RESTAURANT MANAGEMENT SYSTEM

EU JIA XIN

Contents

Introduction

This report aims to cover the design and developing process of a restaurant management system. This system is requested by the restaurant who wishes to manage themselves effectively via a system especially during the Covid19 pandemic to minimise contact. The goal of this system is to provide a platform for paperless order taking and payment including contactless payment, managing inventory system and staff of the restaurant. Implementation of the system not only help maintains safe social distancing; it also greatly decreases the workload of the staff members so less staff needs to be hired.

The operational concepts of the system are to be compatible on both mouse and keyboard device or device without mouse input, login function with masked password where every user information is stored in database, and user with special access can register or de-register user from the system where the details can be added or deleted from the database. User with special access can also alter, remove or add item in the menu, which is stored in a binary file. Registered user or customer as guest can place order and complete the payment process after total price is calculated, with considerations like extra charges, service charge and discount. The details of code put together to program the system will be further covered in the later section of this report.

Designing the User Interface

Main Theme

The main colours used as the theme of this system are :

#05CEE6	#d9c1ff	#ffffff	#E43BE1

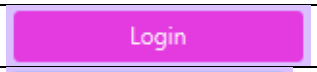
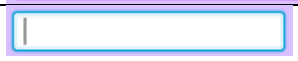
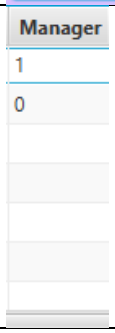
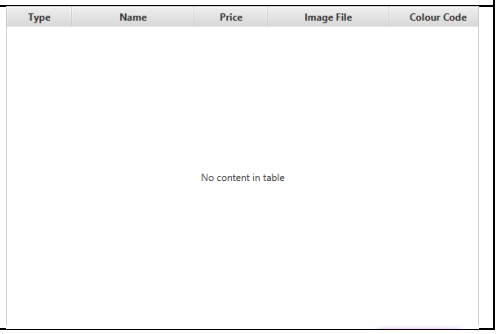
Main Pane

The main panes used to build the scenes in this system are:

AnchorPane	BorderPane	HBox	VBox	ScrollPane	GridPane

Main Component

The main components used to make up the scenes in this system are:

Label		Button	
MenuButton		TextField	
PasswordField		CheckBox	
TableColumn		TableView	

Scene

The rough design of how scenes will look like in this system are designed using Storyboard and actual scenes used in the application are built on Scene Builder

(*Both rough design and design on Scene Builder may not resemble the exact layout of the scene as photos cannot be uploaded to Storyboard and some scenes built on Scene Builder are initialized when loaded)

- **Start-Up Scene:** The first scene to show on stage when application is launched, this is where the restaurant's name and logo is displayed and where user will choose their identity to access the system

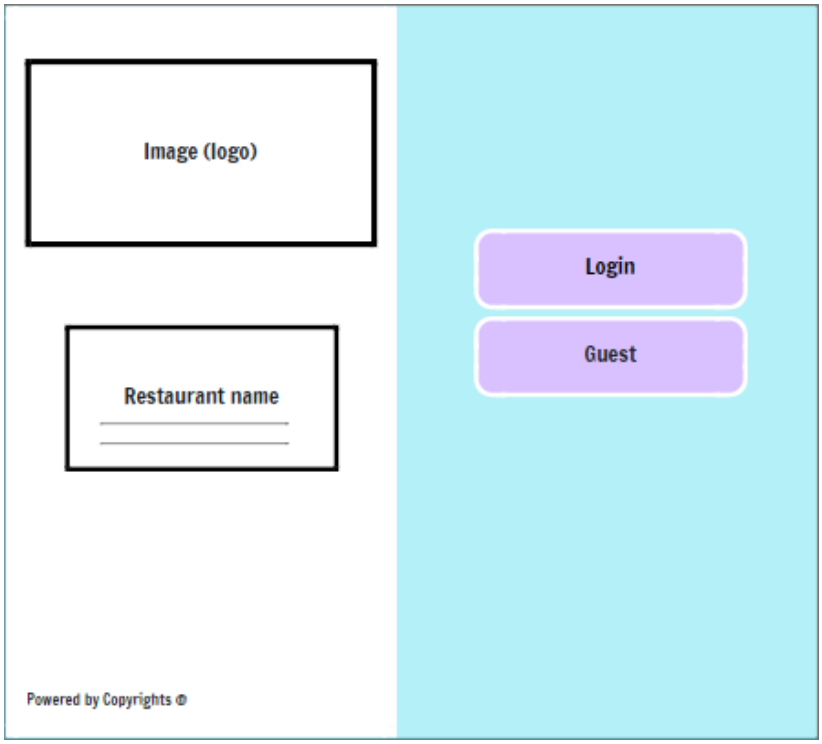


Figure 2.1: Start-up page designed on Storyboard

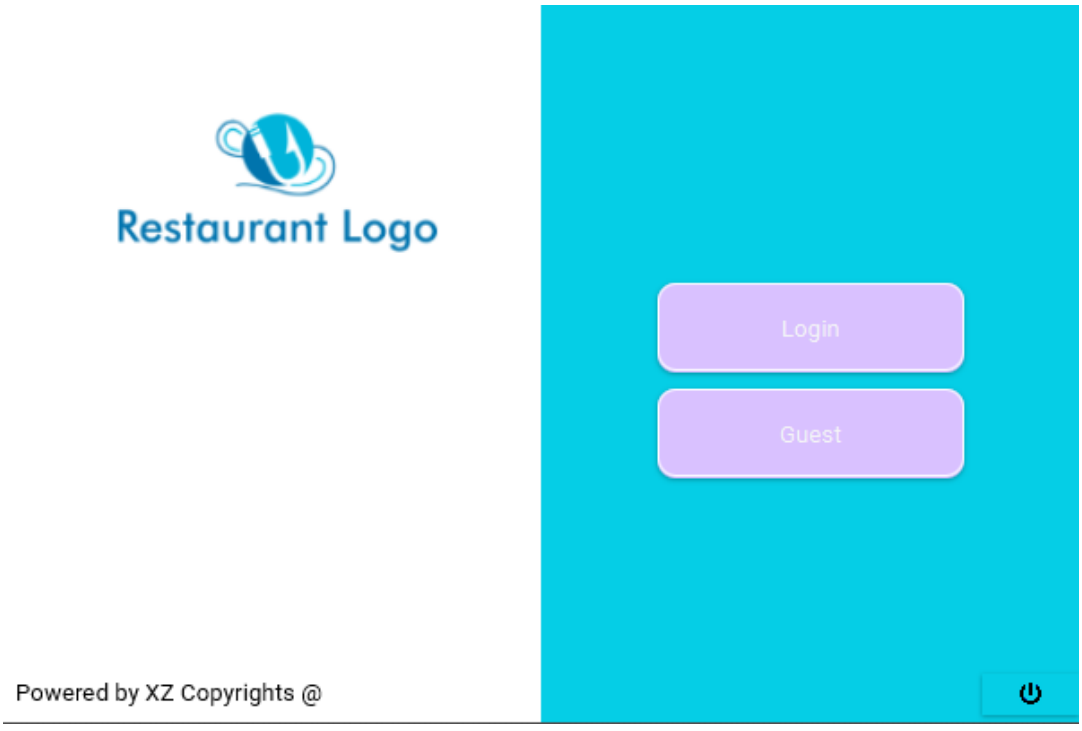


Figure 2.2: startup.fxml built on Scene Builder

Login button- navigates user to login page

Guest button- navigates user to menu

Power off button- close stage and application

- **Login Scene:** User will enter the username and password to login

Figure 2.3: Login page designed on Storyboard

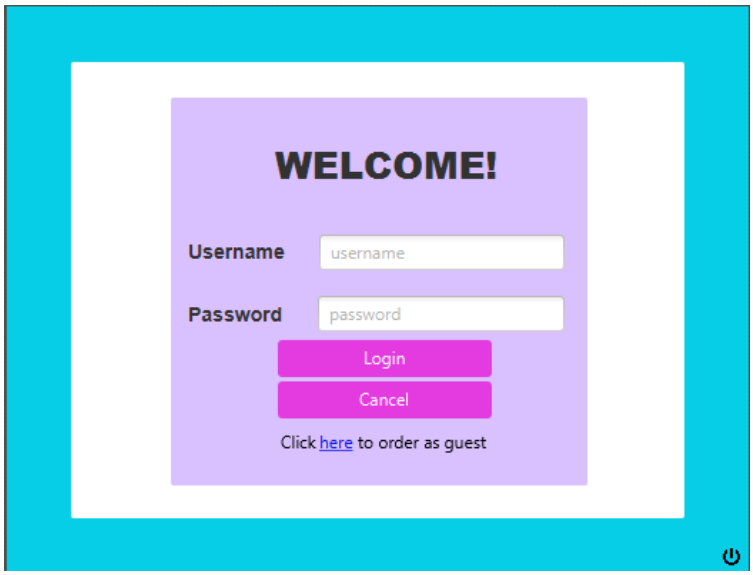


Figure 2.4: login.fxml built on Scene Builder

Username textfield: user prompted to type username, same function as login button if ENTER key pressed

Password passwordfield: user prompted to type password, password will be masked, same function as login button if ENTER key pressed

Login button: checks if both fields are filled, take text in fields and validate if detail entered matches any data in database, if matches then authenticate if registered user is granted special access. User with special access is navigated to manager page, user without will be brought to menu, error message will be shown if details entered do not match any data

Cancel button: navigates user to start up page

Here button: navigates user to menu without signing in

Power off button: close stage and application

- **Manager Page:** where user with special access choose what operation they wish to perform

Figure 2.5: Manager Operations page designed on Storyboard

Figure 2.6: managerpage.fxml built on Scene Builder

Logout button: signs user out of account and navigates to start-up page

Register admin button: navigates user to register page

Deregister admin button: navigates user to de-register admin page

Manage inventory: navigate user to inventory page

Order button: navigates user to menu

- **Register page:** user with special access can register new user here by filling up the details

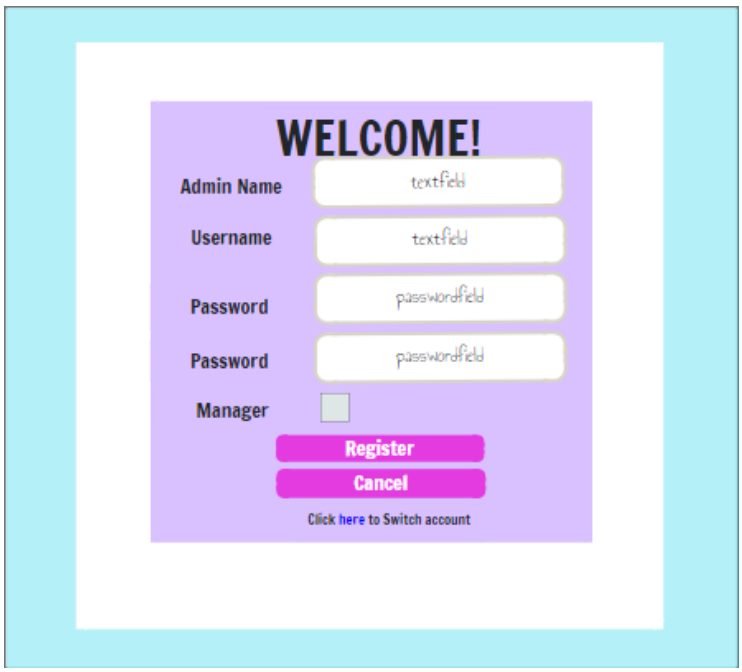


Figure 2.7: Register page designed on Storyboard

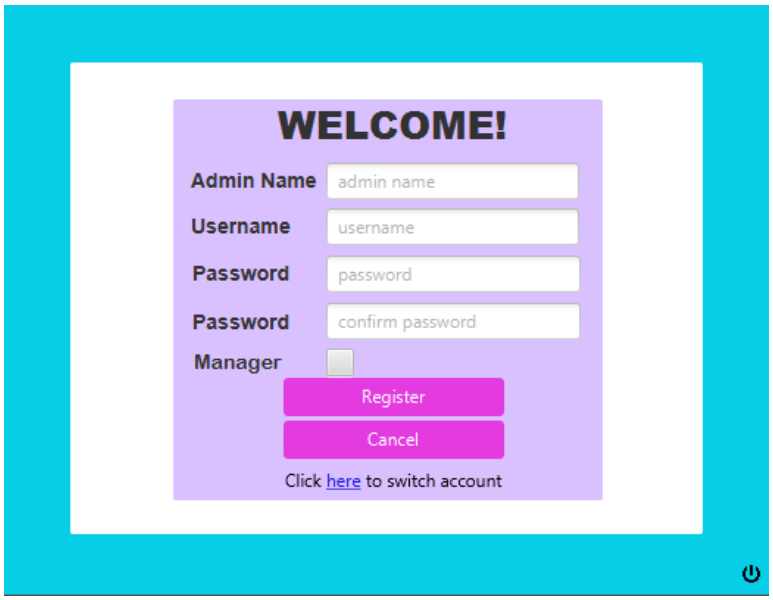


Figure 2.8: register.fxml built on Scene Builder

Admin name textfield: user prompted to type admin name, same function as register button if ENTER key pressed

Username textfield: user prompted to type username, same function as register button if ENTER key pressed

Password passwordfield, confirm password passwordfield: user prompted to type password, password will be masked, same function as register button if ENTER key pressed

Manager checkbox: user select check box if registering user with special access

Register button: checks if all fields have been filled, check if password is set at a safe length, check if password and confirm password contain the same text, convert Boolean value of checkbox to 0/1, insert all data into table in database where password is hashed and username must be unique. Error message is shown if any of the requirements is not met, navigates to manager page after successful registration

Cancel button: navigates to manager page

Here button: signs current user out and navigate to login page

Power button: close stage and application

- **De-register page:** View and select from all registered user to remove from database

