|  |  |
| --- | --- |
| **Project – External Documentation** |  |
| ISYS6197003  Business Application Development |
| Odd Semester Year 2023 |

# Project Title

SeRuput Teh’s Inventory & Transaction Management System

# Introduction

SeRuput Teh is an expanding store specializing in premium tea products. SeRuput Teh aims to enhance the customer experience by launching a digital platform. The owner has given you their trust to develop a Java application to manage SeRuput Teh’s inventory and transactions.. The app has two types of users: customers and admins.

Customers can easily buy tea. They pick the tea they want, put it in their cart, and then check out. It's a straightforward way for them to get their favorite tea. Admins, on the other hand, have more control. They can add new teas, update existing ones, see what's there, and delete things if needed. This helps keep the store's inventory organized.

# 

# Report / Documentation

## Login Scene

The login feature in this JavaFX application utilizes a BorderPane as the base layout manager. The primary user interface is constructed using a GridPane positioned in the center of the BorderPane. This GridPane serves as the central form for user login.

**Event Handlers**:

1. Hyperlink Event: Clicking on a hyperlink triggers an event that redirects the user to the registration page. This provides a seamless transition for users who need to create an account.
2. Login Button Event: When the login button is clicked, an event handler is invoked. This event handler calls the `isValid` method to validate the entered username and password

**Methods:**

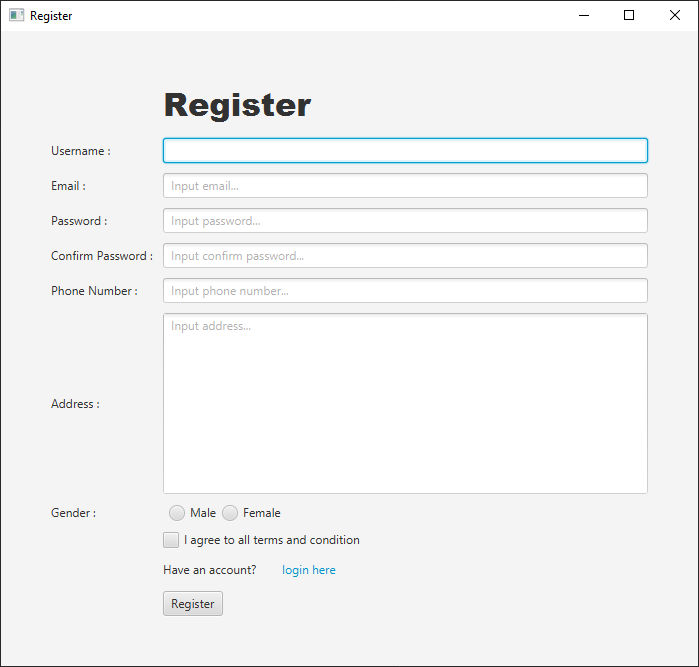
isValid Method:

The `isValid` method performs the validation of the provided username and password. It returns a boolean value based on the success of the validation. If the credentials are valid, it initializes the `homePageView` constructor and changes the primaryStage's scene to the homepage scene. If the credentials are invalid, an error message is displayed.

**Components**:

* Base Layout Manager: BorderPane
* Middle Form: Constructed using GridPane
* Event Handlers: Hyperlink and Login Button
* Validation: Handled by the `isValid` method, which validates the username and password.
* Page Navigation: Successful login transitions to the home page; unsuccessful login displays an error message.

## Register Scene



The Register Scene uses two classes: RegisterView and RegisterController. The RegisterView class defines the display of the scene, using a BorderPane as the root layout. The form is laid out using GridPane centered in the BorderPane. This page serves as a way for customers to register and create their accounts in the database.

The RegisterController handles the logic of the Register Scene, namely handling buttons, inserting data into the database, and validating the data.

**Event Handlers, as defined in the RegisterController:**

1. Hyperlink Event: Clicking on a hyperlink triggers an event that redirects the user to the login page, inversely to the Login Scene.
2. Register Button Event: Clicking the Register Button will call the validation methods for each field, as according to requirement. If the fields pass the validation, the data is inserted into the database, and the user will go to the Login Scene. If not, an error popup will show, displaying the validation failures.

**Methods:**

1. RegisterController:
   1. showSuccessPopup()

If called, show an Alert displaying the registration is successful.

* 1. goToLogin()

Create a new LoginView class and show it in the stage.

* 1. completeRegister()

Inserts the registration data into the database using prepared statements.

* 1. getUser()

Gets a list of users from the database. Returns an ArrayList of users.

* 1. generateID()

Generates an ID based on the highest number of the ID in the database, then increments it by one. Returns a string of the ID.

* 1. validateEmail(), validateUsernameUnique, etc.

Validates the associated TextField according to the requirements. If validated, returns true (boolean).

* 1. showErrorPopup()

Show an error Alert, listing the validation failures.

1. RegisterView

showRegisterScene()

Displays the main stage, by creating the root, BorderPane, and calling createRegisterForm(), to create the center content.

**Components:**

1. Label: For showing the title, and show the TextField’s purpose.
2. TextField: For user text input.
3. RadioButton: For user choice input.
4. CheckBox: For user checkbox input.
5. Button: For calling the event handler when clicked by the user to register,
6. HyperLink: For calling the event handler when clicked by the user to move to Login.

