
Vanilla Bea...	3.95	A creamy blend of vanilla bean, milk, and ice topped with whipped cream, pe...

Add MenuItem :


Starbucks Application

Menu Management

Name	Price	Description
Caffè Mocha	3.45	Espresso with bittersweet mocha ...
Cappuccino	2.95	Dark, rich espresso under a smoo...
Espresso		Rich and caramelly espresso in its...
Flat White		Smooth ristretto shots of espress...
Latte Macchiato		Layered espresso with steamed w...
Caramel Macchiato		Freshly steamed milk with vanilla ...
White Chocolate Mocha		Espresso with white chocolate sa...
Pike Place Roast		Smooth, balanced and rich flavor...
Nitro Cold Brew		Velvety-smooth cold brew with a...

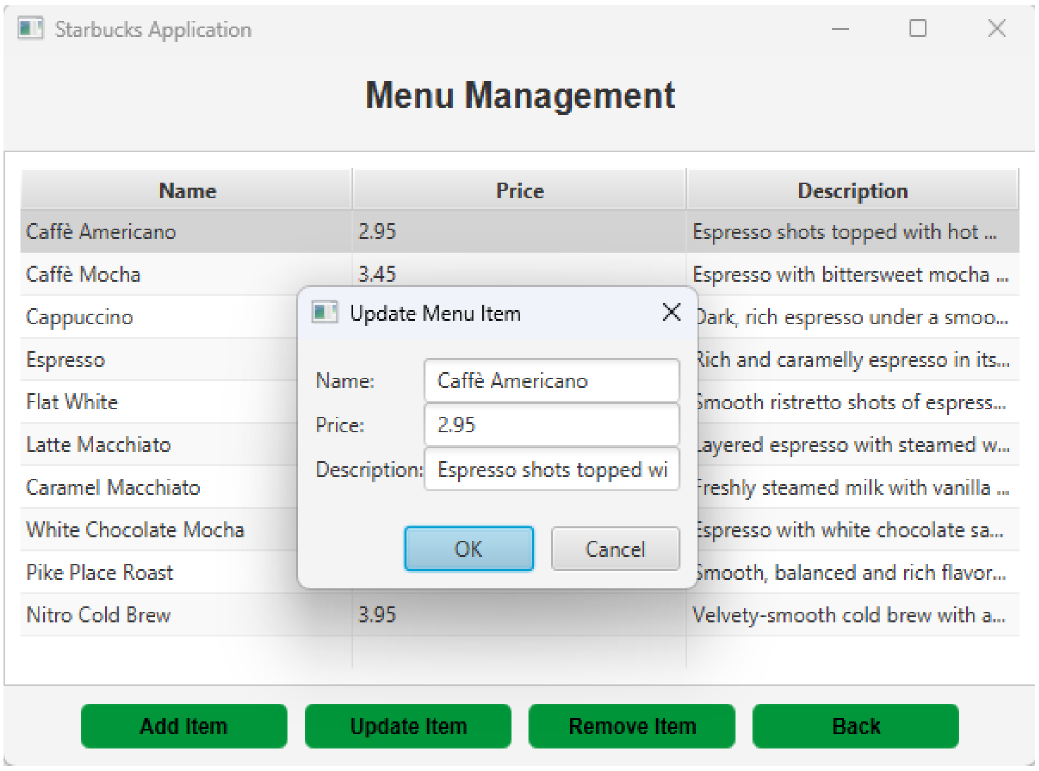
Add New Menu Item

Name: Caffè Americano

Price: 2.95

Description: Espresso shots topped wi

Update MenuItem :


Starbucks Application

Menu Management

Name	Price	Description
Caffè Americano	2.95	Espresso shots topped with hot ...
Caffè Mocha	3.45	Espresso with bittersweet mocha ...
Cappuccino		Dark, rich espresso under a smoo...
Espresso		Rich and caramelly espresso in its...
Flat White		Smooth ristretto shots of espress...
Latte Macchiato		Layered espresso with steamed w...
Caramel Macchiato		Freshly steamed milk with vanilla ...
White Chocolate Mocha		Espresso with white chocolate sa...
Pike Place Roast		Smooth, balanced and rich flavor...
Nitro Cold Brew	3.95	Velvety-smooth cold brew with a...