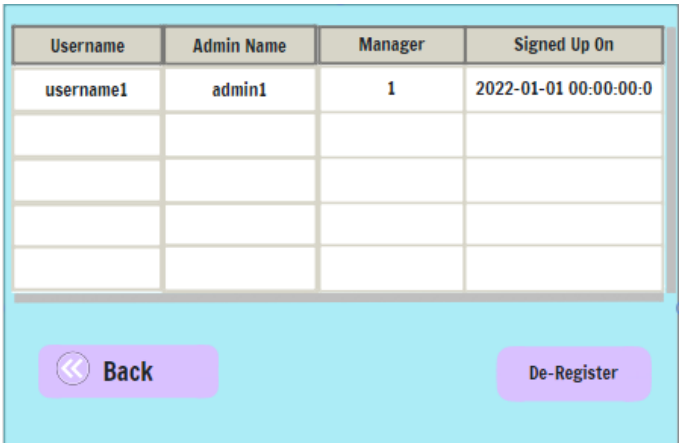


Figure 2.9: Deregister page designed on Storyboard

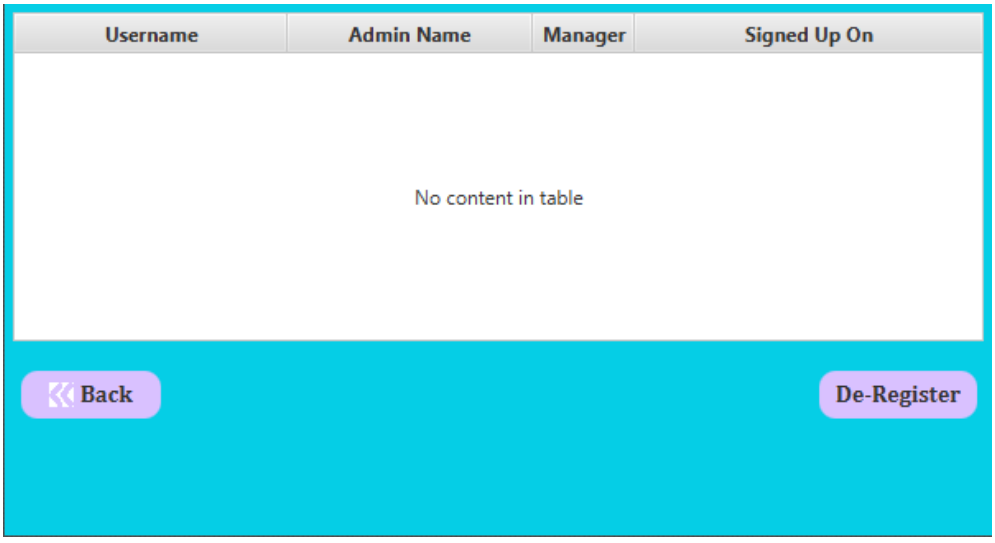


Figure 2.10: deregister.fxml built on Scene Builder

Table view: Data retrieved from database will be listed on specific columns in table when scene is loaded, data clicked on is selected

Back button: navigates user to manager page

De-register button: pop up window created to show new warning stage, if confirm, selected data will be deleted from database, updates table view

- **Inventory Page:** Lists all items in menu and each item can be deleted, modified or added

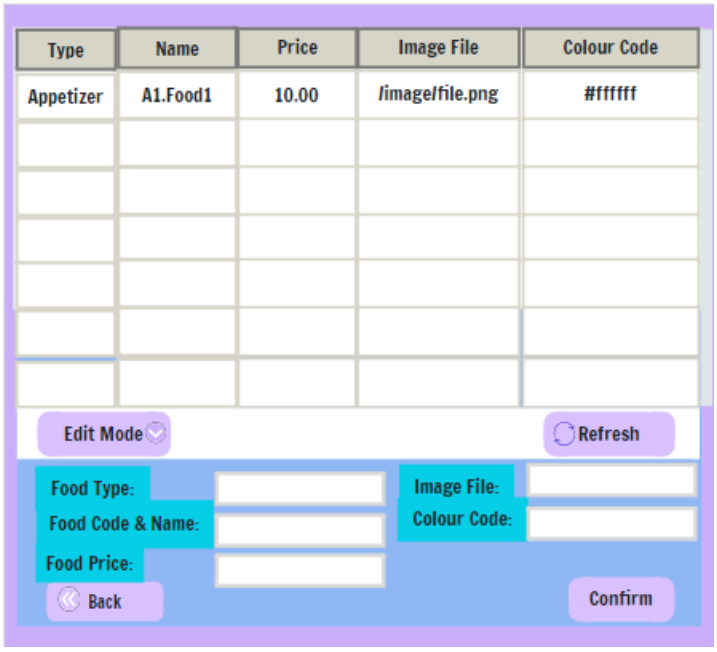


Figure 2.11: Inventory Page designed on Storyboard

Figure 2.12: inventory.fxml built on Scene Builder

Table view: holds columns where each column will be loaded with respective data in menu, can be selected and data on selected row can be passed to fields below

Edit mode menu button: show drop down menu of possible edit modes, fields will be enabled or disabled accordingly

Refresh button: Refresh item shown on table

Text fields: Allow user to enter new item or edit existing item, or only display details of item if remove

Back button: navigates user to manager page

Confirm button: pop up window created to show new warning stage, if confirm, modification will be done to the menu based on edit mode selected on menu button

- **Appetizer Menu:** Display all appetizers on menu including the name, price and image of food and allow user to place order

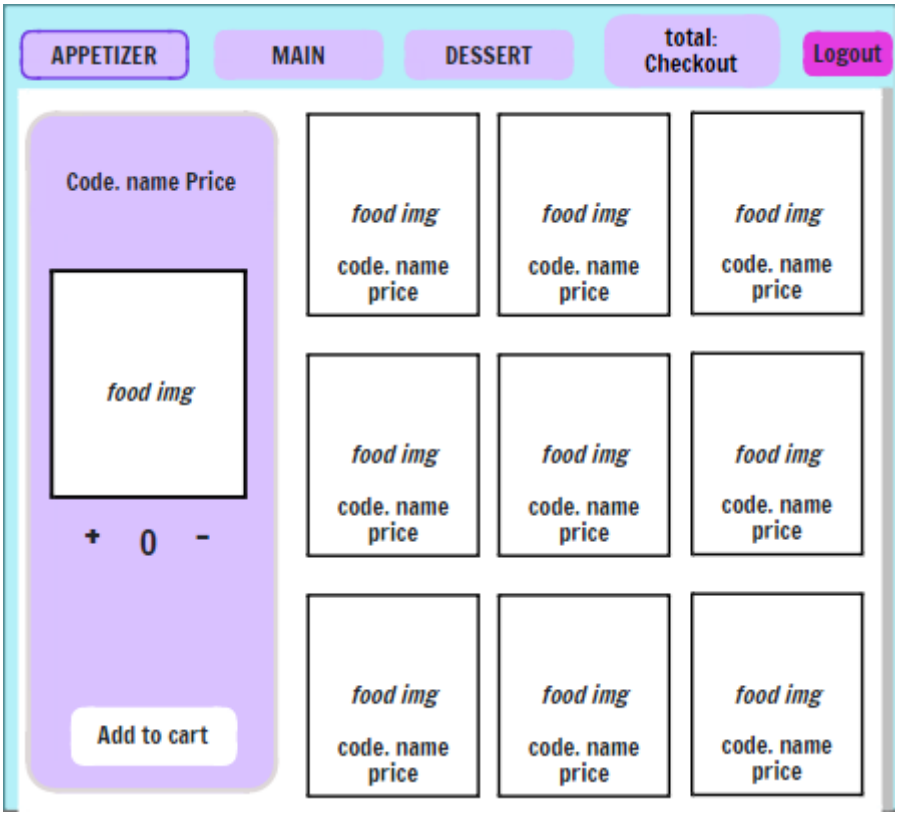


Figure 2.13: Appetizer menu designed on Storyboard

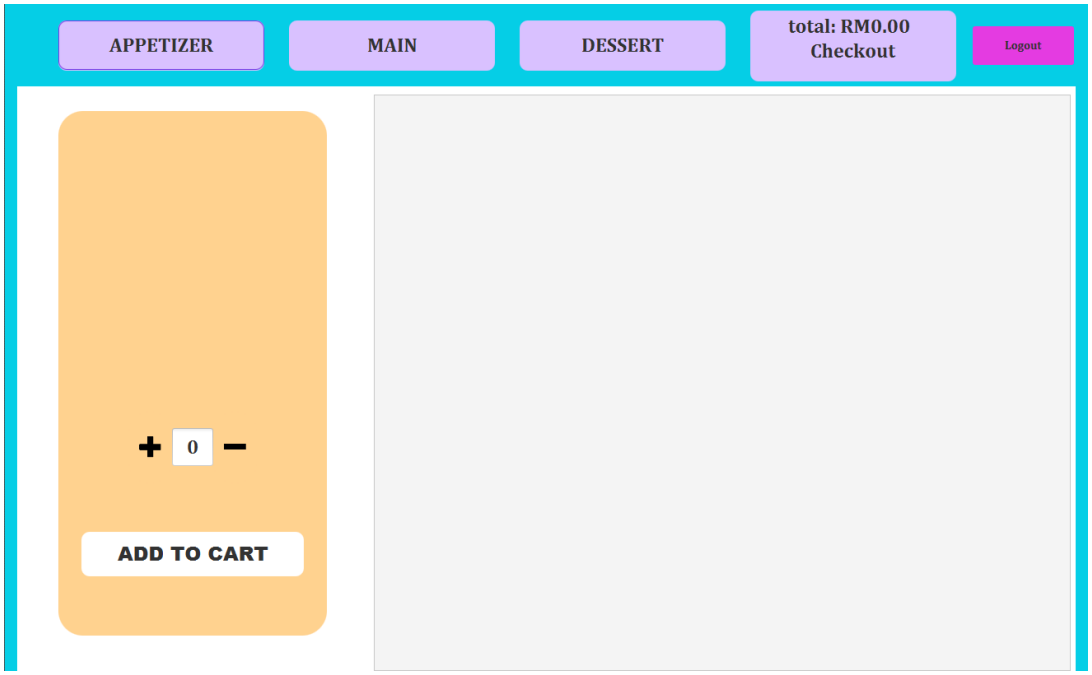


Figure 2.14: appetizermenu.fxml built on Scene Builder

Appetizer Button: no event on action, border highlighted

Main button: navigates user to scene displaying main dishes on menu

Dessert button: navigates user to scene displaying dessert on menu

Total & Checkout button: display raw total price each time item is added to cart, check if cart is empty, navigates user to checkout

Logout button: signs user out of account and navigates to start-up page

Add button: increment quantity of food to order

Minus button: decrement quantity of food to order, until 0

Quantity textfield: quantity can be set using buttons or manually entered

Add to cart button: add item selected and quantity set to cart, updates total

VBox on the left: display chosen food in a larger view

ScrollPane on right: all appetizers on menu loaded as smaller view onto grid pane held in scroll pane

- **Main Menu:** Display all main dishes on menu including the name, price and image of food and allow user to place order

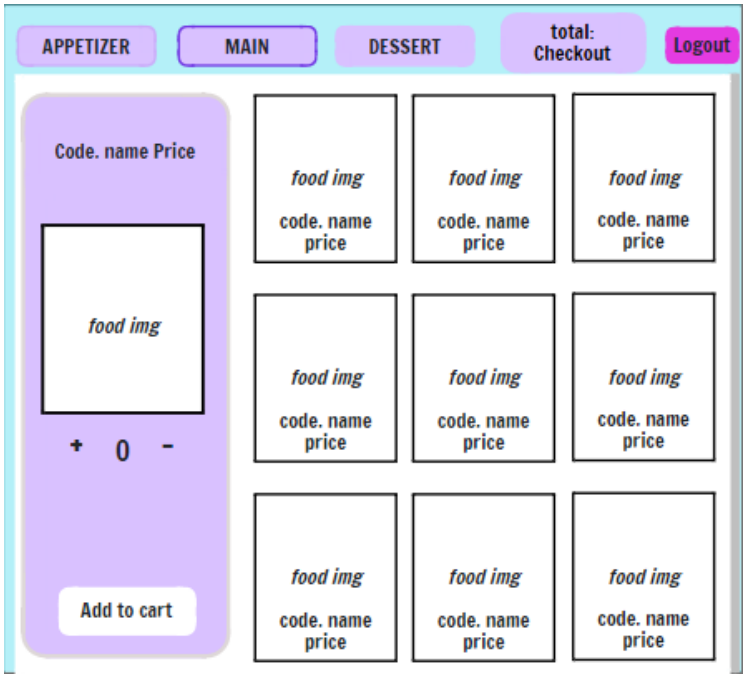


Figure 2.15: Main menu designed on Storyboard

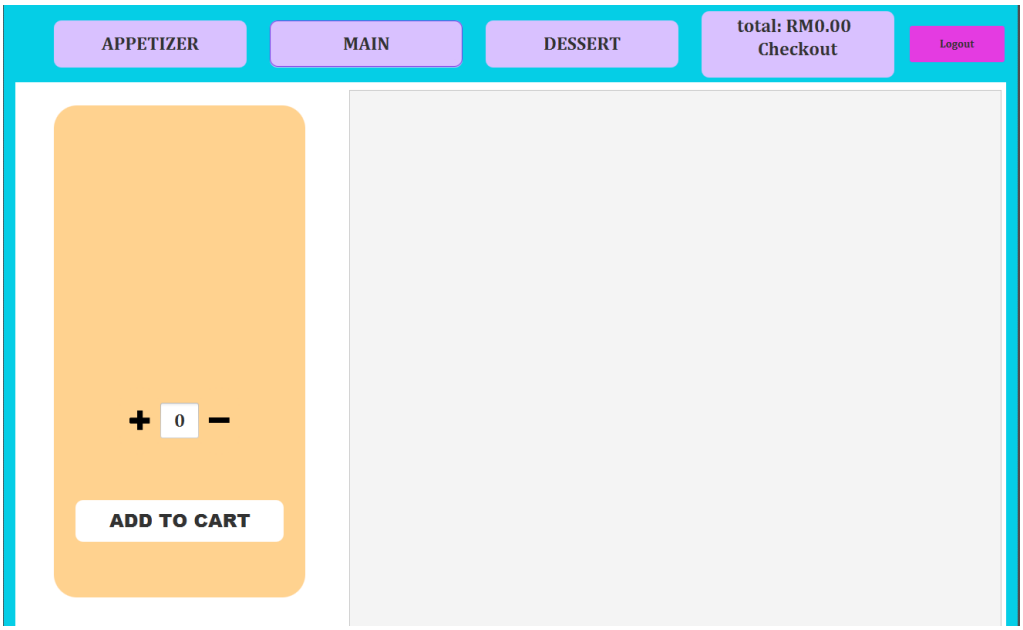


Figure 2.16: mainmenu.fxml built on Scene Builder

Similar to appetizer menu except:

Appetizer button: navigates user to scene displaying appetizer on menu

Main Button: no event on action, border highlighted

ScrollPane on right: all main dishes on menu loaded as smaller view onto grid pane held in scroll pane

- **Dessert Menu:** Display all dessert on menu including the name, price and image of food and allow user to place order

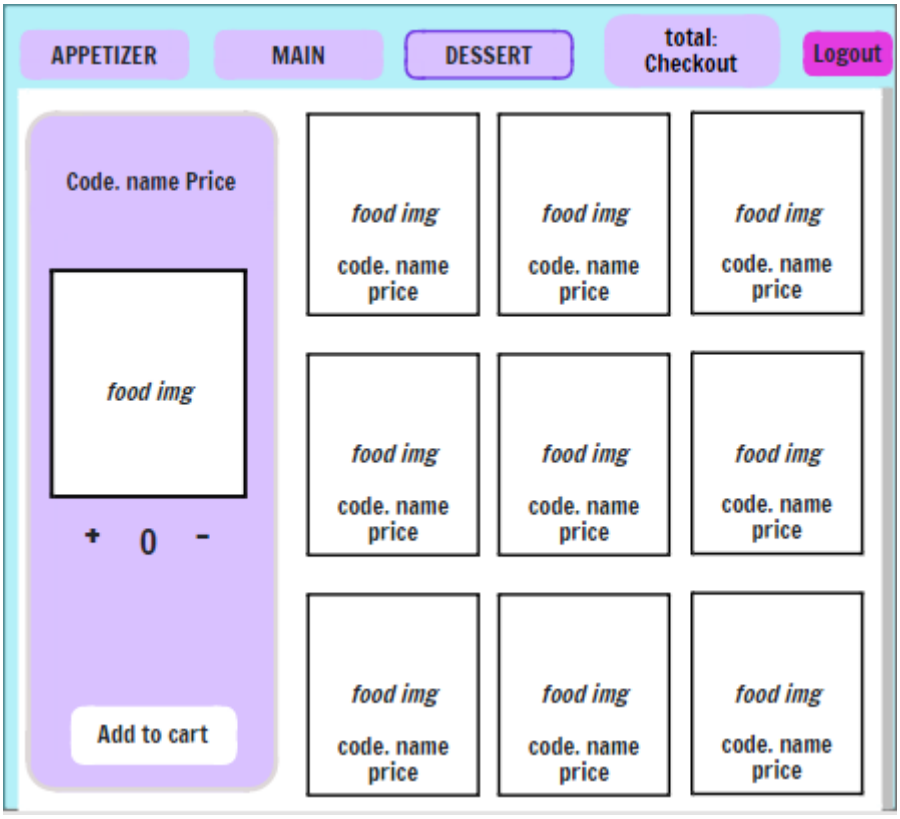


Figure 2.17: Dessert menu designed on Storyboard

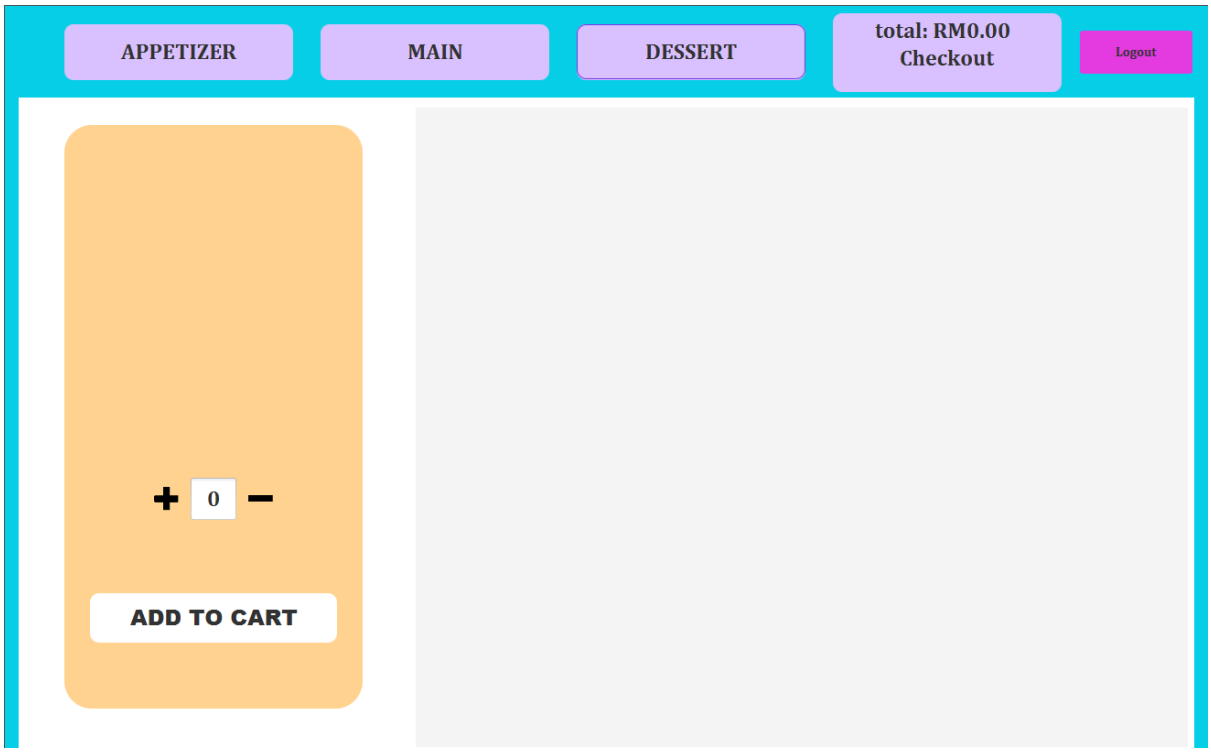


Figure 2.18: dessertmenu.fxml built on Scene Builder

Similar to appetizer menu except:

Appetizer button: navigates user to scene displaying appetizer on menu

Dessert Button: no event on action, border highlighted

ScrollPane on right: all dessert on menu loaded as smaller view onto grid pane held in scroll pane

- **Checkout Page:** display customer's order and set other order details before payment

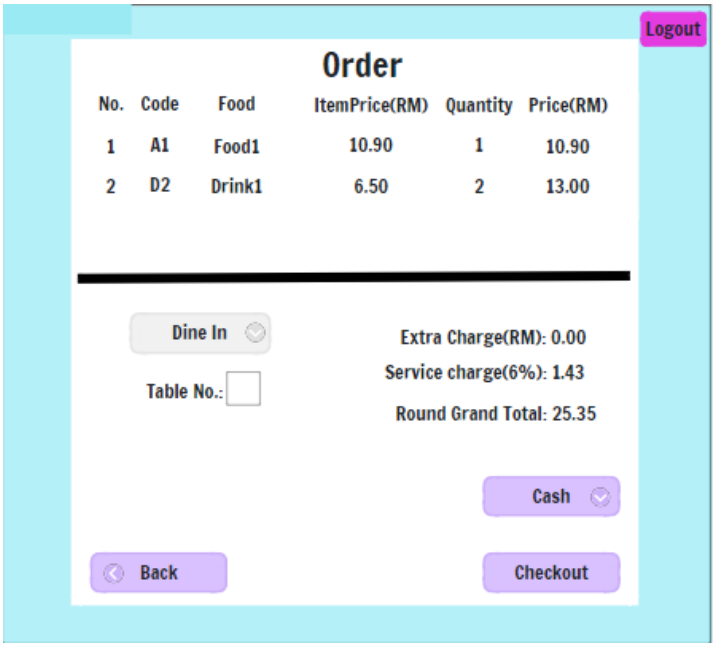


Figure 2.19: Checkout page designed on Storyboard

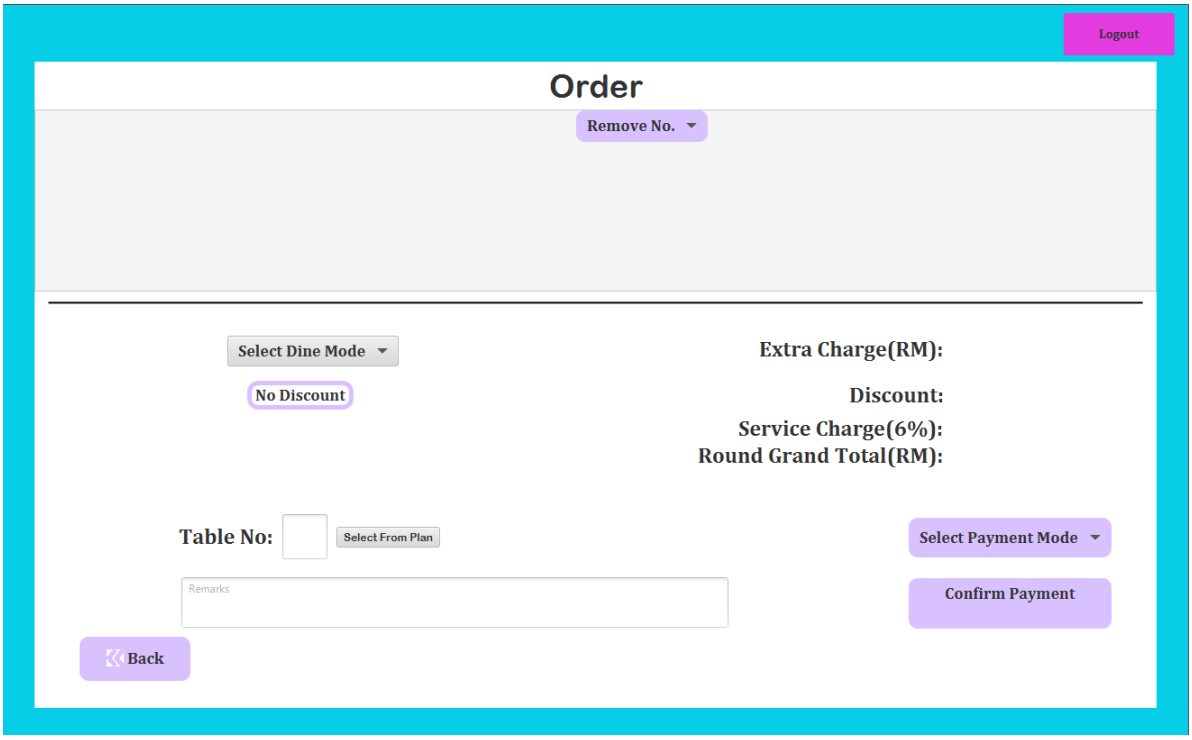


Figure 2.20: checkout.fxml built on Scene Builder

Logout button: abandons order, signs user out of account and navigates to start-up page

Scroll pane on upper half: Display all details of each item in cart

Remove menu button: drop down menu button which displays the item number of the order and number selected will be removed from order

Dine mode menu button: drop down menu button which displays the possible dine mode to be selected and help determines extra charge

Discount label: text automatically set if discount requirement is met, help determines discount amount

Table number textfield: enter table number if dine in or number will be passed from selected table

Select table number button: new window shown where scene displays the table plan of the restaurant and user click on table to select

Remarks textarea: takes any additional remarks from customer

Extra charge label: displays extra charges if any

Discount label: displays discount on price including extra charge if any

Service charge label: displays additional 6% on price after discount

Round grand total label: displays rounded total amount to be paid including all extra charges and service charge on discounted price

Payment mode menu button: drop down menu button which displays the possible payment mode to be selected and help determines the page to be brought to for payment

Back button: navigates user to appetizer menu to add-on to their order

Checkout button: checks if all order and payment details have been specified, pop up window created to show payment page according to payment method, navigates to end page after payment is done

- **Cash payment:** show payment details by cash including amount paid and change, pop up stage on top of checkout page

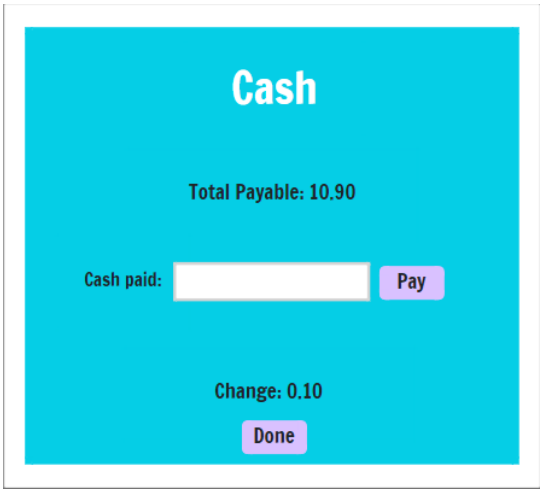


Figure 2.21: Cash payment page designed on Storyboard

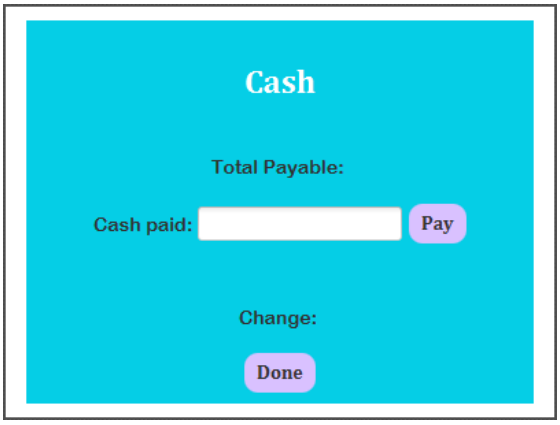


Figure 2.22: cash.fxml built on Scene Builder

Total payable label: display total to be paid when initialized

Cash paid textfield: input value of cash paid

Pay button: checks if payment is sufficient, pop up window created to show new warning stage, if confirm, done button will be set enabled

Done button: close current stage

Connect to bank page: prompt user to insert card or scan and establish connection with bank (not implemented)