## Navigation Bar

The Navbar class provides two distinct navigation bars for users and administrators in a JavaFX application. It encapsulates the creation of MenuBars with associated menus and menu items, along with event handlers for each menu item.

### User Navigation Bar

The userNavbar method creates a MenuBar tailored for regular users. It includes menus for Home, Cart, and Account. The associated menu items allow users to navigate to the home page, view their cart, check transaction history, and log out.

### Admin Navigation Bar

The adminNavbar method generates a MenuBar specifically designed for administrators. It comprises menus for Home, Manage Product, and Account. Admins can navigate to the home page, manage products, and log out.

Methods:

1. public MenuBar userNavbar(Stage primaryStage, User userSession)

This method creates and returns a MenuBar customized for regular users.

Menu Items:

* **Home:** Navigates to the home page.
* **Cart:** Redirects to the user's cart.
* **Account:**
  + - **Transaction History:** Displays the user's transaction history.
    - **Log Out:** Logs the user out and redirects to the login page.

1. public MenuBar adminNavbar(Stage primaryStage, User userSession)

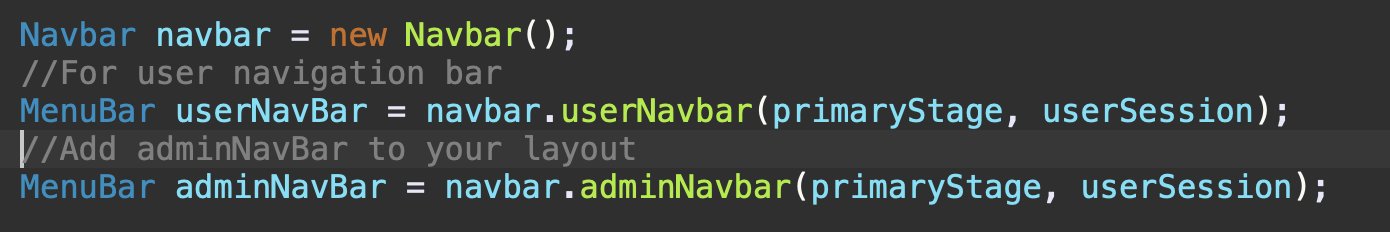
This method generates and returns a MenuBar tailored for administrators.  
Menu Items:

* **Home:** Navigates to the home page.
* **Manage Product:**
  + - **Manage Product:** Allows administrators to edit and manage products.
* **Account:**
  + - **Log Out:** Logs the admin out and redirects to the login page.