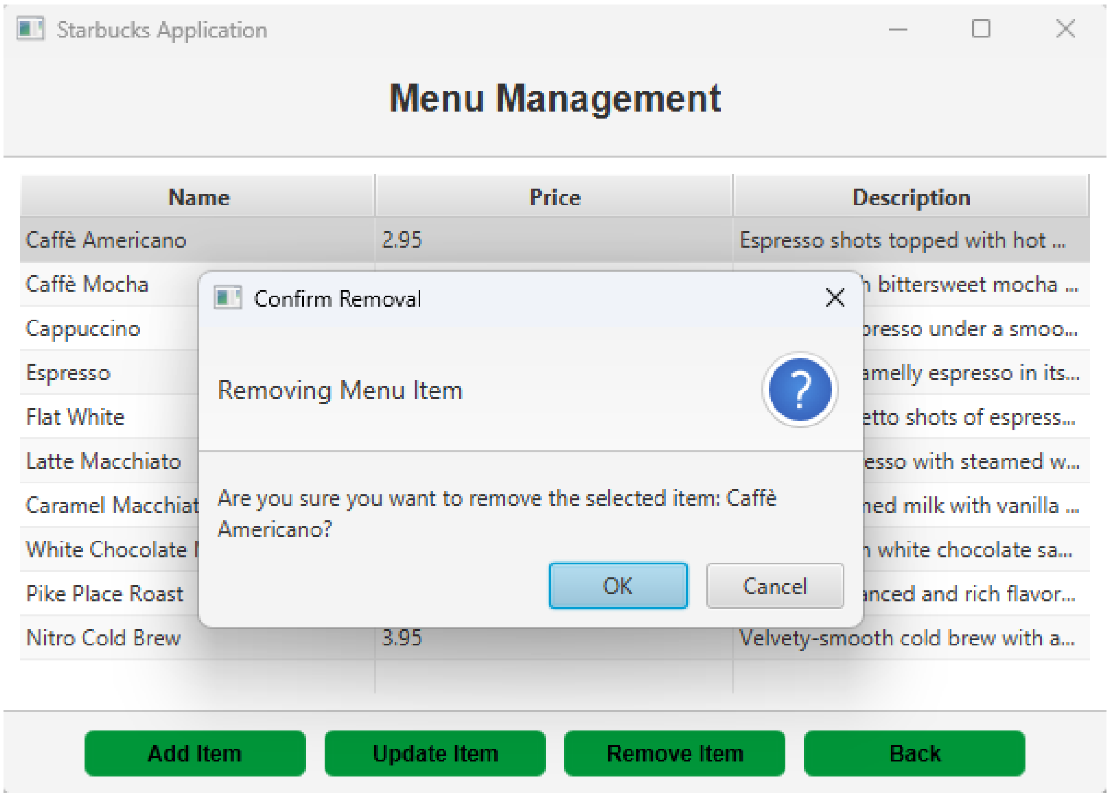
Update Menu Item

Name: Caffè Americano

Price: 2.95

Description: Espresso shots topped wi

Remove MenuItem :


Starbucks Application

Menu Management

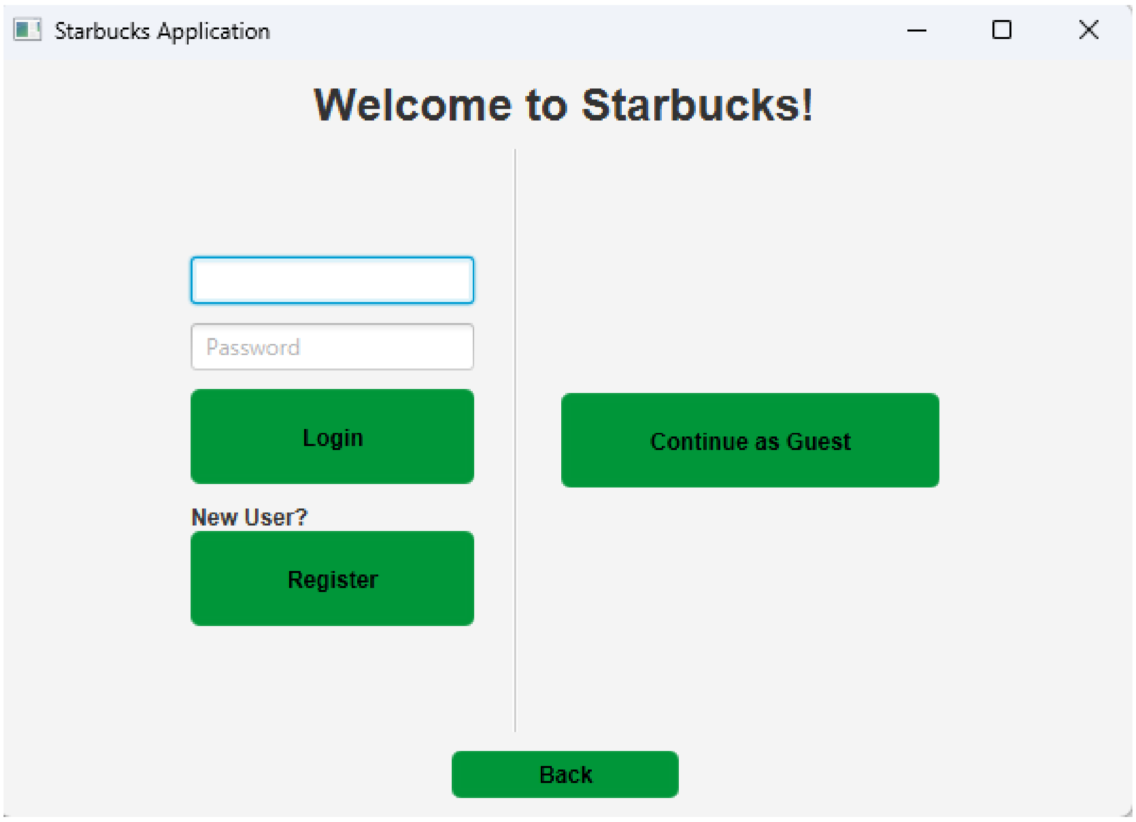
Name	Price	Description
Caffè Americano	2.95	Espresso shots topped with hot ...
Caffè Mocha		n bittersweet mocha ...
Cappuccino		presso under a smoo...
Espresso		amelly espresso in its...
Flat White		etto shots of espress...
Latte Macchiato		esso with steamed w...
Caramel Macchiat		ned milk with vanilla ...
White Chocolate I		n white chocolate sa...
Pike Place Roast		anced and rich flavor...
Nitro Cold Brew	3.95	Velvety-smooth cold brew with a...