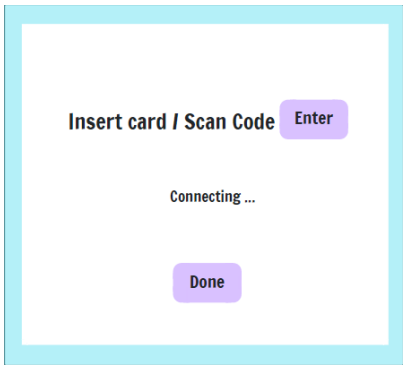


Figure 2.23: Connect page designed on Storyboard

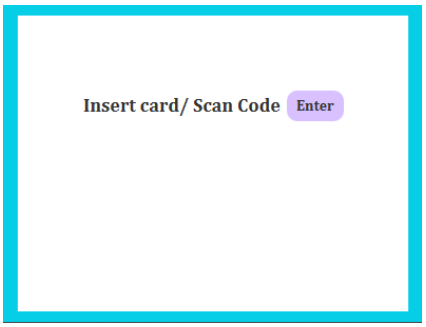


Figure 2.24: connectbank.fxml built on SceneBuilder

Enter button: pop up window created to show new warning stage, if confirm, connectiing animation and text will become visible and play, and finally done button will be set enabled

Done button: close current stage

- **End page:** Last scene of the order process

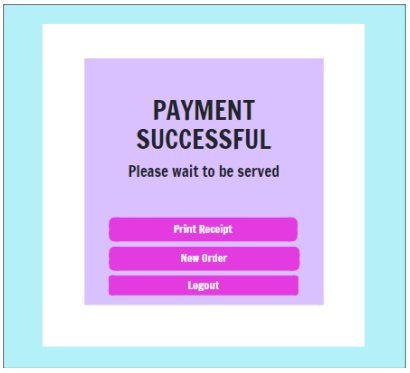


Figure 2.25: End page designed on Storyboard

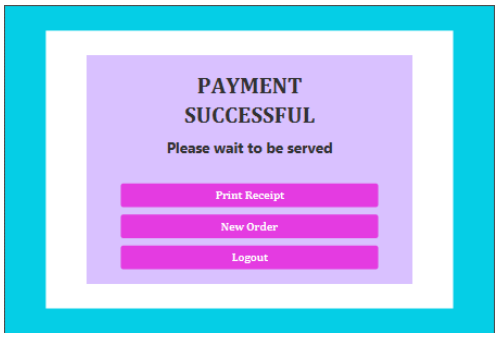


Figure 2.26: endpage.fxml built on Scene Builder

Print receipt button: output receipt on console

New order button: navigates to appetizer menu

Logout button: signs user out of account and navigates to start-up page

- **Table plan:** To select seating table if dine in

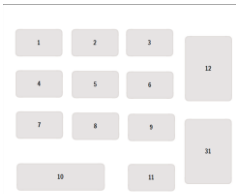


Figure 2.27: Table plan designed on Storyboard

Figure 2.28: tableplan.fxml built on Scene Builder

Button: get table number when button clicked, close stage, navigates back to checkout page

Coding the System in Java

Types of file

The types of files that make up this system are:

Java Class	Runnable Java Class	Java Interface	Java FXML	CSS file	MySQL	BIN file	Portable Network Graphics

Java Class

Held in package RestaurantSystem

(*Ref to comment right next to or on top of each line of code for exact comment of what each line or each part of code does)

Object Constructing Class

Blueprint of each type of object involved in the system

- **Admin class:** simple class with only constructor, getters and setters, representing each registered user, each admin object is inserted into ObservableList to be listed on TableView

```java
public class Admin {
    String username,adminName, manager, signDate;

    public Admin(String username, String adminName, String manager, String signDate) {
        this.username = username;
        this.adminName = adminName;
        this.manager = manager;
        this.signDate = signDate;
    }

    public String getUsername() {
        return username;
    }

    public String getAdminName() {
        return adminName;
    }

    public String getManager() {
        return manager;
    }

    public String getSignDate() {
        return signDate;
    }
}
```

Figure 3.1.1: Admin class

- **Cart class:** simple class with constructor, getters and setters, representing each item in order including the food type and quantity

```java
package RestaurantSystem;

public class Cart {
    private  Food food;
    private int quantity;

    //holds the food and quantity of each food added to cart
    public Cart(Food food, int quantity) {
        this.food = food;
        this.quantity = quantity;
    }

    public Food getFood() { return food; }

    public void setFood(Food food) {this.food = food;}

    public int getQuantity() {return quantity;}

    public void setQuantity(int quantity) { this.quantity = quantity; }

}
```

Figure 3.1.2: Cart class

- **Food class:** simple class with getters and setters, representing each food in the menu including the details when displayed on scene, implements Serializable so Food object can be serialized

```
package RestaurantSystem;

import java.io.Serializable;
import java.math.BigDecimal;

//class for food
public class Food implements Serializable {
    private String name;  //name with code of food
    private String type;  //type of food
    private BigDecimal price;  //single price of food, BigDecimal to keep ending 0 after decimal point
    private String img;  //string to image file
    private String color;  //string to colour for styling background

    //getters and setters
    public String getType() { return type; }

    public void setType(String type) { this.type = type; }

    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public BigDecimal getPrice() { return price; }

    public void setPrice(BigDecimal price) { this.price = price; }

    public String getImg() { return img; }

    public void setImg(String img) { this.img = img; }

    public String getColor() { return color; }

    public void setColor(String color) { this.color = color; }
}
```

Figure 3.1.3: Food class

- **Menu class:** simple class instantiating Food object to set up menu on system

```
package RestaurantSystem;

import java.io.*;
import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;

//class to initialise sample data for menu
public class Menu {
    private static BigDecimal bd;
    private static ArrayList<Food>menu=new ArrayList<>();
```

Figure 3.1.4: Menu class

```
public static void addMenu() {
    Food f;
    f = new Food();
    f.setName("M1.Chicken Fried Rice");
    f.setType("Main");
    bd = new BigDecimal( val: 10.90).setScale( newScale: 2, RoundingMode.HALF_UP);
    f.setPrice(bd);
    f.setImg("/image/chickenfriedrice.png");
    f.setColor("ffd28f");
    menu.add(f);
    //code omitted
```

Figure 3.1.5: addMenu method in Menu class

addMenu(): create new object of Food type and add each Food to Arraylist menu

```
//write all food to file
public static void writeMenu(){
    addMenu();
    try {
        FileOutputStream fos=new FileOutputStream( name: "src/data/Menu.bin");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(menu);
        fos.close();
        oos.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 3.1.6: writeMenu method in Menu class

writemenu(): call addMenu() and write Arraylist menu onto BIN file as stream

- **Order class:** class with getter and setter to keep order including Arraylist of Cart object and total cost

```
package RestaurantSystem;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.ArrayList;

//class to hold all orders in cart
public class Order {
    private BigDecimal bd = new BigDecimal( val: 0.00).setScale( newScale: 2, RoundingMode.HALF_UP);
    private BigDecimal total=bd;
    private ArrayList<Cart> cartArr=new ArrayList<>();

    public BigDecimal getTotal() { return total; }

    public void setTotal(BigDecimal total) { this.total = total; }

    public ArrayList<Cart> getCart() { return cartArr; }
```

Figure 3.1.7: Order class

```
//add food selected and the quantity to cart
public void addCart(Cart c){
    Boolean repeated=false;
    for(Cart cart: cartArr){
        if(cart.getFood().getName().equals(c.getFood().getName())){
            cart.setQuantity(cart.getQuantity()+c.getQuantity());  //increment quantity if food is already in cart
            repeated=true;
        }
    }
    if(!repeated){
        cartArr.add(c);
    }
}
```

Figure 3.1.8: addCart method in Order class

addCart(): add new Cart object to Arraylist cartArr when each item with specified quantity is added to order

- **Table class:** simple class with getter and setter, representing the table each order is to be served to

```
package RestaurantSystem;

//class for table data
public class Table {
    private String tableNo;

    public String getTableNo() { return tableNo; }

    public void setTableNo(String tableNo) { this.tableNo = tableNo; }
}
```

Figure 3.1.9 Table class

Controller Class

Event handlers and other logic controllers for each scene

AppetizerController: controller for appetizermenu.fxml, implements Initializable

```
public class AppetizerController implements Initializable{
    //fxml id for all referenced components on appetizer menu page
    @FXML
    private GridPane grid;
    @FXML
    private AnchorPane appetizerAnchor;
    @FXML
    private StackPane menuStackPane;
    @FXML
    private ScrollPane scroll;
    @FXML
    private VBox chosenFood;
    @FXML
    private ImageView foodImage;
    @FXML
    private Label foodNameLabel,foodPriceLabel,checkoutErrorLabel,totalLabel;
    @FXML
    private Button logoutButton,addButton,minusButton,addToCartButton,mainButton,checkoutButton,dessertButton;
    @FXML
    private TextField qtyLabel;

    private ArrayList<Food> appetizer = new ArrayList<~>();
    private Image img;
    private MyListener myListener;  //interface
    private static BigDecimal bd;
    public static Order order=new Order();  //holds information of current order
```

Figure 3.2.1: AppetizerController

```
//method to load data from file into arraylist
private ArrayList <Food> getData(){
    ArrayList<Food> appetizer=new ArrayList<>();
    try {
        FileInputStream fis=new FileInputStream( name: "src/data/Menu.bin");
        ObjectInputStream ois=new ObjectInputStream(fis);
        appetizer=(ArrayList<Food>) ois.readObject();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (EOFException e){

    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    return appetizer;
}
```

Figure 3.2.2: getData function in AppetizerController

getData(): read from bin file and pass deserialized Object as Arraylist of Food and return the Arraylist

```
//initialize scene to load each food onto scene
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    scroll.setFocusTraversable(true);
    totalLabel.setText(String.valueOf(order.getTotal()));
    appetizer.addAll(getData());
    for(Food f:appetizer) {
        if (f.getType().equals("Appetizer")) {
            setChosenFood(f);
            myListener = new MyListener() {
                @Override
                public void onClickListener(Food food) { setChosenFood(food); }
            };
            break;
        }
    }
    int row=1;
    int column =0;
    //load menu scene into each grid on grid pane
    try {
        for (int i = 0; i < appetizer.size(); i++) {
            if(appetizer.get(i).getType().equals("Appetizer")) {
                FXMLLoader loader = new FXMLLoader();
                loader.setLocation(getClass().getResource( name: "item.fxml"));
                AnchorPane anchorPane = loader.load();
                FoodController foodController = loader.getController();
                foodController.setData(appetizer.get(i), myListener);
                if (column == 3) {
                    column = 0;
                    row++;
                }

                //layout of grid pane
                grid.setHgap(50);
                grid.setVgap(30);
                grid.add(anchorPane, column++, row,  i2: 1,  i3: 1);
                grid.setMinSize(Region.USE_COMPUTED_SIZE, Region.USE_COMPUTED_SIZE);
                grid.setPrefSize(Region.USE_COMPUTED_SIZE, Region.USE_COMPUTED_SIZE);
                grid.setMaxSize(Region.USE_PREF_SIZE, Region.USE_PREF_SIZE);
                grid.setMargin(anchorPane, new Insets( v: 10));
            }
        }
    }catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 3.2.3: Overrides initialize method in AppetizerController

initialize(): overrides method in Initializable interface, when scene is loaded, set the first Food of type Appetizer on larger display, load all Food in Arraylist of type Appetizer onto grid pane, moving to next row every three column. Also pass the last item with named Juice to CheckoutController as free drink

```
//set chosen food information on larger display
private void setChosenFood(Food food){
    foodNameLabel.setText(food.getName());
    foodPriceLabel.setText("RM "+food.getPrice());
    qtyLabel.setText("0");
    img=new Image(getClass().getResourceAsStream(food.getImg()));
    foodImage.setImage(img);
    chosenFood.setStyle("-fx-background-color:#"+food.getColor()+";\n" +
            "    -fx-background-radius:30;");
    addButton.setStyle("-fx-background-color:#"+food.getColor()+";");
    minusButton.setStyle("-fx-background-color:#"+food.getColor()+";");
}
```

Figure 3.2.4: setChosenFood method in AppetizerController

setChosenFood(): style components that are part of the larger display view using css

```
public void logoutButtonOnAction(ActionEvent e){
    order=new Order();
    CheckoutController.order=new Order();
    MainController.order=new Order();
    DessertController.order=new Order();
    EndController.admin="Guest";
    Stage stage;
    //get stage where button is clicked
    stage = (Stage) logoutButton.getScene().getWindow();
    Parent root;
    //set scene to startup scene
    try {
        root = FXMLLoader.load(getClass().getResource( name: "startup.fxml"));
        Scene scene = new Scene(root, v: 600, v1: 400);
        stage.setScene(scene);
        stage.show();
        //set stage to centre of screen
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.5: logoutButtonOnAction ActionEvent handler in AppetizerController

logoutButtonOnAction():called when logoutButton on action, abandon current order, set user as Guest, load startup scene onto stage

```
//add or subtract quantity
public void qtyButtonOnAction(ActionEvent e){
    if (e.getSource()==addButton) {
        qtyLabel.setText(String.valueOf(Integer.parseInt(qtyLabel.getText()) + 1));
    }else{
        if(Integer.parseInt(qtyLabel.getText())>0) {  //only subtract if more than 0
            qtyLabel.setText(String.valueOf(Integer.parseInt(qtyLabel.getText()) - 1));
        }
    }
}
```

Figure 3.2.6: qtyButtonOnAction ActionEvent handler in AppetizerController

qtyButtonOnAction(): called when addButton or minusButton on action, increment quantity textfield if source is addButton, decrement if source is minusButton until 0

```java
//add food and quantity into arraylist when action on button
public void addToCart(ActionEvent e) {
    int qty = Integer.parseInt(qtyLabel.getText());
    double total=Double.parseDouble(totalLabel.getText());
    if (qty != 0) {
        Food f = new Food();
        f.setName(foodNameLabel.getText());
        for (Food food : appetizer) {
            if (food.getName().equals(f.getName())) {
                f = food;
                break;
            }
        }
        order.addCart(new Cart(f, Integer.parseInt(qtyLabel.getText())));
        bd=BigDecimal.valueOf(total).add(BigDecimal.valueOf(qty).multiply(f.getPrice()));
        order.setTotal(bd);
        totalLabel.setText(String.valueOf(order.getTotal()));  //increment value of total
        qtyLabel.setText("0");  //set text field back to 0
        scroll.requestFocus();
    }
}
```

Figure 3.2.7: addToCart ActionEvent handler in AppetizerController

addToCart(): called when addToCartButton is on action, get text from quantity textfield, if quantity is not 0, pass selected food and quantity to Cart constructor, add object to cartArr of order, set total by adding new price on to previous total, set focus back to scroll pane