Event Handling

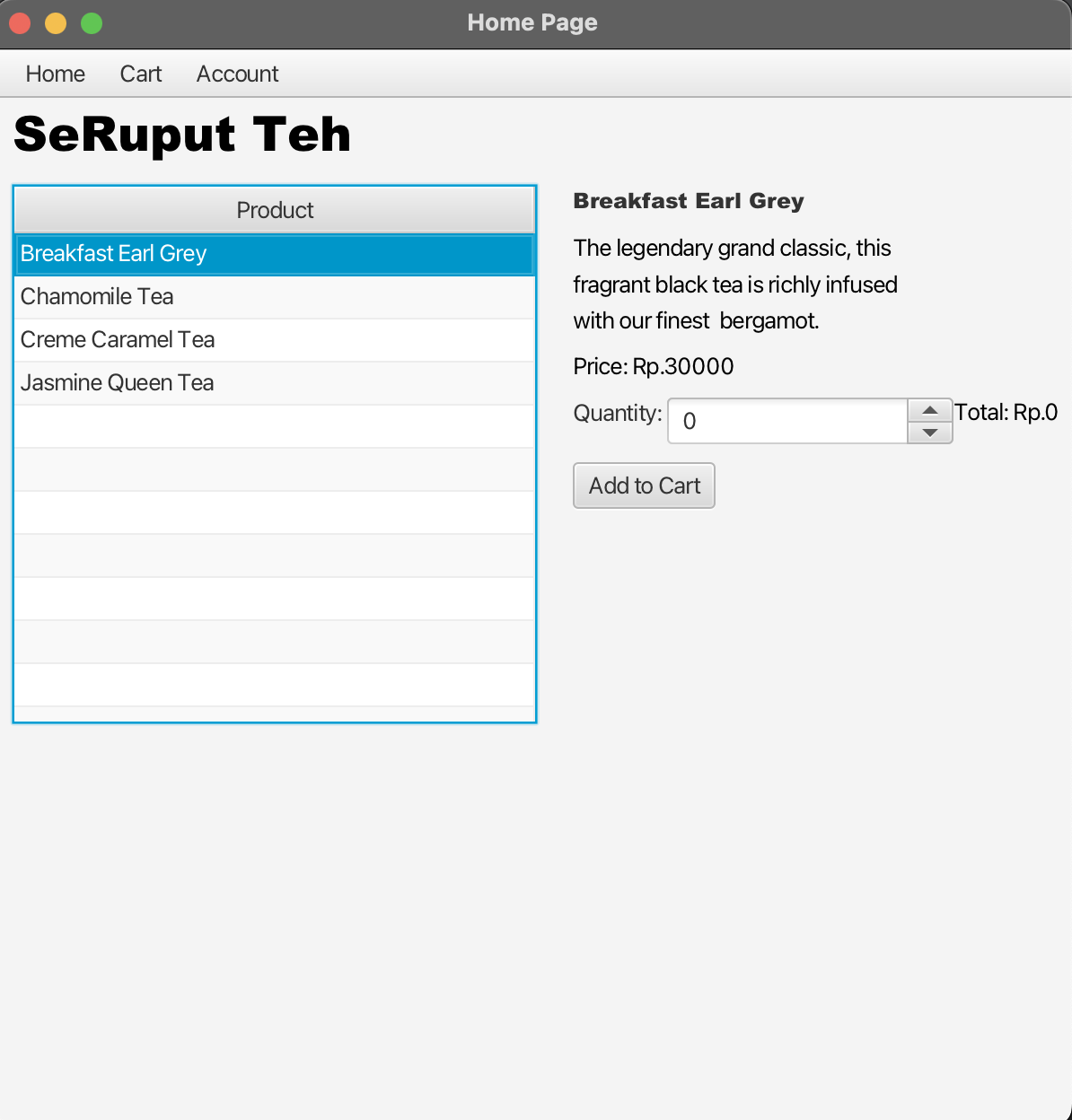
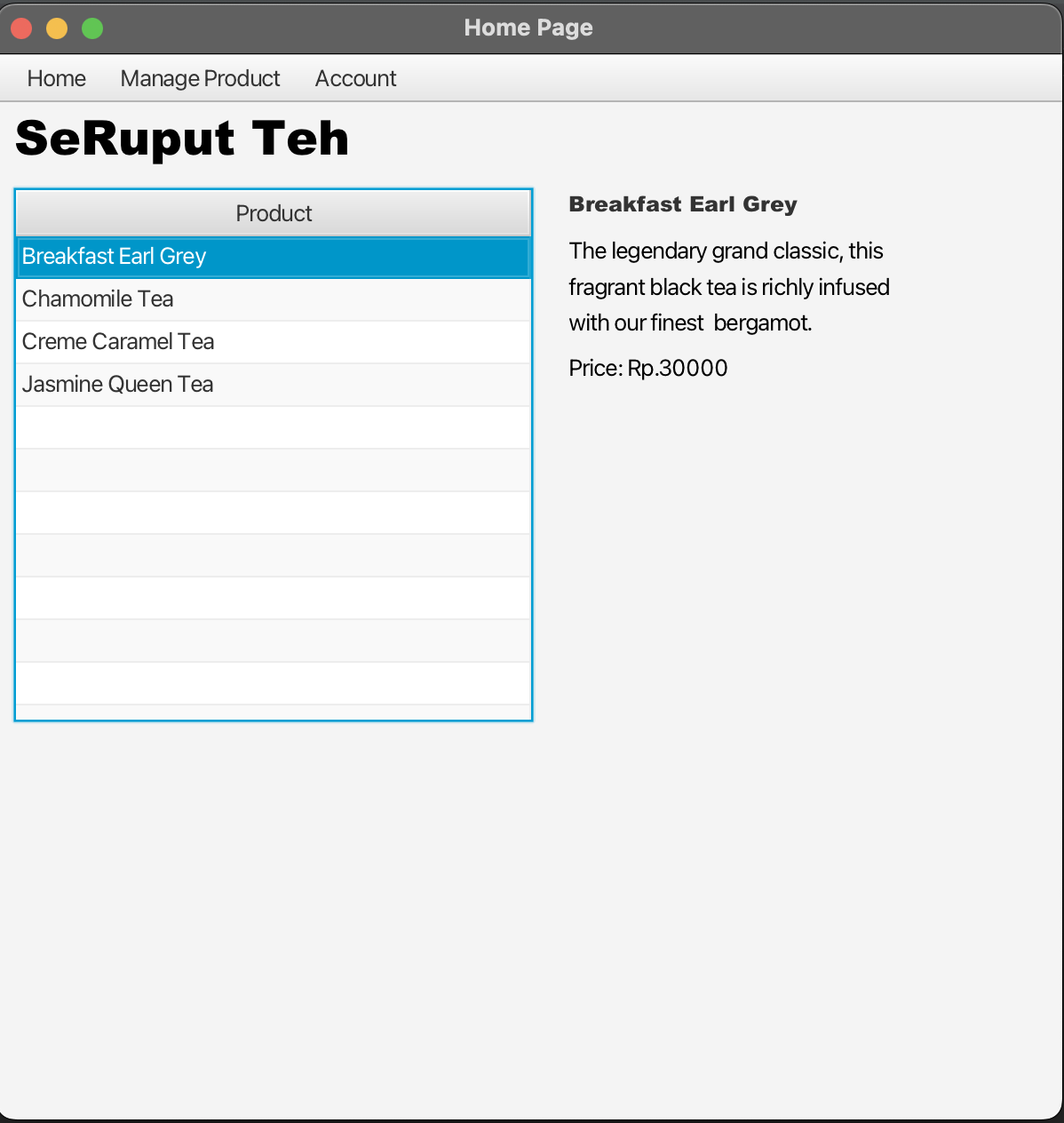
Event listeners are added to each menu item to define the actions taken when an item is selected.