Confirm Removal

Removing Menu Item

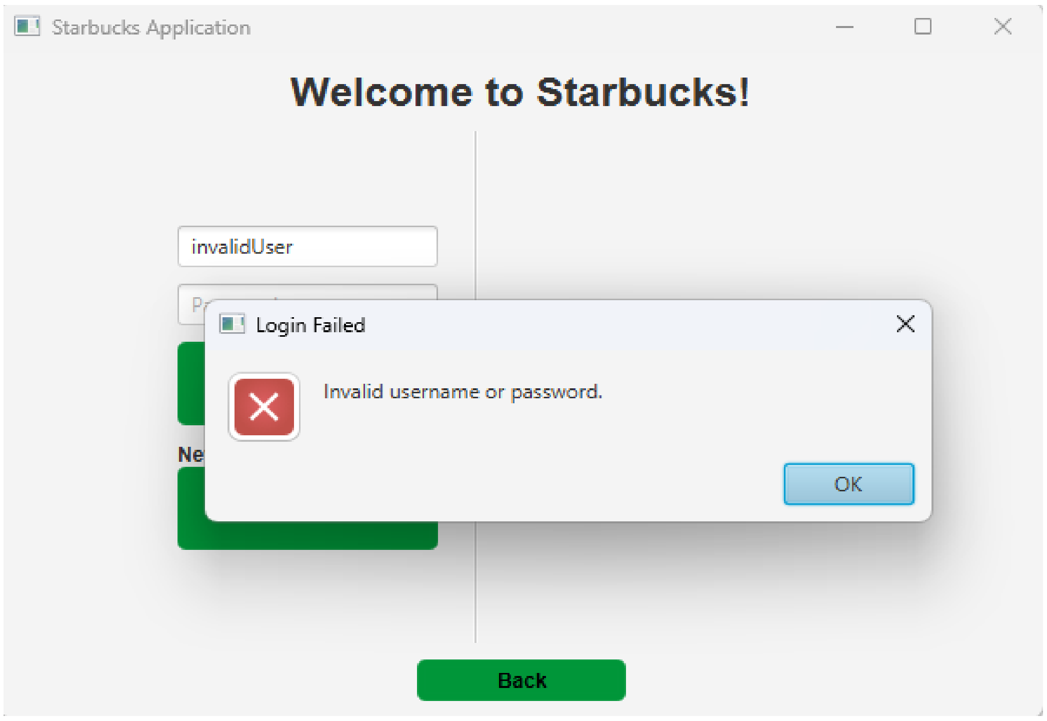
Are you sure you want to remove the selected item: Caffè Americano?

CustomerManagement view :

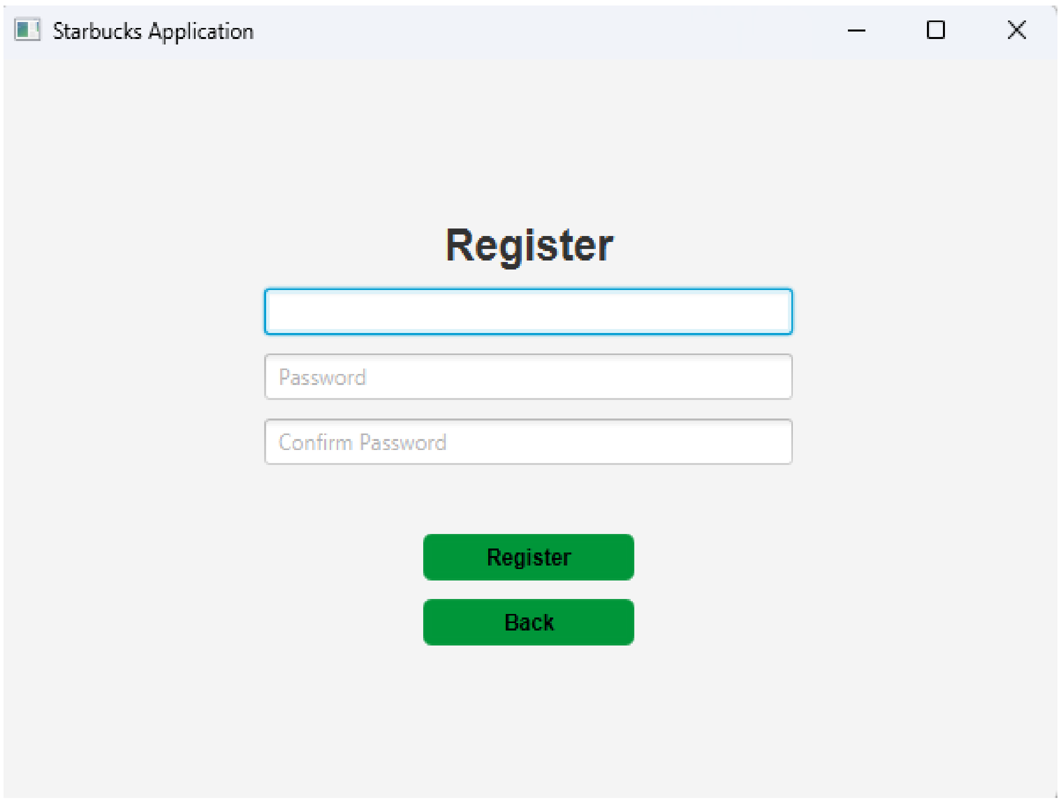

Starbucks Application

Welcome to Starbucks!

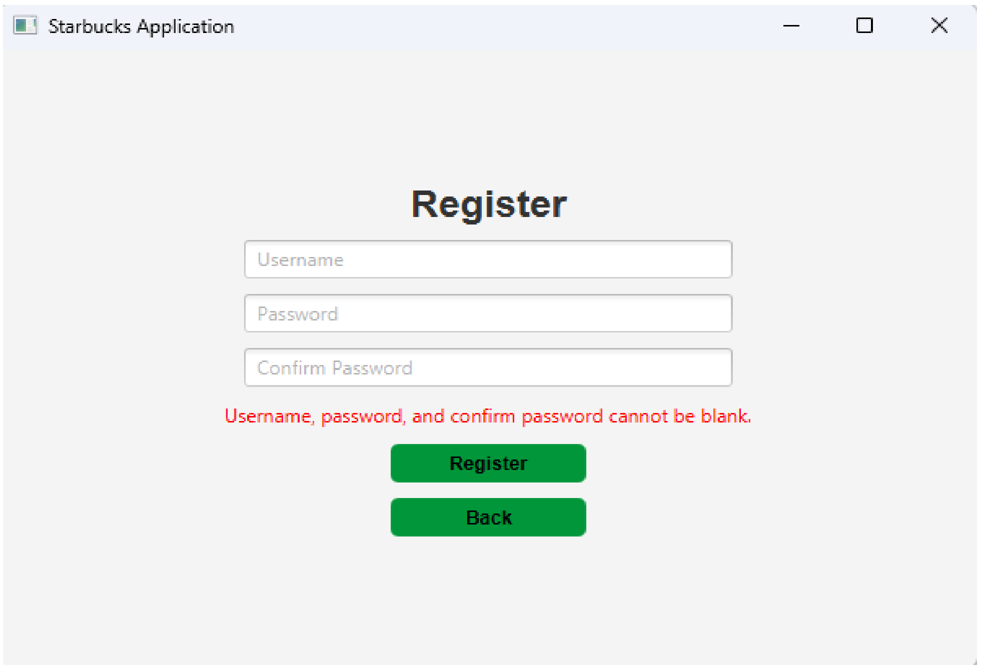
Login Validation:



RegisterHandler view :

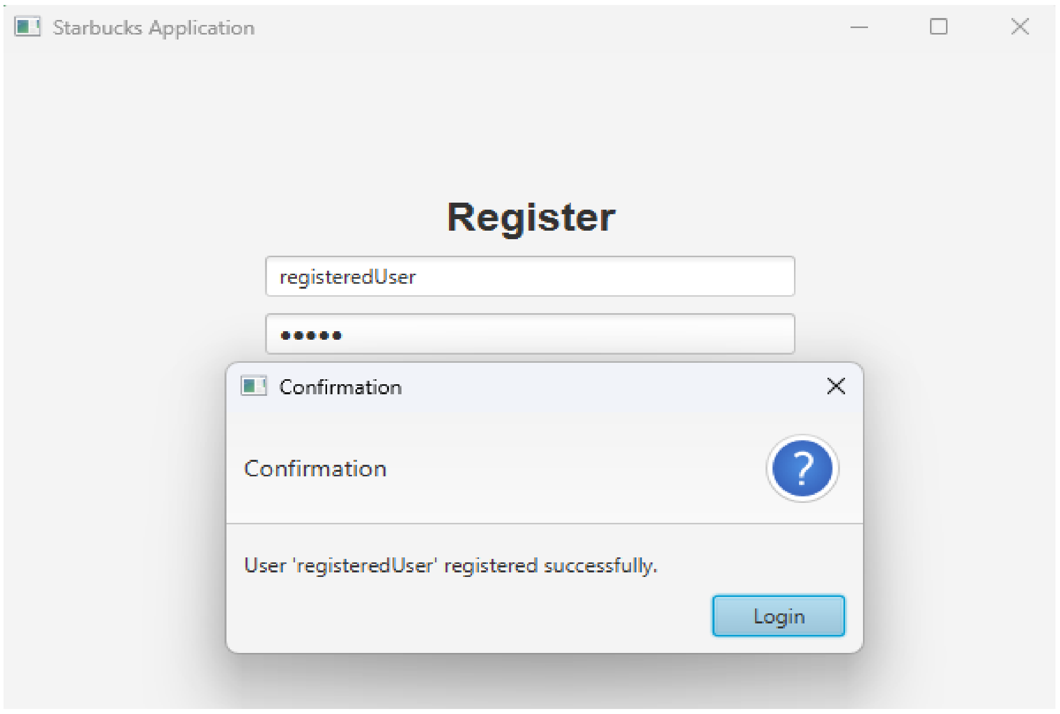


Register validation :



Username, password, and confirm password cannot be blank.

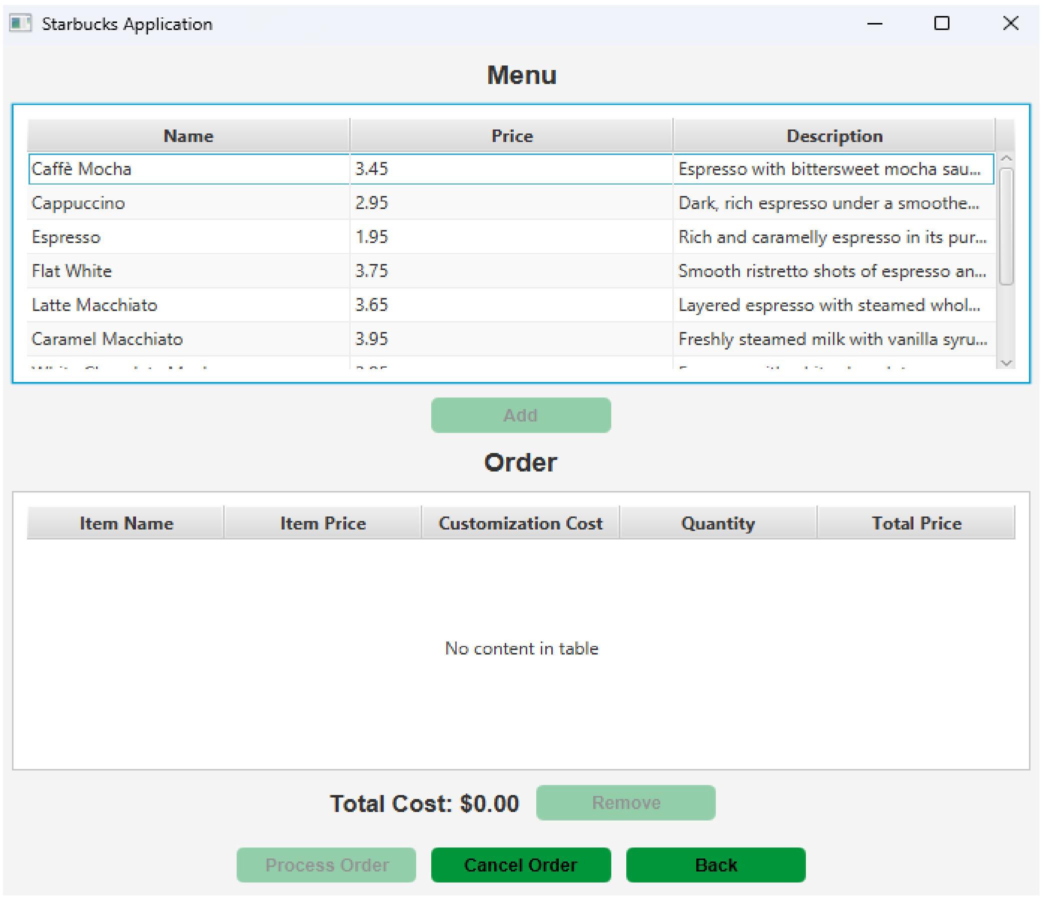
Register confirmation :



Confirmation

User 'registeredUser' registered successfully.