```java
//different components request focus if for different key pressed on different previously focused components
//for easier control on device without mouse
public void switchFocus(KeyEvent e) {
    if (e.getCode() == KeyCode.LEFT) {
        if (e.getSource() == scroll) {
            addButton.requestFocus();
        } else if (e.getSource() == qtyLabel) {
            addButton.requestFocus();
        }
    }else if(e.getCode()==KeyCode.ENTER){
        if(e.getSource()==qtyLabel) {
            addToCartButton.requestFocus();
        }else if(e.getSource()==scroll){
            AnchorPane pane= (AnchorPane) grid.getChildren().get(0);
            pane.getChildren().get(0).requestFocus();
        }
    }else if(e.getCode()==KeyCode.RIGHT){
        if(e.getSource()==qtyLabel){
            minusButton.requestFocus();
        }
    }

}
```

Figure 3.2.8: switchFocus KeyEvent handler in AppetizerController

switchFocus(): to focus on different components/pane when different key pressed, for easier control on device without mouse

```
public void switchMenu(ActionEvent e){
    Scene scene;
    //get stage where button is clicked
    scene = mainButton.getScene();
    Parent root;
    String file="";
    //load different fxml file according to button clicked
    if(e.getSource()==mainButton){
        file="mainmenu.fxml";
        MainController.order=order;
    }
    else if(e.getSource()==dessertButton){
        file="dessertmenu.fxml";
        DessertController.order=order;
    }else if(e.getSource()==checkoutButton){
        if(order.getCart().size()==0){
            //error animation if cart is empty
            checkoutErrorLabel.setText("No food in cart!");
            //translate slightly and return to position in 0.1s for 5 times, then hide message
            TranslateTransition move=new TranslateTransition();
            move.setByX(2);
            move.setDuration(Duration.millis(100));
            move.setCycleCount(5);
            move.setAutoReverse(true);
            move.setNode(checkoutErrorLabel);
            move.setOnFinished(event-> {
                checkoutErrorLabel.setText("");
            });
            move.play();
            return;
        //only allow checkout if there is food ordered
        }else {
            file = "checkout.fxml";
            CheckoutController.order = order;
        }
    }
    //set scene to selected scene
    try {
        root = FXMLLoader.load(getClass().getResource(file));
        root.translateXProperty().set(scene.getWidth());
        menuStackPane.getChildren().add(root);
        //set timeline for scene sliding animation
        Timeline timeline =new Timeline();
        //scene ease in from right to left in 0.5 seconds
        KeyValue kv= new KeyValue(root.translateXProperty(), t: 0, Interpolator.EASE_IN);
        KeyFrame kf=new KeyFrame(Duration.seconds(0.5),kv);
        timeline.getKeyFrames().add(kf);
        timeline.setOnFinished(event ->{  //remove previous pane from parent after animation
            menuStackPane.getChildren().remove(appetizerAnchor);
        });
        timeline.play();
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.9: switchMenu ActionEvent handler in AppetizerController

switchMenu(): called when either mainButton/dessertButton/checkoutButton on action, get current scene, set file name to load and assign order in controller of the scene to load to reference current order according to button clicked, if source is checkoutButton, play error message animation and

return when order.cartArr is empty. Load the root onto StackPane on current scene using a sliding in animation, remove previous scene after animation

- **MainController:** controller for mainmenu.fxml, implements Initializable, similar to AppetizerController but only loads Food of type Main in initialize method without passing Juice to CheckoutController, and slight difference in direction of animation in switchMenu method
- **DessertController:** controller for dessertmenu.fxml, implements Initializable, similar to AppetizerController but only loads Food of type Dessert in initialize method without passing Juice to CheckoutController, and slight difference in direction of animation in switchMenu method
- **CashController:** controller for cash.fxml, implements initializable

```
public class CashController implements Initializable {
    //fxml id for all referenced components on cash page
    @FXML
    private Label change,total,errorLabel;
    @FXML
    private TextField paid;
    @FXML
    private Button payButton,doneButton;

    public static BigDecimal totalPayable;  //references to value of round grand total label on checkout.fxml, set by Checkout.fxml
    public static boolean confirmation;  //references to value of button selected on paymentconfirmation.fxml, set by PaymentConfirmationController.fxml
```

Figure 3.2.10: CashController

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    total.setText(total.getText()+totalPayable);  //initialize scene to show the total to be paid
    doneButton.setDisable(true);
}
```

Figure 3.2.11: initialize method in CashController

initialize(): overrides initialize method in interface Initializable, set total payable on total label and disable doneButton

```
public void pay(ActionEvent e){
    change.setText(change.getText().substring(0,change.getText().indexOf(" ")+1));  //take substring up to space without including any text after space
    if(paid.getText().equals(""))return;
    BigDecimal amtPaid=new BigDecimal(paid.getText()).setScale( newScale: 2, RoundingMode.HALF_UP);
    if(amtPaid.compareTo(totalPayable)<0){  //check if amount paid is less than needed
        errorLabel.setText("Insufficient amount");
    }
    else {
        confirmation=false;
        Stage secondaryStage=new Stage();
        secondaryStage.initStyle(StageStyle.UNDECORATED);
        try {
            secondaryStage.setScene(new Scene(FXMLLoader.load(getClass().getResource( name: "paymentconfirmation.fxml")), v: 300, v1: 200));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        secondaryStage.showAndWait();  //show warning stage and wait until stage is closed
        CheckoutController.confirmation=confirmation;
        if(!confirmation){
            Stage stage=(Stage) payButton.getScene().getWindow();
            stage.close();
        };  //check value of confirmation assigned in warning controller
        errorLabel.setText("");
        change.setText(change.getText()+amtPaid.subtract(totalPayable).setScale( newScale: 2,RoundingMode.HALF_UP));  //calculate change
        doneButton.setDisable(false);  //enable done button
    }
}
```

Figure 3.2.12: pay ActionEvent handler in CashController

pay(): called when payButton on Action, check if any value has been inserted in paid textfield, if yes the get value and pass as Bigdecimal, else output error message and return. Show warning stage on

top and wait until confirmation or cancellation of payment where value of boolean confirmation will be set. If confirm, calculate and display change, enable doneButton

```
public void doneOnAction(ActionEvent e){
    EndController.paid=paid.getText();  //pass value to end controller
    EndController.change=change.getText().substring(change.getText().indexOf(" ")+1);
    Stage stage= (Stage) doneButton.getScene().getWindow();  //close window
    stage.close();
}
```

Figure 3.2.13: doneOnAction ActionEvent handler in CashController

doneOnAction(): called when doneButton on action, pass value of paid label and change label to EndController, close current stage

```
public void switchFocus(KeyEvent e){
    if(e.getCode()== KeyCode.ENTER){
        if(e.getSource()==paid){
            payButton.requestFocus();
        }
    }else if(e.getCode()==KeyCode.RIGHT){
        if(e.getSource()==paid){
            payButton.requestFocus();
        }else if(e.getSource()==payButton){
            doneButton.requestFocus();
        }
    }
}
```

Figure 3.2.14: switchFocus KeyEvent handler in CashController

switchFocus(): similar to switchFocus in appetizerController

CheckoutController: controller for checkout.fxml, implements initializable

```
public class CheckoutController implements Initializable {
    //fxml id for all referenced components on checkout page
    @FXML
    private AnchorPane checkoutAnchor;
    @FXML
    private ScrollPane scrollPane;
    @FXML
    private Label codeLabel,extraChargeLabel,foodLabel,noLabel,priceLabel,ipLabel,qtyLabel,serviceCharge,grandTotal,discountMenu,errorLabel,discountLabel;
    @FXML
    private MenuButton menuButton,paymentMenu,remove;
    @FXML
    private Button backButton,confirmPaymentButton,selectTableButton,logoutButton;
    @FXML
    private TextField tableField;
    @FXML
    private TextArea remarkText;

    private  BigDecimal bd = BigDecimal.valueOf(0);
    public static Order order;
    private BigDecimal bdTotal;
    public static Table table=new Table();
    public static Food freeDrink;
    public  static boolean confirmation;
    private static String []discount={" No Discount "," Set Combo 15% Off"," Free Drink "};  //can be adjusted according to current available discount
```

Figure 3.2.15: CheckoutController

```
//initialize scene by displaying all the food in cart and the details
@Override
public void initialize(URL url, ResourceBundle resourceBundle) { displayOrder(); }
```

Figure 3.2.16: initialize method in CheckoutController

Initialize(): overrides initialize method in interface Initializable, display details of order

```
public void checkSetCombo(){
    discountMenu.setText(discount[0]);
    int main=0,appetizer=0,dessert=0;
    for(Cart c:order.getCart()){
        if(c.getFood().getType().equals("Main")){
            main++;
        }else if(c.getFood().getType().equals("Appetizer")){
            appetizer++;
        }else{
            dessert++;
        }
    }
    if(appetizer>0 &&main>0&&dessert>0){
        discountMenu.setText(discount[1]);
    }else checkFreeDrink();
}
```

Figure 3.2.17: checkSetCombo method in CheckoutController

checkSetCombo(): checks for discount, checks if order contains at least one food of type appetizer and main and dessert, then set text on discount label or call other method to check discount at lower priority

```
public void checkFreeDrink(){
    if(order.getTotal().compareTo(BigDecimal.valueOf(70))>0){
        discountMenu.setText(discount[2]);
        Cart free=new Cart(freeDrink, quantity: 1);
        order.addCart(free);
    }
}
```

Figure 3.2.18: checkSetCombo method in CheckoutController

checkFreeDrink(): checks if raw total is more than 70, then set text on discount label, add freeDrink passed from AppetizerController to cart

```
//action on menu button and its menu items
public void dineMode(ActionEvent e){
    for(MenuItem mi:menuButton.getItems()){
        if(e.getSource()==mi) {
            menuButton.setText(mi.getText());  //display text of menu item selected on menu button
        }
    }
    String dineMode=menuButton.getText();
    //extra charge based on selected mode
    calcTotal(dineMode);
}
```

Figure 3.2.19: dineMode ActionEvent handler on CheckoutController

dineMode(): called when any menuItem of menuButton on action, set text of menuButton to text on source

```
public void displayOrder(){
    if(order.getCart().isEmpty()){  //return to menu if all item has been removed
        back();
    }
    bdTotal=new BigDecimal( val: 0);
    bd=new BigDecimal( val: 0);
    discountLabel.setText("");
    grandTotal.setText("");
    serviceCharge.setText("");
    checkSetCombo();  //checks for discount
    String code,name,price,qty,total;
    noLabel.setText("No.");
    codeLabel.setText("Code");
    foodLabel.setText("Food");
    ipLabel.setText("Item Price(RM)");
    qtyLabel.setText("Quantity");
    priceLabel.setText("Price(RM)");
    BigDecimal bd;
    remove.getItems().clear();  //remove all menu item in menu button
    int count=1;
    for(Cart o: order.getCart()){
        noLabel.setText(noLabel.getText()+"\n"+count);  //count of different food in list
        code=o.getFood().getName().substring(0,o.getFood().getName().indexOf("."));
        codeLabel.setText(codeLabel.getText()+"\n"+code);  //code of each food
        name=o.getFood().getName().substring(o.getFood().getName().indexOf(".")+1).trim();
        foodLabel.setText(foodLabel.getText()+"\n"+name);  //name of food
        price= String.valueOf(o.getFood().getPrice());
        ipLabel.setText(ipLabel.getText()+"\n"+price);  //single price of food
        qty= String.valueOf(o.getQuantity());
        qtyLabel.setText(qtyLabel.getText()+"\n"+qty);  //quantity of food
        bd=new BigDecimal(o.getQuantity()).multiply(o.getFood().getPrice()).setScale( newScale: 2,RoundingMode.HALF_UP);  //calculate the price of each item by their quantity selected
        total= String.valueOf(bd);
        priceLabel.setText(priceLabel.getText()+"\n"+total);
        bdTotal=new BigDecimal(String.valueOf(bd.add(bdTotal))).setScale( newScale: 2,RoundingMode.HALF_UP);  //calculate running total of price to pay as each item is added
        MenuItem menuItem=new MenuItem(String.valueOf(count));  //create new menu item for each order in cart
        menuItem.setOnAction(removeMenuButton);  //add event handler to menu item
        remove.getItems().add(menuItem);  //add menu item to menu button
        count++;
    }
    calcTotal(menuButton.getText());
}
```

Figure 3.2.20: displayOrder method in CheckouController

displayOrder(): if no item in order Arraylist, call back method to switch scene to appetizer menu. clear text in labels displaying price, clear menuItems in removeMenuButton, set variables to store price to 0, set header for each label to show details of each item in cart, check for discount, loop through cartArr of order and display details in respective label, calculating the running total when each item is loop through, create new MenuItem with text the same as order number and add to remove MenuButton, call method to calculate final price