* **Home Page Item:** Navigates to the home page, creating a new HomePageView instance.
* **My Cart Item:** Initiates the CartView for user cart visualization.
* **Transaction History Item:** Displays the user's transaction history through Transaction Scene.
* **Log Out Item:** Logs the user out, redirecting to the login page (LoginView).
* **Manage Product Item:** Allows administrators to manage products via EditProductScene.

How to implement the navigation bars in this application:

## Home Scene (Customer & Admin)

The HomePageView class represents the home page scene of a JavaFX application. It provides a user-friendly interface for both regular customers and administrators. The class dynamically adapts its behavior based on the user's role, introducing specific features for each role.



**Constructor**

public HomePageView(Stage primaryStage, User userSession)

Parameters:

* + primaryStage: The primary stage of the JavaFX application.
  + userSession: The user object representing the current user.

**Methods**

1. public void show()

* **Description:**
  + - Sets up and displays the home page scene.
    - Utilizes the Navbar class to create a navigation bar based on the user's role.

2. private BorderPane createCenterContent()

* **Description:**
  + - Constructs the central content of the home page using a BorderPane layout.
    - Displays a list of products in a TableView.
    - Provides details about the selected product on the right side.
    - Initializes event listeners for product selection and updating the right side.

3. private void displayProductDetails(Product product, VBox vBox)

* **Description:**
  + - Clears existing content in the VBox and displays detailed information about the selected product.
    - For regular customers, includes options to select quantity, calculate the total price, and add the product to the cart.
    - For administrators, displays basic product information without quantity and add-to-cart options.

4. private void addToCart(Product product, Integer value)

* **Description:**
  + - Adds the selected product to the user's cart with the specified quantity.
    - Handles database operations to update the cart.
    - Displays success or error alerts based on the operation's outcome.

5. public List<Product> populateTableView()

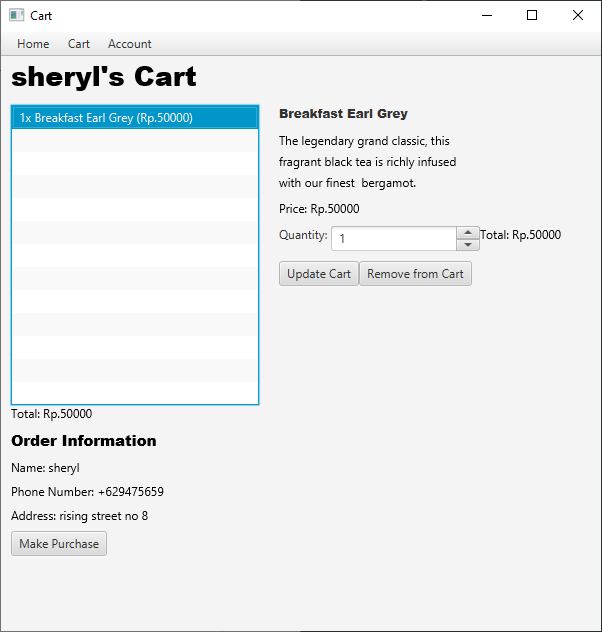
* **Description:**
  + - Retrieves a list of products from the database to populate the TableView.
    - Returns a list of Product objects.

**Role-Based Behavior**

1. **Regular Customer (Role: "Customer"):**
   1. Displays navigation options for Home, Cart, and Account.
   2. Allows the user to view products, select a product for detailed information, and add the product to the cart.
2. **Administrator (Role: "Admin")**
   1. Shows navigation options for Home, Manage Product, and Account.
   2. Displays basic product information without quantity and add-to-cart options.
   3. Provides options for managing products, such as editing and updating.

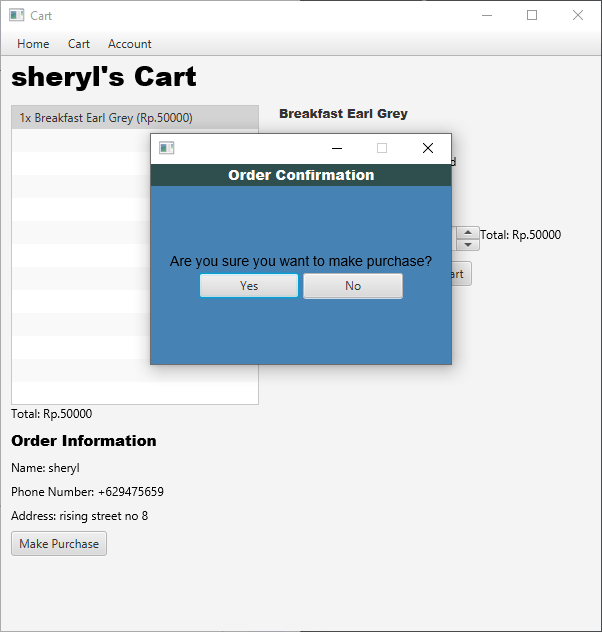
## Cart Scene

Cart Scene without any items in the cart



Cart Scene with an item in the cart and selected

The Cart Scene is defined by the CartView class. The CartScene allows customers to check their orders, adjust item quantities or delete items in the cart, and make purchases.