OrderManagement view :



Menu

Name	Price	Description
Caffè Mocha	3.45	Espresso with bittersweet mocha sau...
Cappuccino	2.95	Dark, rich espresso under a smoothe...
Espresso	1.95	Rich and caramelly espresso in its pur...
Flat White	3.75	Smooth ristretto shots of espresso an...
Latte Macchiato	3.65	Layered espresso with steamed whol...
Caramel Macchiato	3.95	Freshly steamed milk with vanilla syru...

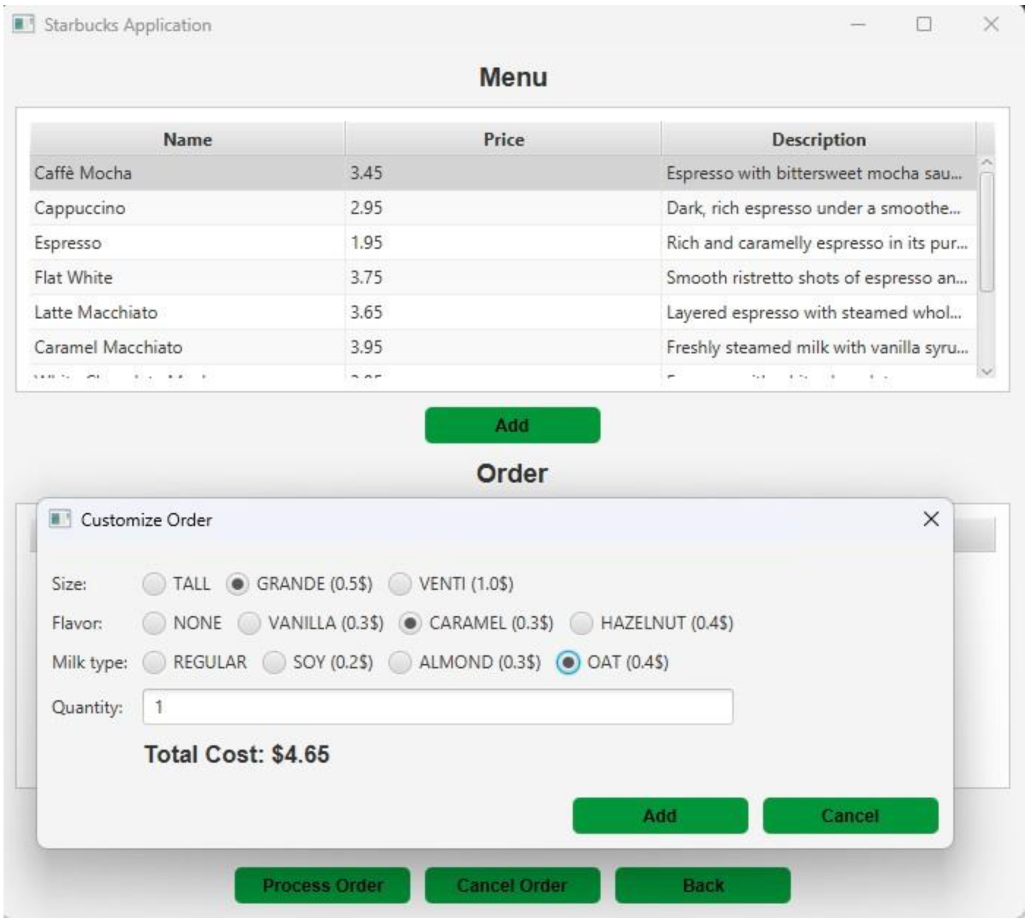
Order

Item Name	Item Price	Customization Cost	Quantity	Total Price

No content in table

Total Cost: $0.00

Order Add Item and Customize :



Menu

Name	Price	Description
Caffè Mocha	3.45	Espresso with bittersweet mocha sau...
Cappuccino	2.95	Dark, rich espresso under a smoothe...
Espresso	1.95	Rich and caramelly espresso in its pur...
Flat White	3.75	Smooth ristretto shots of espresso an...
Latte Macchiato	3.65	Layered espresso with steamed whol...
Caramel Macchiato	3.95	Freshly steamed milk with vanilla syru...

Order

Customize Order

Size: TALL [✓ GRANDE (0.5$)] VENTI (1.0$)