```
public void logoutButtonOnAction(ActionEvent e){
    order=new Order();
    AppetizerController.order=new Order();
    MainController.order=new Order();
    DessertController.order=new Order();
    table=new Table();
    EndController.admin="Guest";
    Stage stage;
    //get stage where button is clicked
    stage = (Stage) logoutButton.getScene().getWindow();
    Parent root;
    //set scene to startup scene when logout
    try {
        root = FXMLLoader.load(getClass().getResource( name: "startup.fxml"));
        Scene scene = new Scene(root, v: 600, v1: 400);
        stage.setScene(scene);
        stage.show();
        //set stage to centre of screen
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.21: logoutButtonOnAction actionEvent handler in CheckouController

logoutButtonOnAction():called when logoutButton on action, abandon current order, set user as Guest, load startup scene onto stage

```
public void calcTotal(String mode){
    bdTotal=order.getTotal();
    if(mode.equals("Dine In")) {
        bd = new BigDecimal( val: 0.00).setScale( newScale: 2, RoundingMode.HALF_UP);
        extraChargeLabel.setText(String.valueOf(bd));
        tableField.setDisable(false);
        selectTableButton.setDisable(false);
    }else if(mode.equals("Take Away")){
        bd = new BigDecimal( val: 1.00).setScale( newScale: 2, RoundingMode.HALF_UP);
        extraChargeLabel.setText(String.valueOf(bd));
        tableField.setDisable(true);  //disable text field and button if take away selected
        selectTableButton.setDisable(true);
    }else return;
    if(discountMenu.getText().equals(discount[1])){
        discountLabel.setText("-"+bdTotal.multiply(BigDecimal.valueOf(0.15)).setScale( newScale: 2,RoundingMode.HALF_UP));
        bdTotal=bdTotal.multiply(BigDecimal.valueOf(0.85));
    }else if(discountMenu.getText().equals(discount[2])){
        discountLabel.setText("-"+freeDrink.getPrice());
        bdTotal=bdTotal.subtract(freeDrink.getPrice());
    }
    BigDecimal temp=bdTotal;
    bdTotal=bdTotal.add(bd);  //calculate total including service charge
    BigDecimal sCharge=bdTotal.multiply(BigDecimal.valueOf(0.06)).setScale( newScale: 2,RoundingMode.HALF_UP);  //multiply by 0.06 to get service charge
    serviceCharge.setText(String.valueOf(sCharge));
    bdTotal=new BigDecimal(String.valueOf(bdTotal.add(sCharge).setScale( newScale: 2, RoundingMode.HALF_UP)));  //add service charge to total
    String decimal=String.valueOf(bdTotal).substring(String.valueOf(bdTotal).indexOf(".")+2);  //get value of digit at second decimal place
    //round value
    if(Integer.parseInt(decimal)<=2 && Integer.parseInt(decimal)>0 ){ //round down to 0.00 if ends with 1/2
        decimal="0";
    }else if(Integer.parseInt(decimal)>=3 && Integer.parseInt(decimal)<7){  //round up to 0.05 if ends with 3/4/6
        decimal="5";
    }else{  //round up to 0.1 if ends with 7/8/9
        decimal="0";
        bdTotal=bdTotal.add(BigDecimal.valueOf(0.1));
    }
    grandTotal.setText(String.valueOf(bdTotal).substring(0,String.valueOf(bdTotal).indexOf(".")+2)+decimal);  //display final total on label
    bdTotal=temp;
}
```

Figure 3.2.22: calcTotal method in CheckoutController

calcTotal(): take text in dine mode menuButtoon as String parameter, set extraCharge label according to dine mode and add extra charge to raw total, return if dine mode not selected, set discountLabel according to discounted value shown in discountMenu label, set total to discounted price, calculate and display service charge on service charge label, add service charge to total, round total according to ending decimal, display total grandTotal label

```
EventHandler<ActionEvent> removeMenuButton=new EventHandler<ActionEvent>() {  //event handler attached to each menu item in remove menu button
    @Override
    public void handle(ActionEvent e) {
        MenuItem mi= (MenuItem) e.getSource();
        int index= Integer.parseInt(mi.getText());  //get index of element to remove
        Cart remove=order.getCart().get(index-1);  //remove order from cart
        int qty=remove.getQuantity();
        order.getCart().remove( index: index-1);
        bdTotal=bdTotal.subtract(BigDecimal.valueOf(qty).multiply(remove.getFood().getPrice()));  //decrease total
        order.setTotal(bdTotal);
        calcTotal(menuButton.getText());
        displayOrder();
    }
};
```

Figure 3.2.23: removeMenuButton ActionEvent handler in CheckoutController

removeMenuButton(): called when any MenuItem of remove MenuButton is selected, remove cart item from cartArr of order, decrease value of total, call displayOrder methodto display updated order

```
//method to display text from selected menu item on menu button
public void paymentMode(ActionEvent e) {
    for (MenuItem mi : paymentMenu.getItems()) {
        if (e.getSource() == mi) {
            paymentMenu.setText(mi.getText());
        }
    }
}
```

Figure 3.2.24: paymentMode ActionEvent handler in CheckoutController

paymentMode():called when any menuItem of paymentMenu MenuButton on action, set text of paymentMenu to text on source

```
public void backButtonOnAction(ActionEvent e){
    Scene scene;
    AppetizerController.order=order;  //pass order which may be modified to appetizer controller
    //get scene where button is clicked
    scene= backButton.getScene();
    //get root of scene which is a stack pane
    StackPane menuStackPane= (StackPane) scene.getRoot();
    Parent root;
    //set root to selected scene
    try {
        root = FXMLLoader.load(getClass().getResource( name: "appetizermenu.fxml"));
        //set direction of root animation from left to right
        root.translateXProperty().set(scene.getWidth()*-1);
        menuStackPane.getChildren().add(root);  //add children to stack pane
        Timeline timeline =new Timeline();
        KeyValue kv= new KeyValue(root.translateXProperty(), t: 0, Interpolator.EASE_IN);
        KeyFrame kf=new KeyFrame(Duration.seconds(0.5),kv);  //set keyframe to 0.5 seconds
        timeline.getKeyFrames().add(kf);  //add keyframe to timeline
        timeline.setOnFinished(event ->{
            menuStackPane.getChildren().remove(checkoutAnchor);  //remove current scene after animation
        });
        timeline.play();
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.25: backButtonOnAction ActionEvent handler & back() method in CheckoutController

backButtonOnAction(): called when backButton on action, call back method which set order in AppetizerController to current controller, load appetizermenu scene onto StackPane on current scene using a sliding in animation, remove previous scene after animation

```
//open a pop-up window to select table based on table plan
public void selectTableOnAction(ActionEvent e){
    try {
        Parent root = FXMLLoader.load(getClass().getResource( name: "tableplan.fxml"));
        Stage stage=new Stage();  //create new stage
        stage.setScene(new Scene(root, v: 600, v1: 400));
        stage.showAndWait();  //show stage and wait until stage is closed

    } catch (IOException ie) {
        ie.printStackTrace();
    }

    tableField.setText(table.getTableNo());  //set table no. selected on text field
}
```

Figure 3.2.26: selectTableOnAction ActionEvent handler in CheckoutController

selectTableOnAction(): called when selectTable button on action, run new window to show table plan scene on stage and wait, set text of tableField on current stage to table no when new stage is closed

```
public void confirm(ActionEvent e){
    //get stage where button is clicked
    confirmation=false;
    if(grandTotal.getText().isEmpty()||menuButton.getText().equals("Select Dine Mode")||paymentMenu.getText().equals("Select Payment Mode")||(menuButton.getText().equals("Dine In")
            &&tableField.getText().isEmpty())){
        errorLabel.setText("Missing Order Details");
        TranslateTransition move=new TranslateTransition();
        move.setByX(2);
        move.setDuration(Duration.millis(100));
        move.setCycleCount(8);
        move.setAutoReverse(true);
        move.setNode(errorLabel);
        move.setOnFinished(event-> {
            errorLabel.setText("");
        });
        move.play();
        return;
    }
    Stage secondaryStage=new Stage();
    String file="";
    if(paymentMenu.getText().equals("Cash")) {
        file="cash.fxml";
        CashController.totalPayable=new BigDecimal(grandTotal.getText()).setScale( newScale: 2, RoundingMode.HALF_UP);
    }
    else if(paymentMenu.getText().equals("E-Wallet")||paymentMenu.getText().equals("Credit Card")) file="connectbank.fxml";
    try {
        secondaryStage.setScene(new Scene(FXMLLoader.load(getClass().getResource(file)), v: 400, v1: 300));
        secondaryStage.initStyle(StageStyle.UNDECORATED);
        secondaryStage.showAndWait();
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    if(!confirmation)return;
    Stage stage = (Stage) logoutButton.getScene().getWindow();
    Parent root;
    EndController.order=order;
    EndController.sCharge=serviceCharge.getText();
    EndController.dineMode=menuButton.getText();
    EndController.eCharge=extraChargeLabel.getText();
    EndController.discount=discountLabel.getText();
    EndController.payMode=paymentMenu.getText();
    EndController.remark=remarkText.getText();
    //set scene to startup scene when logout
    try {
        root = FXMLLoader.load(getClass().getResource( name: "endpage.fxml"));
        Scene scene = new Scene(root, v: 600, v1: 400);
        stage.setScene(scene);
        stage.show();
        //set stage to centre of screen
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
```

Figure 3.2.27: confirm ActionEvent handler in CheckoutController

confirm(): called when confirmButton on action, checks if any field is empty or menuButton is not selected and set then output error message animation and return. Load scene based on selected payment mode, set totalPayable in CashController to value displayed on grandTotal label, load scene to new stage, show stage and wait, load endpage scene onto current stage when new stage is closed and confirmation set by CashController is true

```
//different components request focus if for different key pressed on different previously focused components
//for easier control on device without mouse
public void switchFocus(KeyEvent e){
    if(e.getCode()== KeyCode.RIGHT ||e.getSource()==selectTableButton){
        if(e.getSource()==menuButton){
            paymentMenu.requestFocus();
        }else if(e.getSource()==tableField){
            selectTableButton.requestFocus();
        }else if(e.getSource()==paymentMenu ||e.getSource()==backButton){
            confirmPaymentButton.requestFocus();
        }else if(e.getSource()==remove){
            menuButton.requestFocus();
        }
    }else if(e.getCode()==KeyCode.LEFT){
        if(e.getSource()==paymentMenu){
            selectTableButton.requestFocus();
        }else if(e.getSource()==confirmPaymentButton){
            backButton.requestFocus();
        }else if(e.getSource()==remove){
            scrollPane.requestFocus();
        }else if(e.getSource()==menuButton){
            remove.requestFocus();
        }
    }else if(e.getCode()==KeyCode.ENTER){
        if(e.getSource()==tableField){
            paymentMenu.requestFocus();
        }else if(e.getSource()==scrollPane){
            remove.requestFocus();
        }
    }

}
```

Figure 3.2.28: switchFocus Keyevent Handler on CheckoutController

switchFocus(): similar to switchFocus in appetizerController

- **ConnectBankController:** controller for cash.fxml

```
public class ConnectbankController {
    //fxml id for all referenced components on checkout page
    @FXML
    private Circle circle1, circle2,circle3,circle4;

    @FXML
    private Button doneButton;

    @FXML
    private Label connectingLabel;

    @FXML
    private HBox loadingHbox;

    public static boolean confirmation=false;
```

Figure 3.2.29: ConnectBankController

```
public void enterButtonOnAction(ActionEvent e){
    confirmation=false;
    Stage secondaryStage=new Stage();
    secondaryStage.initStyle(StageStyle.UNDECORATED);
    try {
        secondaryStage.setScene(new Scene(FXMLLoader.load(getClass().getResource( name: "paymentconfirmation.fxml")), v: 300, v1: 200));
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    secondaryStage.showAndWait();  //show warning scene on new stage and wait
    CheckoutController.confirmation=confirmation;
    if(!confirmation) {
        Stage stage=(Stage) doneButton.getScene().getWindow();
        stage.close();
    }
    connectingLabel.setVisible(true);  //set label to visible
    loadingHbox.setVisible(true);
    bounce(circle1, duration: 500);  //play animation on circles
    bounce(circle2, duration: 600);
    bounce(circle3, duration: 700);
    bounce(circle4, duration: 800);
}
```

Figure 3.2.30: enterButtonOnAction ActionEvent handler in ConnectBankController

enterButtonOnAction(): called when enterButton on action, show warning stage on top and wait until confirmation or cancellation of payment where value of boolean confirmation will be set. If confirm, set label with “connecting” text to visible and call bounce method play animation on circles

```
//create animation bounce, take in node and double value of duration
public void bounce(Node n, double duration){
    TranslateTransition move=new TranslateTransition();
    move.setByY(10);
    move.setNode(n);
    move.setAutoReverse(true);
    move.setCycleCount(8);
    move.setDuration(Duration.millis(duration));
    move.play();
    if(n==circle4) move.setOnFinished(event -> doneButton.setVisible(true));
}
```

Figure 3.2.31: bounce method in ConnectBankController

bounce(): take a node and double value as parameter, set node to animation and set value as duration of animation in millisecond, if node is the last circle, make doneButton visible if at end of animation, play animation

```
public void doneOnAction(ActionEvent e){
    Stage stage= (Stage) doneButton.getScene().getWindow();
    stage.close();  //close stage
}
```

Figure 3.2.32: doneOnAction ActionEvent handler in ConnectBankController

doneOnAction(): called when doneButton on action, close the current stage

- **DeregisterController:** controller for deregister.fxml, implements initializable

```
public class DeregisterController implements Initializable {

    @FXML
    private TableColumn<Admin, String> adminColumn,managerColumn,signUpColumn,usernameColumn;
    @FXML
    private TableView<Admin> table;
    @FXML
    private Button backButton,removeButton;

    public static boolean confirmation;
    ObservableList<Admin> list= FXCollections.observableArrayList();
```

Figure 3.2.33: DeregisterController

```
@Override
public void initialize(URL url, ResourceBundle resourceBundle) { updateTable(); }
```

Figure 3.2.34: initialize method in DeregisterController

Initialize(): overrides initialize method in interface Initializable, initialize scene by calling updateTable method to set up TableView

```
public void updateTable(){
    table.getItems().clear();  //clear all items from table view
    //define which column holds what value
    usernameColumn.setCellValueFactory(new PropertyValueFactory<>( s: "Username"));
    adminColumn.setCellValueFactory(new PropertyValueFactory<>( s: "AdminName"));
    managerColumn.setCellValueFactory(new PropertyValueFactory<>( s: "Manager"));
    signUpColumn.setCellValueFactory(new PropertyValueFactory<>( s: "SignDate"));
    DatabaseConnection connectNow = new DatabaseConnection(); //instantiate new connection
    Connection connectDb= connectNow.getConnection(); //establish connection with database connection returned

    //get data from database and load each row to observable list
    try{
        ResultSet query = connectDb.createStatement().executeQuery( sql: "SELECT * FROM admin"); //apply select query

        while(query.next()){
            list.add(new Admin(query.getString( columnLabel: "username"), query.getString( columnLabel: "adminName"),
                    query.getString( columnLabel: "manager"), query.getString( columnLabel: "signed_up_on")));
        }

    }catch(Exception e){
        e.printStackTrace();
    }
    table.setItems(list);  //add items from observable list to table view
    if(table.getItems().size()==1){  //must have at least one admin, remove button disabled if only one admin
        removeButton.setDisable(true);
    }
}
```

Figure 3.2.35: updateTable method in DeregisterController

updateTable(): clear all TableColumns in TableView, assign the variable which each column displays, create connection with database, execute query to select all data from table admin, pass data from each row to new admin constructor and add to ObservableList, set table using ObservableList, if only one item in table, disable remove button