Cart Scene after Make Purchase is pressed.

The CartView class handles both displaying the scene and handling the logic of the scene. The CartView uses a VBox as the root layout, and BorderPane to display the center content. The VBox contains the Navbar and then the BorderPane.

**Event Handlers:**

1. List Listener: When an item is clicked in the list, the VBox on the right of the ListView changes, displaying the details of the product chosen by the customer.
2. Update Cart Handler: When clicked, add the value in the spinner to the quantity of the item selected in the cart. If the quantity equals zero, the item in the cart is deleted.
3. Delete Cart: When clicked, deletes the item in the cart from the database.
4. Make Purchase Handler: When clicked, display a confirmation popup. If “Yes” is selected, create a new TransactionHeader and TransactionDetail rows in the database, and delete all items in the customer’s cart from the database.
5. Y/N Handler: In the confirmation popup, if yes is pressed, proceed with the transaction. If No is pressed, close the popup.

Methods:

1. showCartView()

Displays the scene by creating the Navbar, the center content by calling createCenterContent(), and the root layout, then insert the Navbar and center content into the VBox root, then show.

1. purchasePopUp()

Create a new stage, then add the Yes/No Buttons. If yes is pressed, proceed with the transaction. If No is pressed, close the popup.

1. purchase()

Inserts the data into the database by first generating an ID, then insert the ID and userID into TransactionHeader. Then select all the cart items of the customer’s in the database, then insert it in the while loop to the TransactionDetail with the TransactionID. FInally, delete the items in the cart.

1. generateID()

Generates a TransactionID based on the highest number of the TransactionID in the database, then incrementing it by 1 (One). Returns a string of the ID.

1. displayProductDetails(CartList cart, VBox vBox)

Called by the Listener in the ListView, this method displays the details of the product selected. It first clears the vBox on the right, then displays the item’s details.

1. updateCart(CartList cart, Integer value)

This method updates the quantity of the item selected in the cart, by adding the number in the spinner to the quantity. If the resulting quantity is equal to zero, the item is deleted. If less than zero, an error popup will be shown, and no changes will be made. If there’s no error, update the database, then show a success popup.

1. populateListView()

This method populates the ListView with the items in the cart of the customer’s from the database. Returns a List of CartList items.

1. createCenterContent()

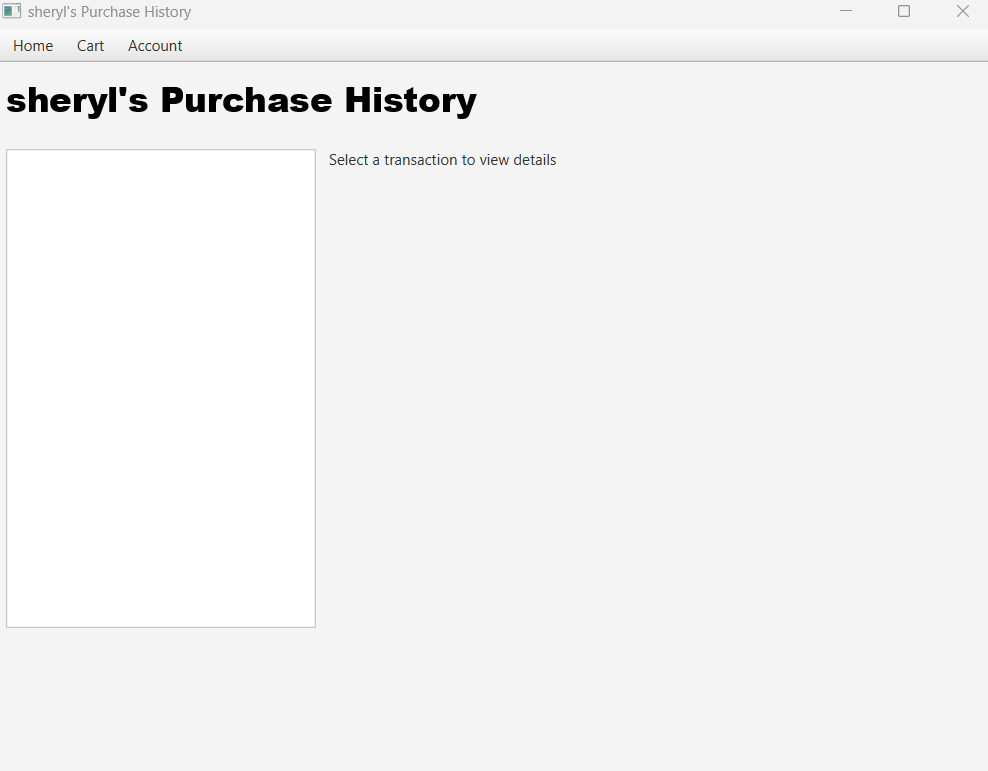
Creates the main display, showing customer details with createUserDetailContent() method, the ListView, and the VBox on the right for showing item details.