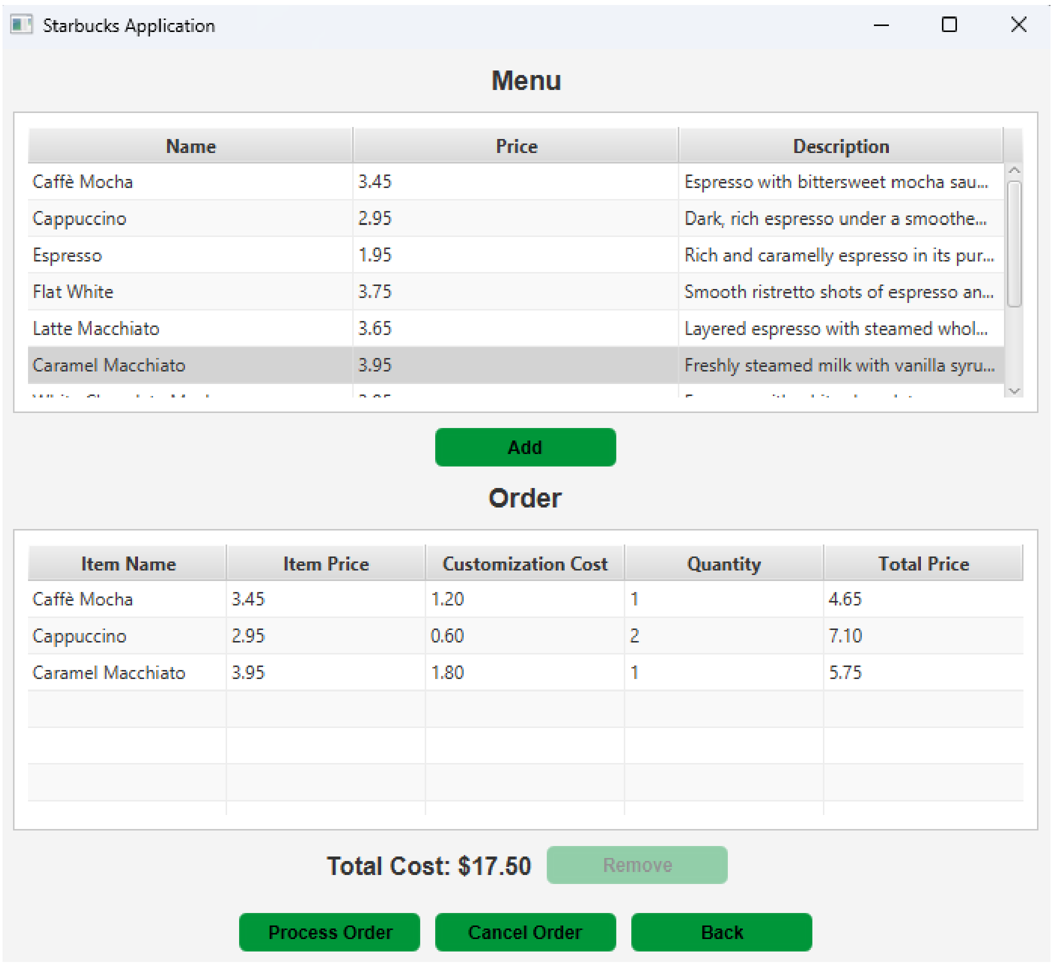
Flavor: NONE VANILLA (0.3$) [✓ CARAMEL (0.3$)] HAZELNUT (0.4$)

Milk type: REGULAR SOY (0.2$) ALMOND (0.3$) [✓ OAT (0.4$)]

Quantity: 1

Total Cost: $4.65

OrderManagement items view :



Menu

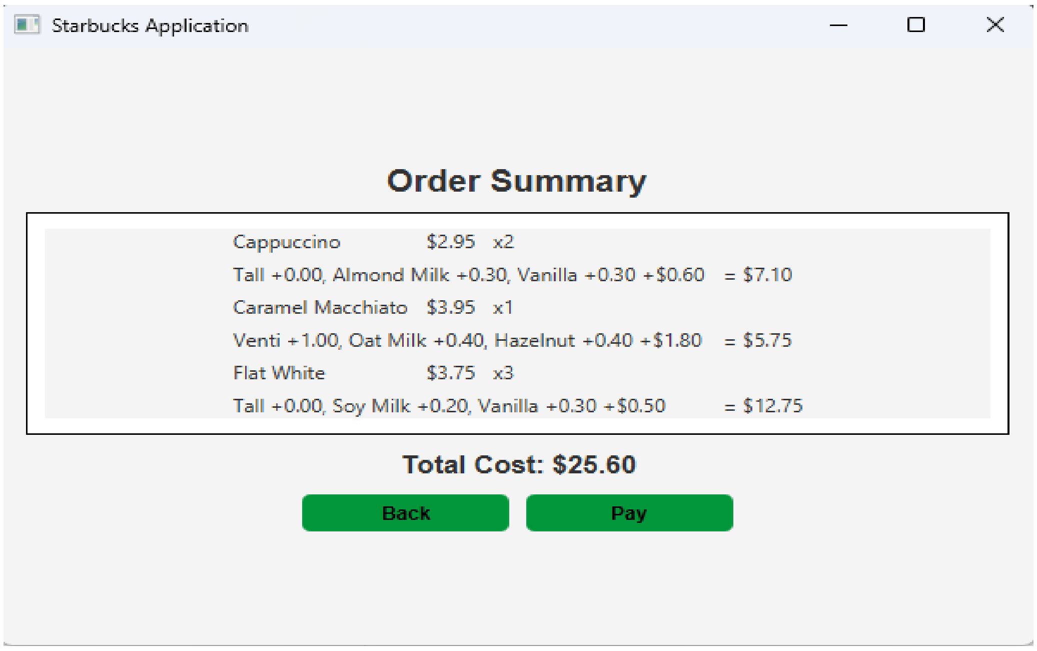
Name	Price	Description
Caffè Mocha	3.45	Espresso with bittersweet mocha sau...
Cappuccino	2.95	Dark, rich espresso under a smoothe...
Espresso	1.95	Rich and caramelly espresso in its pur...
Flat White	3.75	Smooth ristretto shots of espresso an...
Latte Macchiato	3.65	Layered espresso with steamed whol...
Caramel Macchiato	3.95	Freshly steamed milk with vanilla syru...