```
public void backButtonOnAction(ActionEvent e){
    Stage stage;
    //get stage where button is clicked
    stage = (Stage) backButton.getScene().getWindow();
    Parent root;
    //set scene to manager page scene
    try {
        root = FXMLLoader.load(getClass().getResource( name: "managerpage.fxml"));
        Scene scene = new Scene(root, v: 400, v1: 300);
        stage.setScene(scene);
        stage.show();
        //set scene to centre of screen
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.36: backButtonOnAction ActionEvent handler in DeregisterController

backButtonOnAction(): called when backButton on action, get current stage and load managerpage.fxml as scene onto current stage

```
public void deregister(ActionEvent e){
    confirmation=false;
    Stage secondaryStage=new Stage();
    secondaryStage.initStyle(StageStyle.UNDECORATED);
    try {
        secondaryStage.setScene(new Scene(FXMLLoader.load(getClass().getResource( name: "warning.fxml")), v: 300, v1: 200));
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    secondaryStage.showAndWait();  //show stage with warning scene
    if(!confirmation)return;  //only proceed if confirmation is true
    DatabaseConnection connectNow = new DatabaseConnection(); //instantiate new connection
    Connection connectDb= connectNow.getConnection(); //establish connection with database connection returned
    Admin selected=table.getSelectionModel().getSelectedItem();
    PreparedStatement statement=null;
    String username=selected.getUsername();
    try{
        statement=connectDb.prepareStatement( sql: "DELETE FROM admin WHERE username= ?");  //prepare statement
        statement.setString( parameterIndex: 1,username);  //pass username as first parameter
        statement.execute();  //execute query to delete
    }catch(Exception exp){
        exp.printStackTrace();
    }

    updateTable();  //update table view
}
```

Figure 3.2.37: deregister ActionEvent handler in DeregisterController

deregister(): called when deregisterButton on action, show warning stage on top and wait until confirmation or cancellation, if confirm connect to database, get the username of selected row on TableView, execute delete statement where username is the username selected, call updateTable method to show updated TableView

```
public void switchFocus(KeyEvent e){
    if(e.getCode()== KeyCode.RIGHT){
        if(e.getSource()==table){
            removeButton.requestFocus();
        }
    }else if(e.getCode()==KeyCode.UP){
        if(e.getSource()==backButton||e.getSource()==removeButton){
            table.requestFocus();
        }
    }
}
```

Figure 3.2.38: switchFocus KeyEvent handler in DeregisterController

switchFocus(): similar to switchFocus in appetizerController

- **EndController:** controller for endpage.fxml

```
public class EndController {

    @FXML
    private Button logoutButton, newButton,printButton;

    public static Order order;

    public static  String admin="Guest";
    //string to be used for printing receipt, values assigned in CheckoutController and CashController
    public static String eCharge,sCharge,dineMode,payMode,total,tableNo,paid,change,remark,discount;
```

Figure 3.2.39: EndController

```
//text to be printed on receipt
public void printReceipt(){
    StringBuilder string = new StringBuilder();
    string.append("Order Created By: "+admin+ " "+LocalDateTime.now());string.append(String.format("%20s \n","Table:"+tableNo));
    string.append(String.format("%-5s %-30s %-15s %-10s %s %n","Code","Food Name","Item Price(RM)","Quantity","Price(RM)"));
    for(Cart c: order.getCart()){
        string.append(String.format("%-5s %-30s %-15s %-10s %s %n",c.getFood().getName().substring(0,c.getFood().getName().indexOf(".")),
                c.getFood().getName().substring(c.getFood().getName().indexOf(".")+1).trim(),c.getFood().getPrice(),c.getQuantity(),
                new BigDecimal(c.getQuantity()).multiply(c.getFood().getPrice()).setScale( newScale: 2, RoundingMode.HALF_UP)));
    }
    string.append(String.format("\n %70s \n","Extra Charge:"+eCharge+" ("+dineMode+")"));
    string.append(String.format("%70s \n","Discount:"+discount));
    string.append(String.format("%70s \n","Service Charge(6%):"+sCharge));
    string.append(String.format("%70s \n","Grand Total:"+total));
    if(payMode.equals("Cash")){
        string.append(String.format("%70s \n","Cash Paid:"+paid));
        string.append(String.format("%70s \n","Change:"+change));
    }
    string.append(String.format("%s \n","Remark:"+remark));
    System.out.println(string);
}
```

Figure 3.2.40: printReceipt method in EndController

printReceipt(): output order and payment details on console, only output cash payment details if payment mode is cash

```java
public void buttonOnAction(ActionEvent e){
    AppetizerController.order=new Order();
    CheckoutController.order=new Order();
    DessertController.order=new Order();
    MainController.order=new Order();
    Stage stage;
    //get stage where button is clicked
    stage = (Stage) logoutButton.getScene().getWindow();
    Parent root;
    String file="";
    Scene scene;
    //width and height of scene
    double height=0;
    double width=0;
    if(e.getSource()==logoutButton){
        file="startup.fxml";
        height=400;
        width=600;
        admin="Guest";
    }else if(e.getSource()==newButton ){
        file="appetizermenu.fxml";
        width=1315;
        height=810;
    }else if(e.getSource()==printButton){
        printReceipt();
        return;
    }
    //set scene to startup scene
    try {
        root = FXMLLoader.load(getClass().getResource(file));
        scene = new Scene(root,width,height);
        stage.setScene(scene);
        stage.show();
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.41: buttonOnAction ActionEvent handler in EndController

buttonOnAction(): called when printButton, newButton or logoutButton on action, new Order instantiated for order in all controllers, assign name of file to load as String and the size of scene if source is newButton or logoutButton, load scene onto current stage, if source is printButton, call printreceipt method and return

- **FoodController:** controller for item.fxml

```java
public class FoodController {
    //fxml id for food item
    @FXML
    private ImageView image;

    @FXML
    private Label nameLabel,priceLabel;

    private Food food;
    private MyListener myListener;
```

Figure 3.2.42: FoodController

```
//set up each pane for food when loaded
public void setData(Food f, MyListener myListener){
    this.food=f;
    this.myListener=myListener;
    nameLabel.setText(f.getName());
    priceLabel.setText("RM "+f.getPrice());
    Image img=new Image(getClass().getResourceAsStream(f.getImg()));
    image.setImage(img);
}
```

Figure 3.2.43: setData method in FoodController

setData(): take Food and MyListener type variables as parameter, called in MenuController when data is loaded from file, set name and price of food on labels, set image file on imageViewer

```
public void click(MouseEvent e) { myListener.onClickListener(food); }
```

Figure 3.2.44: click MouseEvent handler in FoodController

click(): called when mouse click on anchorPane or button holding all components, calls method in interface MyListener

```
public void enter(KeyEvent e) {
    if(e.getCode()== KeyCode.ENTER )
    myListener.onClickListener(food);
}
```

Figure 3.2.45: enter MouseEvent handler in FoodController

enter(): called when button is focused and enter key is pressed, similar to click method

- **InventoryController:** controller for inventory.fxml, implements initializable

```
public class InventoryController implements Initializable {
    //fxml id for all referenced components on inventory page
    @FXML
    private TableColumn<Food, String> typeColumn,nameColumn,colorColumn,imgColumn;
    @FXML
    private TextField alterName,alterPrice,altertype,alterColor,alterImg;
    @FXML
    private TableColumn<Food, BigDecimal> priceColumn;
    @FXML
    private TableView<Food> table;
    @FXML
    private MenuButton editModeMenu;
    @FXML
    private Button refreshButton,confirmButton,backButton;
    @FXML
    private Label messageLabel;

    public static boolean confirmation=false;
    private static ObservableList<Food>list= FXCollections.observableArrayList();
```

Figure 3.2.46: InventoryController

```
//initialize scene
@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    setTable();
    table.requestFocus();
}
```

Figure 3.2.47: initialize method in InventoryController

initializable(): overrides initializable method in Initializable interface, calls method to set-up table when scene is loaded, focus on TableView at start for easier control with keyboard

```
//method to read data from file
public static ArrayList<Food> getData() {
    ArrayList<Food> food=new ArrayList<>();
    try {
        FileInputStream fis = new FileInputStream( name: "src/data/Menu.bin");
        ObjectInputStream ois = new ObjectInputStream(fis);
        food = (ArrayList<Food>) ois.readObject();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (EOFException e) {

    } catch (IOException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    return food;
}
```

Figure 3.2.48: getData method in InventoryController

getData(): read from bin file and load deserialized stream onto ArrayList of Food type, returning the Arraylist

```
//setup table view based on menu
public void setTable(){
    for(Food f:getData()){
        list.add(f);
    }
    //set value of each column based on the variable
    typeColumn.setCellValueFactory(new PropertyValueFactory<Food,String>( s: "Type"));
    nameColumn.setCellValueFactory(new PropertyValueFactory<Food,String>( s: "Name"));
    priceColumn.setCellValueFactory(new PropertyValueFactory<Food,BigDecimal>( s: "Price"));
    imgColumn.setCellValueFactory(new PropertyValueFactory<Food,String>( s: "Img"));
    colorColumn.setCellValueFactory(new PropertyValueFactory<Food,String>( s: "Color"));
    table.setItems(list);
}
```

Figure 3.2.49: setTable method in InventoryController

setTable(): loop through Arraylist returned by getData method and add each element to Observable list, specify which column displays which variable, set elements in ObservableList onto TableView

```
//remove previous data in table, set table with new data
public void refresh(ActionEvent e){
    table.getItems().clear();
    setTable();
}
```

Figure 3.2.50: refresh method in InventoryController

refresh(): clear all items in TableView, call method to set TableView

```
//method to control menu button
public void editMode(ActionEvent e) {
    for (MenuItem mi : editModeMenu.getItems()) {
        if (e.getSource() == mi) {
            editModeMenu.setText(mi.getText());  //set menu button to text of menu item selected
            if(mi.getText().equals("Remove")){
                altertype.setDisable(true);  //disable all text fields
                alterImg.setDisable(true);
                alterName.setDisable(true);
                alterPrice.setDisable(true);
                alterColor.setDisable(true);
            }else {
                alterImg.setDisable(false);  //enable all text fields
                alterName.setDisable(false);
                alterPrice.setDisable(false);
                alterColor.setDisable(false);
                altertype.setDisable(false);
                if(mi.getText().equals("Add")){
                    altertype.clear();  //clear text in all text fields
                    alterImg.clear();
                    alterName.clear();
                    alterPrice.clear();
                    alterColor.clear();
                }
            }
        }
    }
}
```

Figure 3.2.51: editMode ActionEvent handler in InventoryController

editMode(): called when MenuItem on editModeMenu MenuButton on action, set text of MenuButton to text on source, if remove all textfields will be disabled, if alter all textfields will be enabled, if add all textfields will be enabled but all text will be cleared to input new data

```
//event handler when enter is pressed on table view
public void enterRow(KeyEvent e){
    if(e.getCode()== KeyCode.ENTER)
    select();
    else if(e.getCode()==KeyCode.RIGHT)
        editModeMenu.requestFocus();
}
```

Figure 3.2.52: enterRow KeyEvent handler in InventoryController

enterRow(): called when KeyPressed when TableView is focused, call method select if enter key pressed, switch focus to editModeMenu MenuButton if right key pressed

```
//event handler when mouse is clicked on table view
public void selectRow(MouseEvent e) { select(); }
```

Figure 3.2.53: selectRow MouseEvent handler in InventoryController

selectRow(): called when TableView is clicked, call method select

```
//set text fields corresponding to columns on selected row
public void select(){
    Food selectedFood=table.getSelectionModel().getSelectedItem();  //get Food object selected
    if(!editModeMenu.getText().equals("Add")) {
        altertype.setText(selectedFood.getType());
        alterName.setText(selectedFood.getName());
        alterPrice.setText(String.valueOf(selectedFood.getPrice()));
        alterImg.setText(selectedFood.getImg());
        alterColor.setText(selectedFood.getColor());
    }
}
```

Figure 3.2.54: select method in InventoryController

select(): get item selected on TableView, set textfields with respective variables of selected Food

```java
//event handler attached to confirm button
public void confirmButtonOnAction(ActionEvent e){
    if(altertype.getText().isEmpty()||alterName.getText().isEmpty()||alterPrice.getText().isEmpty()||alterImg.getText().isEmpty()||alterColor.getText().isEmpty()){
        messageLabel.setText("All fields must be completed");
        messageLabel.setTextFill(Color.RED);
        return;
    }
    confirmation=false;
    //confirmation message before making changes
    Stage secondaryStage=new Stage();
    secondaryStage.initStyle(StageStyle.UNDECORATED);
    try {
        secondaryStage.setScene(new Scene(FXMLLoader.load(getClass().getResource( name: "warning.fxml")), v: 300, v1: 200));
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    secondaryStage.showAndWait();
    if(!confirmation)return;  //only perform following actions if confirm is true
    Food selected=table.getSelectionModel().getSelectedItem();  //get object selected on table
    //response according to the mode selected
    if(editModeMenu.getText().equals("Alter")) alter(selected);
    else if(editModeMenu.getText().equals("Remove"))remove(selected);
    else if(editModeMenu.getText().equals("Add"))add(getData().size());
    else {
        messageLabel.setText("Select mode to modify menu");
        messageLabel.setTextFill(Color.RED);
    }
    //empty all text fields when done
    altertype.clear();
    alterImg.clear();
    alterName.clear();
    alterPrice.clear();
    alterColor.clear();
}
```

Figure 3.2.55: confirmButtonOnAction ActionEvent handler in InventoryController

confirmButtonOnAction(): called when confirmButton on action, checks if all fields are not empty or output error message and return, show warning stage on top and wait until confirmation or cancellation, if confirm assign reference of selected item on TableView to selected of Food type, check text on MenuButton and call appropriate method or set error message, clear all fields when done

```java
//add new food that has been altered to position where it was removed
public void alter(Food selected) { add(remove(selected)); }
```

Figure 3.2.56: alter method in InventoryController

alter(): take variable of Food type as parameter, call add method by passing the value returned from remove method