**Components:**

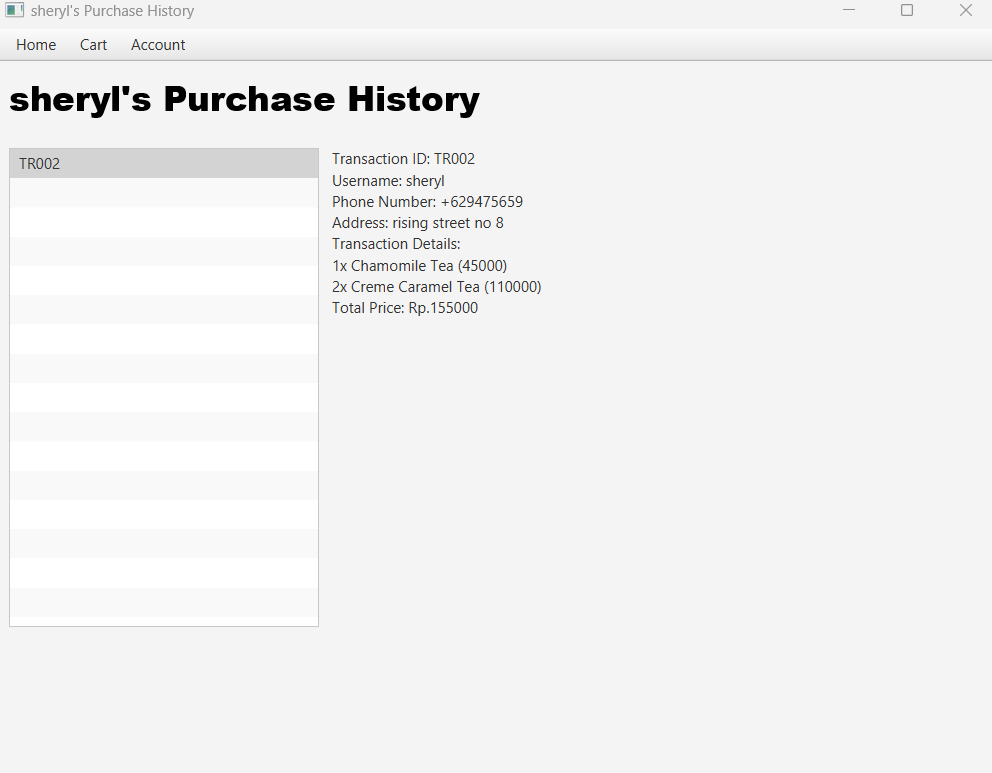
1. Navbar: For user navigation to other pages.
2. ListView: For displaying cart items.
3. Label: For displaying information.
4. Button: For calling event handlers when clicked by the user.
5. Spinner: For user input for integers with up/down arrows to increment/decrement.

## Transaction Scene (Purchase history)

Transaction Scene without any content inside the table



Transaction Scene with content inside the table and selected:



The Transaction Scene class provides a user interface to view the purchase history of a user. The scene is divided into two main sections: the top section displaying user information and a navigation bar, and the center section presenting a list of transaction IDs and their details.

**Event Handlers**

List View Selection

When a user selects a transaction from the ListView, the system fetches and displays detailed information about the selected transaction.

**Methods**

* fetchTransactionIDs

**Description**: Retrieves a list of transaction IDs for the current user from the database. Returns: List of transaction IDs.

* fetchTransactionDetails

**Description**: Retrieves detailed information about a specific transaction, including user details, phone number, address, and a breakdown of purchased products.

**Parameters**: transactionID (The ID of the selected transaction).

**Returns**: A formatted string with transaction details.

* fetchTransactionDetailsList

**Description**: Retrieves a list of transaction details (product ID and quantity) for a specific transaction.

**Parameters**: transactionID (The ID of the selected transaction)

**Returns**: List of TransactionDetail objects.

* fetchProductName

**Description**: Retrieves the name of a product based on its ID.

**Parameters**: productID (The ID of the product)

**Returns**: The name of the product.

* fetchProductPrice

**Description**: Retrieves the price of a product based on its ID.

**Parameters**: productID (The ID of the product)

**Returns**: The price of the product.

* fetchTotalPrice

**Description**: Calculates the total price of a transaction based on the sum of product prices multiplied by their quantities.

**Parameters**: transactionID (The ID of the selected transaction)

**Returns**: he total price of the transaction.

**Components**

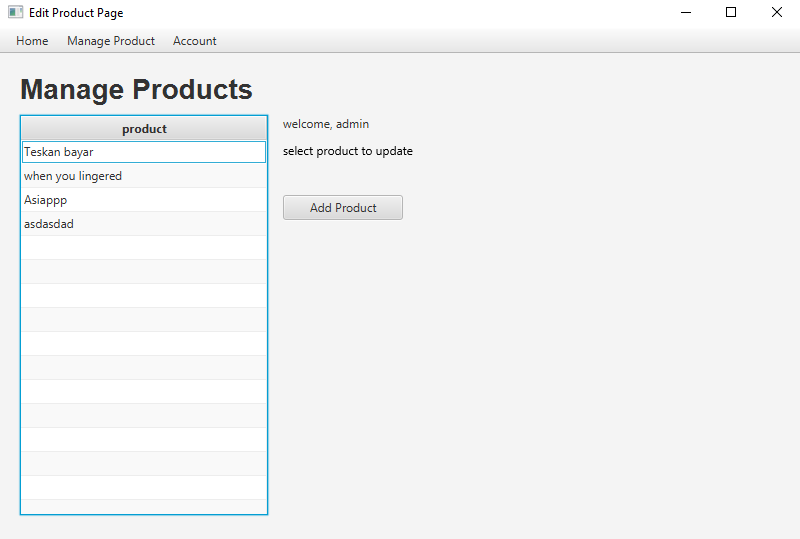
Top Section:

* + Navbar: Provides navigation options for the user.
  + Username Label: Displays the user's name and the purpose of the page.