Order

Item Name	Item Price	Customization Cost	Quantity	Total Price
Caffè Mocha	3.45	1.20	1	4.65
Cappuccino	2.95	0.60	2	7.10
Caramel Macchiato	3.95	1.80	1	5.75

Total Cost: $17.50

OrderProcessing view :



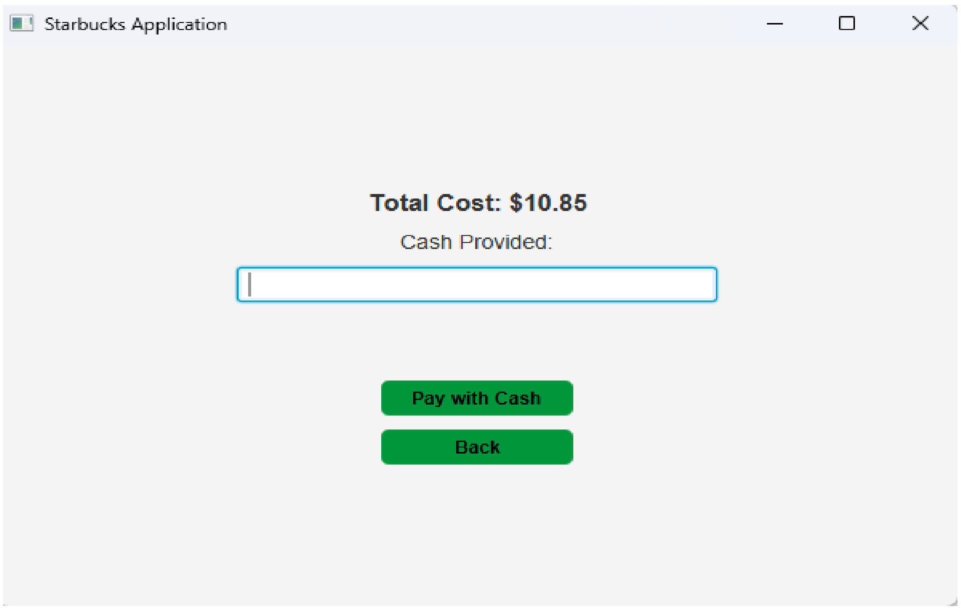
Order Summary

```
Cappuccino          $2.95   x2
Tall +0.00, Almond Milk +0.30, Vanilla +0.30 +$0.60   = $7.10
Caramel Macchiato   $3.95   x1
Venti +1.00, Oat Milk +0.40, Hazelnut +0.40 +$1.80    = $5.75
Flat White          $3.75   x3
Tall +0.00, Soy Milk +0.20, Vanilla +0.30 +$0.50      = $12.75
```

Total Cost: $25.60

PaymentHandler view :

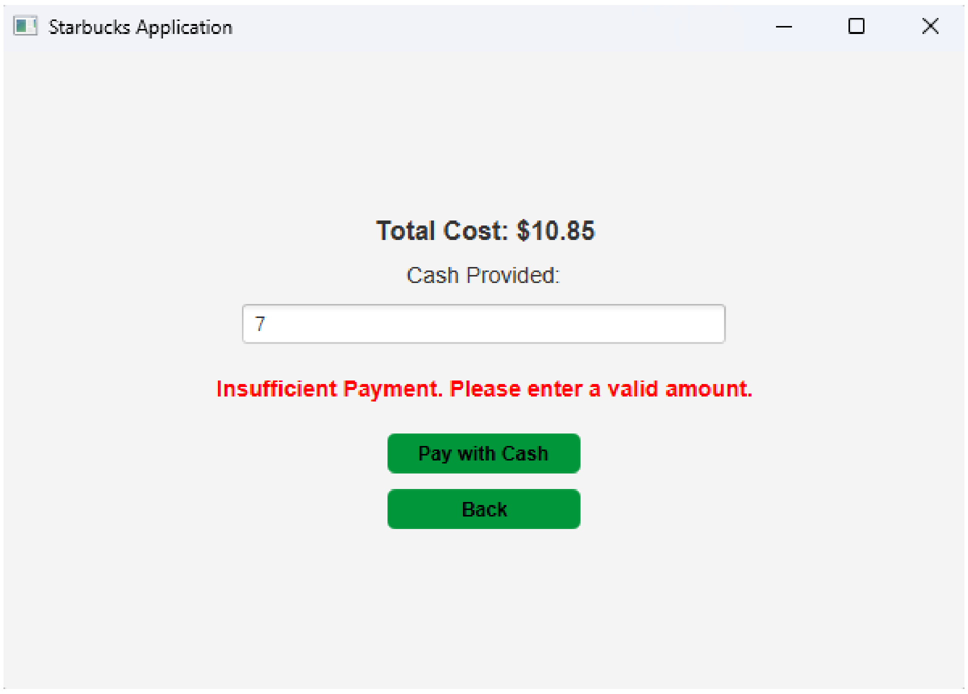
Guest :


Starbucks Application

Total Cost: $10.85

Cash Provided:

Guest Payment Validation :


Starbucks Application

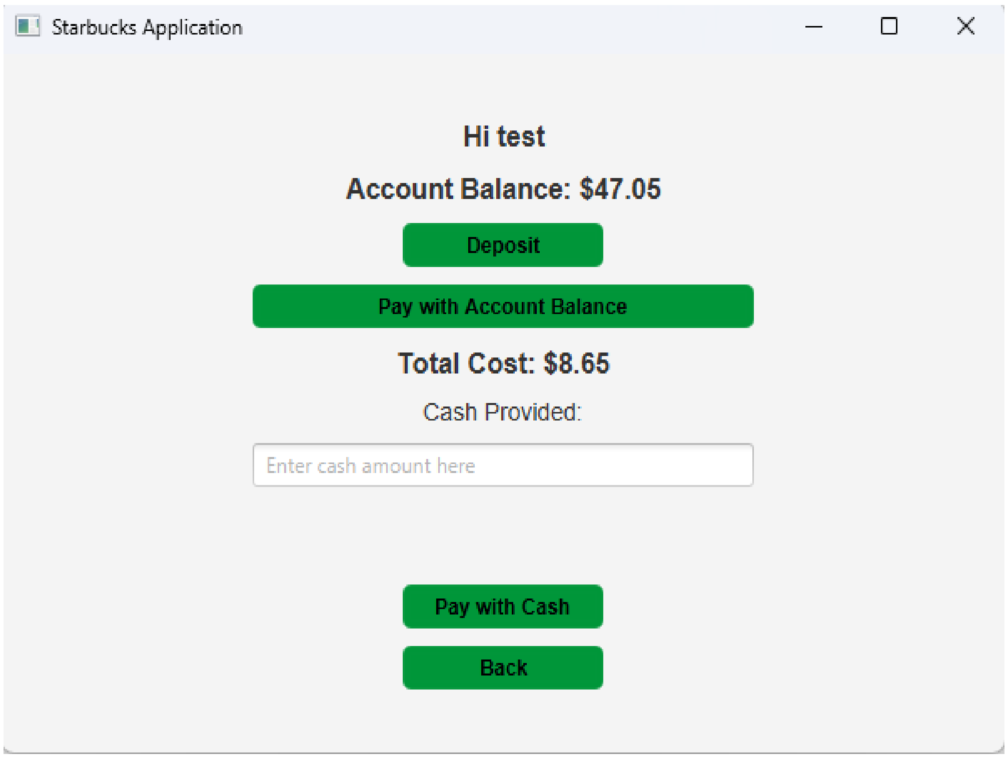
Total Cost: $10.85

Cash Provided:

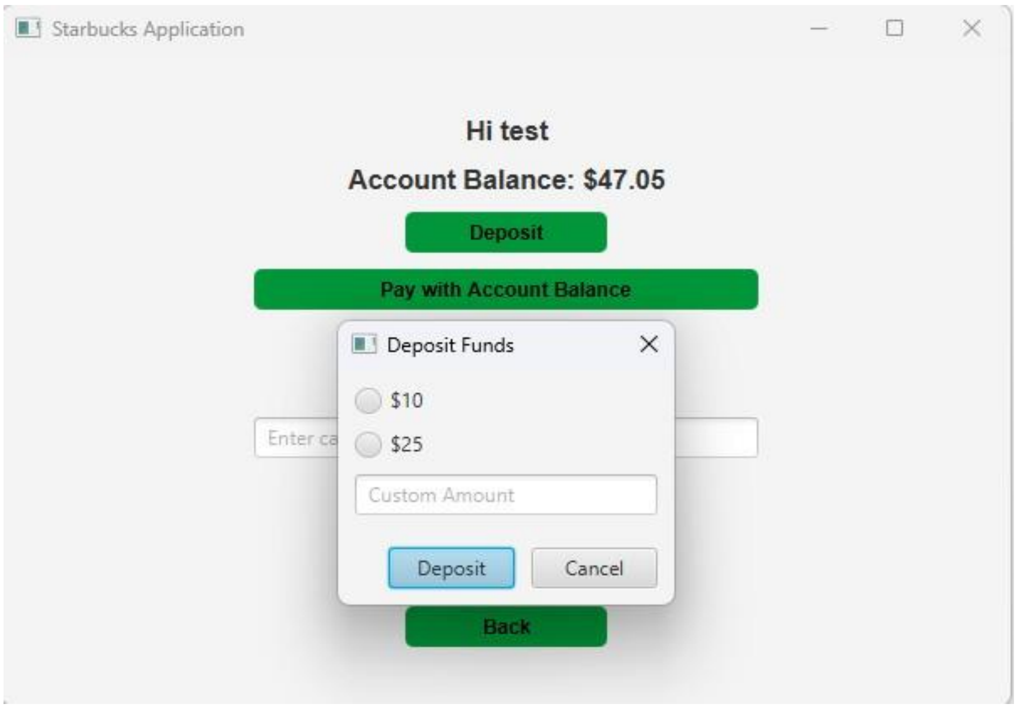
7

Insufficient Payment. Please enter a valid amount.

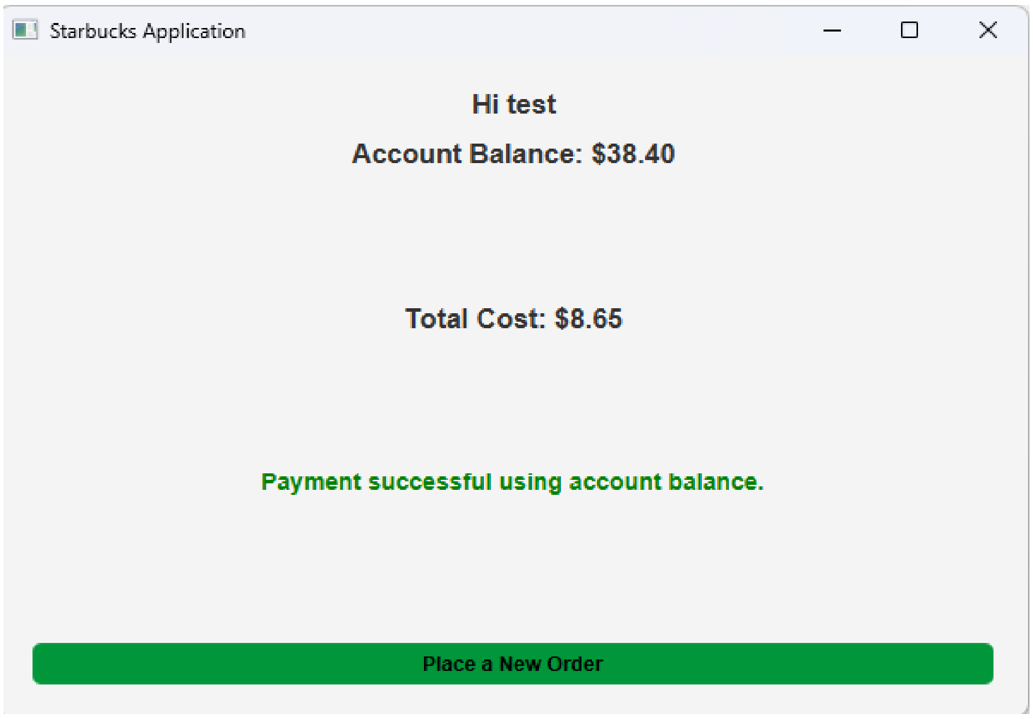
Logged-In User :



Logged-In User Deposit :



Logged-In User Payment processed successfully :



Hi test

Account Balance: $38.40

Total Cost: $8.65

Payment successful using account balance.