Center Section:

* + ListView: Displays a list of transaction IDs for the user to choose from.
  + Label: Displays instructions or detailed information about the selected transaction.

## Edit Product Scene



"Edit Product Scene" is a page designed for adding, updating, and removing products. Before accessing this Edit Product Scene page, the program will check the user's role. If the user's role is not admin, they will not be able to access the page. The "Update" and "Delete" buttons for products will be visible only after the admin clicks on one of the items, allowing them to perform the desired actions.

**Event handlers**

* Add product toggle button: make other buttons invisible except for add button (for update and delete),toggle the product name text field; price text field; and description.
* Update product toggle button: make other buttons invisible except for update button (for add and delete), toggle the update product price.
* Remove toggle button: make other buttons invisible except for remove button (for add and update), toggle the remove confirmation.
* Back button: to make the previous state of an element visible again.

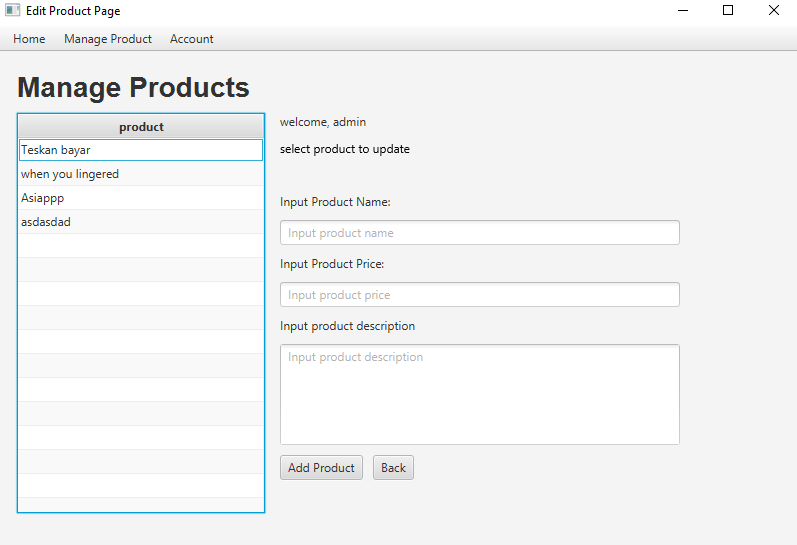
**Methods**

Retrieve record:  
The retrieve record function has no parameters, hence it only acts as a function that will return an arraylist of products. The retrieve record will select all the products from the database system, and then insert the product record to an array list. And finally the function will return the array list of products.

**Components**

* BorderPane: Used for layout organization, dividing the UI into regions such as top, right, bottom, left, and center.
* TableView<Product>: Represents a table for displaying a list of Product objects with rows and columns.
* TableColumn<Product, String>: Defines a column in the TableView, specifying how to display the string attribute of the Product class.
* ObservableList<Product>:A dynamic list that notifies UI components, like TableView, of changes when elements are added, removed, or modified.
* HBox and VBox: Layout containers organizing UI components horizontally (HBox) or vertically (VBox).
* Buttons, ToggleButton, Label, and Field: UI elements for user interaction and information display, such as triggering actions with buttons, toggling options with ToggleButton, showing text with Label, and receiving input with Field.

## Add Product



"Add Product" is an integrated feature within the broader context of the "Edit Product Scene." This feature serves as a functionality for users to input and submit information about a new product, facilitating the addition of this new product to the database system. Essentially, it provides a means for users to contribute new entries to the existing product database through the user interface.

**Event Handlers:**

* Add Product Button: Add product button will listen to click event, after detecting click from user, the program will get the name,price, and description field. The button will call the insertProduct function and insert the product to the database.
* Back button: change to previous state

**Methods:**

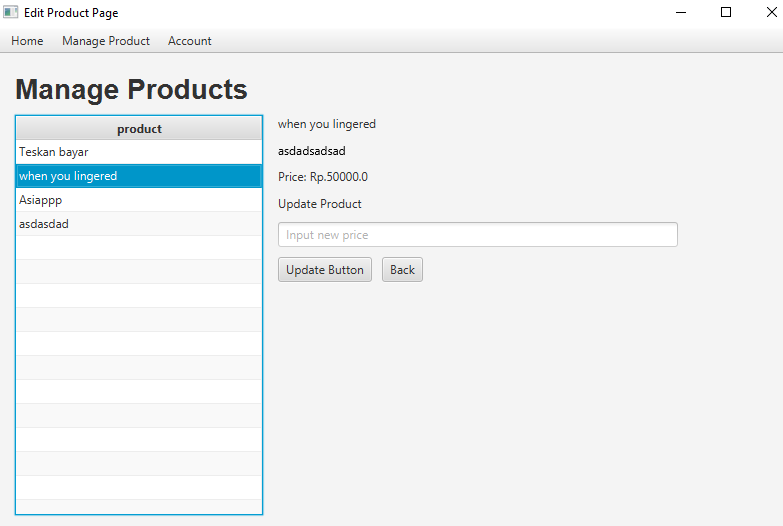
Insert Product:

Insert product function is used to create a new product record to our database system. After the user fills the input, such as name,price text , and description , we use a validation to check whether or not the input is filled correctly. After passing the validation, the product is created as a new record from the database system.

**Components:**

* Label: used for containing a text in “Add Product”.
* TextField: this component used for a text input, allowing users to enter string information into the application. From this text field users can input product name,price, and description.
* Buttons: There are two buttons components in “Add Product” which is Add Product, and Back. The button will act as an event listener to listen clickable event from user

## Update Product



"Update Product" is another functionality that’s integrated from "Edit Product Scene." This feature allows users to modify and submit a new price for an existing product. It offers a user-friendly interface for users to modify the existing product price to the database entries.

**Event handlers:**

* Update Product Button: Update product button will listen to clickable events from the user, after that the program will get the field price value input by the user. After getting the value from the field price , the program will run a function from updateProduct, and if the input passed the validation, it will update the product to the database.
* Back button: change to previous state

**Methods:**

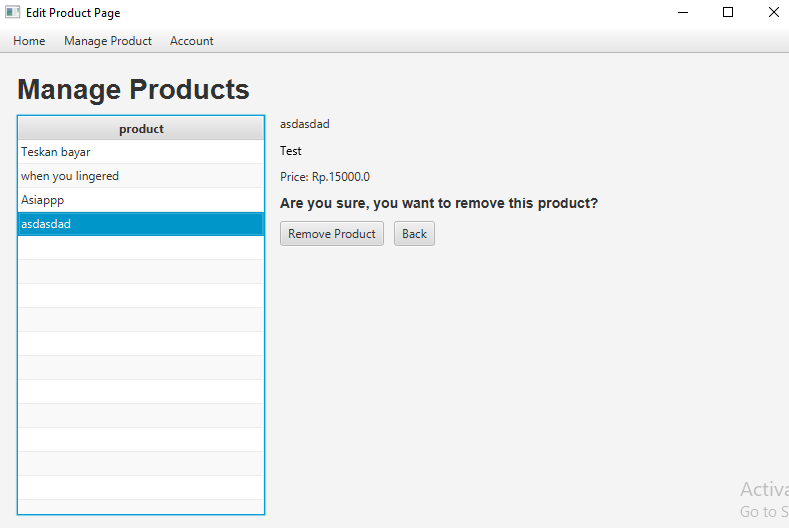
Update product:

For update product function we have a "productID" and "priceText" as function parameters, the "productID" is to locate the selected product for updating by the user. Even though we are using “priceText” for users to input an updated price as a string , the "priceText" , however we have implemented a condition to verify whether the price is either empty or not a numerical value. After passing the validation, the product is then updated based on the product ID retrieved from the database system.

**Components:**

* Label: for containing text for “Update Product”, such as the product title,product description, and product price.
* TextField: this component used for input a new product price from the user. Even though we use TextField as price input, this text field will get validated later on update product method.
* Buttons: UI elements for user interaction and use for clickable events, in “Update Product”, we use Buttons for Update Button, and Back button.

## Remove Product



"Removing a Product" is another feature from “Edit Product Scene”. This functionality allows users to effortlessly delete an existing product from the database, offering an intuitive and user-friendly interface for removing outdated or discontinued items.

**Event handlers:**

* Remove Product Button: remove product button will run a remove product function and pass the product ID as parameters to that function. The product ID gets from the user selected product.
* Back button: change to previous state

**Methods:**

Remove product:

Remove product function is to remove a product selected by the user. User will select the product and click the remove confirmation to remove the product. The function that will take the product Id to find which product record to remove, and finally remove the product from the database system.

**Components:**

* Label: label components used in “remove product” for displaying product name, description , price and other text content (confirmation reminder).
* Buttons: There are two buttons components in “remove product” which is the remove product button (to listen to user clickable events and eventually remove the product) and also the back button to return to the previous state of content.
* **Reference**
  + <https://www.acs.ase.ro/Media/Default/documents/java/ClaudiuVinte/books/ArnoldGoslingHolmes06.pdf>
  + <https://accelconf.web.cern.ch/icalepcs2019/doi/JACoW-ICALEPCS2019-MOPHA173.html>
  + <https://link.springer.com/book/10.1007/978-1-4842-7848-2>
  + <https://dl.acm.org/doi/abs/10.5555/3015063.3015087>
  + <https://books.google.com/books?hl=id&lr=&id=EnWVnPb3ixQC&oi=fnd&pg=PR3&dq=java+database+mysql&ots=TYSKN22bU4&sig=vG6eGRjFcavG1pM55aE-X4NioBk>
* **Group Member**

1. 2602057570 – EDWIN TJANDRAATMADJA
2. 2602052632 – NICHOLAS FRANSISCUS HUANG
3. 2602051094 – JONATHAN SAMUEL TANADI
4. 2602061492 – AVNER RAPHAEL GUNARTO