```java
//write new array to file
public void updateMenu(ArrayList<Food> menu){
    try {
        FileOutputStream fos=new FileOutputStream( name: "src/data/Menu.bin");
        ObjectOutputStream oos=new ObjectOutputStream(fos);
        oos.writeObject(menu);
        fos.close();
        oos.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Figure 3.2.57: updateMenu method in InventoryController

updateMenu(): write Arraylist of Food as stream to bin file

```java
//add food item to arraylist
public void add(int index){
    String []types={"Appetizer","Main","Dessert"};
    ArrayList<Food> menu = getData();
    if(alterName.getText().indexOf(".")==-1){  //name must be made of code and name
        messageLabel.setText("Enter code and name (eg.M1.Name)");
        messageLabel.setTextFill(Color.RED);
        return;
    }
    Boolean matchType=false;
    for(String s:types){
        if(altertype.getText().equalsIgnoreCase(s)) {  //type entered must be from the three valid types
            altertype.setText(s);
            matchType=true;
            break;
        }
    }
    try{
        if(!matchType) {
            throw new Exception();
        }
    } catch (Exception e) {
        messageLabel.setText("Invalid type!");
        messageLabel.setTextFill(Color.RED);
        return;
    }

    //code and name must not be already in arraylist
    for(Food f:menu){
        if(f.getName().substring(0,f.getName().indexOf(".")).equals(alterName.getText().substring(0,alterName.getText().indexOf(".")))){
            messageLabel.setText("Code is already in menu");
            messageLabel.setTextFill(Color.RED);
        }else if(f.getName().substring(f.getName().indexOf(".")+1).equals(alterName.getText().substring(0,alterName.getText().indexOf(".")+1).trim())){
            messageLabel.setText("Name is already in menu");
            messageLabel.setTextFill(Color.RED);
        }
    }
    Food f = new Food();
    f.setType(altertype.getText());
    f.setName(alterName.getText());
    //price must be a number
    try {
        f.setPrice(new BigDecimal(alterPrice.getText()).setScale( newScale: 2, RoundingMode.HALF_UP));
    }catch (Exception e){
        messageLabel.setText("Invalid Price");
        messageLabel.setTextFill(Color.RED);
        return;
    }
    f.setImg(alterImg.getText());
    f.setColor(alterColor.getText());
    menu.add(index, f);
    updateMenu(menu);
    messageLabel.setText("Inventory Updated!");
    messageLabel.setTextFill(Color.BLACK);
}
```

Figure 3.2.58: add method in InventoryController

add(): takes integer as parameter to indicate the index to add item into Arraylist, checks if name is in correct format, checks if type is valid, checks if code and name entered is unique, catch InputMismatchException when value in alterPrice textField is converted to BigDecimal, instantiate new Food object and call appropriate setters by passing values in the textFields, add object to Arraylist at index, call updateMenu method to write Arraylist to file, show message

```java
//remove food item and return the index where the element was removed
public int remove(Food selected){
    int index=-1;
    ArrayList<Food> menu=getData();
    for(Food f:menu){
        if(f.getName().equals(selected.getName())){
            index=menu.indexOf(f);
            menu.remove(f);
            break;
        }
    }
    updateMenu(menu);
    messageLabel.setText("Inventory Updated!");
    messageLabel.setTextFill(Color.BLACK);
    return index;
}
```

Figure 3.2.59: remove method in InventoryController

remove(): takes Food type variable selected as parameter, call method getData to get Arraylist, remove selected from Arraylist, show message and return index where element has been removed,

```java
//different components request focus if for different key pressed on different previously focused components
//for easier control on device without mouse
public void switchFocus(KeyEvent e){
    if(e.getCode()==KeyCode.RIGHT){
        if(e.getSource()==editModeMenu){
            refreshButton.requestFocus();
        }
    }else if(e.getCode()==KeyCode.LEFT){
        if(e.getSource()==refreshButton){
            editModeMenu.requestFocus();
        }
    }else if(e.getCode()==KeyCode.DOWN){
        if(e.getSource()==altertype){
            alterName.requestFocus();
        }else if(e.getSource()==alterName){
            alterPrice.requestFocus();
        }else if (e.getSource()==alterPrice||e.getSource()==refreshButton) {
            alterImg.requestFocus();
        }else if (e.getSource()==alterImg){
            alterColor.requestFocus();
        }else if(e.getSource()==alterColor){
            confirmButton.requestFocus();
        }
    }else if(e.getCode()==KeyCode.UP){
        if(e.getSource()==alterName){
            altertype.requestFocus();
        }else if (e.getSource()==alterPrice) {
            alterName.requestFocus();
        }else if (e.getSource()==alterImg){
            alterPrice.requestFocus();
        }else if(e.getSource()==alterColor){
            alterImg.requestFocus();
        }else if(e.getSource()==altertype){
            refreshButton.requestFocus();
        }else if(e.getSource()==editModeMenu){
            table.requestFocus();
        }
    }
}
```

Figure 3.2.60: switchFocus method in InventoryController

switchFocus(): similar to switchFocus in appetizerController

```
//return to manager page
public void backButtonOnAction(ActionEvent e){
    Stage stage;
    //get stage where button is clicked
    stage = (Stage) backButton.getScene().getWindow();
    Parent root;
    //set scene to manager page scene
    try {
        root = FXMLLoader.load(getClass().getResource( name: "managerpage.fxml"));
        Scene scene = new Scene(root, v: 400, v1: 300);
        stage.setScene(scene);
        stage.show();
        //set scene to centre of screen
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.61: backButtonOnAction ActionEvent handler in InventoryController

backButtonOnAction(): called when backButton on action, get current stage and load managerpage.fxml as scene onto current stage

- **LoginController:** controller for login.fxml

```
public class LoginController {
    //fxml id for login page
    @FXML
    private Button loginCancelButton,loginButton,loginHere,loginExit;
    @FXML
    private Label loginMessage;
    @FXML
    private TextField loginUsername;
    @FXML
    private PasswordField loginPassword;
```

Fgure 3.2.62: LoginController

```
//attached to word here, switch scene to menu when clicked
public void loginHereButtonOnAction(ActionEvent e){
    Stage stage = (Stage) loginCancelButton.getScene().getWindow();
    try {
        Parent root = FXMLLoader.load(getClass().getResource( name: "appetizermenu.fxml"));
        Scene scene = new Scene(root, v: 1315, v1: 810);
        stage.setScene(scene);
        stage.show();
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.63: loginHereButtonOnAction ActionEvent handler in LoginController

loginHereButtonOnAction (): called when loginHere button on action, loads appetizer menu as scene on to current stage

```java
//check content of field, attached to login button on login page called when mouse click, called when keyCode on any field is ENTER
public void loginButtonOnAction(){
    //call method if both fields are not empty
    if (!loginUsername.getText().isEmpty() && !loginPassword.getText().isEmpty()){
        String validate=validateLogin();
        Parent root;
        Scene scene;
        Stage stage = (Stage) loginUsername.getScene().getWindow();
        //manager's operation if admin is manager
        if(validate.equals("manager")){
            try {
                root = FXMLLoader.load(getClass().getResource( name: "managerpage.fxml"));
                scene = new Scene(root, v: 400, v1: 300);
                stage.setScene(scene);
                stage.show();
            } catch (IOException ie) {
                ie.printStackTrace();
            }
        }
        //only allow order if normal admin
        else if(validate.equals("normal")){
            try {
                root = FXMLLoader.load(getClass().getResource( name: "appetizermenu.fxml"));
                scene = new Scene(root, v: 1315, v1: 810);
                stage.setScene(scene);
                stage.show();
            } catch (IOException ie) {
                ie.printStackTrace();
            }
        }
        //set the stage in the middle of screen
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
        //display error message in red on label if either one field is empty
    } else{
        loginMessage.setTextFill(Paint.valueOf("RED"));
        loginMessage.setText("Username and password cannot be blank");
        loginUsername.requestFocus();
    }
}
```

Figure 3.2.64: loginButtonOnAction method in LoginController

loginButtonOnAction(): checks if all fields are not empty or output error message and return, set String to return value of validateLogin method, load scene according to return value onto current stage and show stage

```java
//keyboard event handler attached to all textfields and passwordfields
public void switchField(KeyEvent e){
    //set focused field to next field below if down key pressed
    if(e.getCode()== KeyCode.DOWN) {
        if(e.getSource()==loginUsername) loginPassword.requestFocus();
        else if(e.getSource()==loginPassword) loginButton.requestFocus();
        //set focused field to previous field above if up key pressed
    }else if(e.getCode()==KeyCode.UP){
        if(e.getSource()==loginPassword) loginUsername.requestFocus();
        else if(e.getSource()==loginExit) loginHere.requestFocus();
        //call method to register or login when enter key pressed
    }else if(e.getCode().equals(KeyCode.ENTER)){
        if(e.getSource()==loginUsername ||e.getSource()==loginPassword)loginButtonOnAction();
    }
}
```

Figure 3.2.65: switchField KeyEvent handler in LoginController

switchField(): similar to switchFocus in appetizerController

```
//attached to all exit button, close current stage when clicked
public void exitButtonOnAction(ActionEvent e){
    Button exit= (Button) e.getSource();
    Stage stage = (Stage) exit.getScene().getWindow();
    stage.close();
}
```

Figure 3.2.66: exitButtonOnAction ActionEvent handler in LoginController

exitButtonOnAction(): close current stage

```
//check data entered in database
public String validateLogin(){
    DatabaseConnection connectNow = new DatabaseConnection(); //instantiate new connection
    Connection connectDb= connectNow.getConnection(); //establish connection with database connection returned
    String verifyLogin="SELECT count(1) FROM admin WHERE username='"+ loginUsername.getText().trim() + "' AND password='"
            +loginPassword.getText().hashCode()+"'"; //password is hashed before selecting from database

    try{
        Statement statement=connectDb.createStatement();
        ResultSet query = statement.executeQuery(verifyLogin); //apply select query

        while(query.next()){
            if(query.getInt( columnIndex: 1)==1){ //return column value of first column which is count()
                loginMessage.setTextFill(Paint.valueOf("BLUE"));
                loginMessage.setText("Logging in...");
                if(validateManager())return "manager";
                else return "normal";
            }else{ //empty set if return 0
                loginMessage.setTextFill(Paint.valueOf("RED"));
                loginMessage.setText("Incorrect username or password");
            }
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return "";
}
```

Figure 3.2.67: validateLogin method in LoginController

validateLogin(): create connection to database, execute statement to select count from database where username and hashed password matches value in TextField and hashed value in passwordField, return String according to Boolean value returned by method validateManager, show error message if count is not 1

```
//check if logged-in user is a manager
public boolean validateManager(){
    getAdminName();
    DatabaseConnection connectNow = new DatabaseConnection(); //instantiate new connection
    Connection connectDb= connectNow.getConnection(); //establish connection with database connection returned
    String verifyManager="SELECT manager FROM admin WHERE username='"+ loginUsername.getText().trim()
            + "' AND password='" +loginPassword.getText().hashCode()+"'"; //password is hashed before selecting from database

    try{
        Statement statement=connectDb.createStatement();
        ResultSet query = statement.executeQuery(verifyManager); //apply select query

        while(query.next()){
            if(query.getInt( columnIndex: 1)==1) { //return tinyint of first column which is manager
                return (true);
            }
        }
    }catch(Exception e){
        e.printStackTrace();
    }
    return false;
}
```

Figure 3.2.68: validateManager method in LoginController

validateManager(): create connection to database, execute statement to select manager from admin where username matches value in TextField, return boolean according to value in column

```
//set admin name in endController
public void getAdminName(){
    DatabaseConnection connectNow = new DatabaseConnection(); //instantiate new connection
    Connection connectDb= connectNow.getConnection(); //establish connection with database connection returned
    String getAdmin="SELECT adminName FROM admin WHERE username='"+ loginUsername.getText().trim()
            + "' ";

    try{
        Statement statement=connectDb.createStatement();
        ResultSet query = statement.executeQuery(getAdmin); //apply select query

        while(query.next()){
            EndController.admin= String.valueOf(query.getString( columnIndex: 1));
        }

    }catch(Exception e){
        e.printStackTrace();
    }
}
```

Figure 3.2.69: getAdminName method in LoginController

getAdminName(): create connection to database, execute statement to select adminName from table admin where username matches value in TextField, assign variable in EndController to result in the column returned

```
//event handler when cancel button on login or register page is clicked
public void cancelButtonOnAction(ActionEvent e){
    Stage stage;
    //get stage where button is clicked
    stage = (Stage) loginCancelButton.getScene().getWindow();
    Parent root;
    //set scene to startup scene
    try {
        root = FXMLLoader.load(getClass().getResource( name: "startup.fxml"));
        Scene scene = new Scene(root, v: 600, v1: 400);
        stage.setScene(scene);
        stage.show();
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.70: cancelButtonOnAction ActionEvent handler in LoginController

cancelButtonOnAction(): called when loginCancel button on action, load startup scene onto current stage and show, set stage to centre of screen

- **ManagerController:** controller of mangerpage.fxml

```
public class ManagerController {
    @FXML
    private Button orderButton, registerButton, managerLogout, deregisterButton;
```

Figure 3.2.71: ManagerController

```
public void logoutButtonOnAction(ActionEvent e){
    EndController.admin="Guest";
    Stage stage;
    //get stage where button is clicked
    stage = (Stage) managerLogout.getScene().getWindow();
    Parent root;
    //set scene to startup scene
    try {
        root = FXMLLoader.load(getClass().getResource( name: "startup.fxml"));
        Scene scene = new Scene(root, v: 600, v1: 400);
        stage.setScene(scene);
        stage.show();
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.72: logoutButtonOnAction ActionEvent handler in ManagerController

logoutButtonOnAction(): called when logoutButton on action, set user as Guest, load startup scene onto stage, set stage to centre of screen, show stage

```
public void operationButtonOnAction(ActionEvent e){
    Stage stage= (Stage) registerButton.getScene().getWindow();
    Parent root;
    Scene scene;
    try {
        //load register scene
        if(e.getSource()==registerButton) {
            root = FXMLLoader.load(getClass().getResource( name: "register.fxml"));
            scene = new Scene(root,  v: 520,  v1: 400);
        //load appetizer menu scene
        }else if(e.getSource()==orderButton) {
            root = FXMLLoader.load(getClass().getResource( name: "appetizerMenu.fxml"));
            scene = new Scene(root,  v: 1315,  v1: 810);
        }else if(e.getSource()==deregisterButton){
            root = FXMLLoader.load(getClass().getResource( name: "deregister.fxml"));
            scene = new Scene(root,  v: 600,  v1: 300);
        //load inventory screen
        }else{
            root = FXMLLoader.load(getClass().getResource( name: "inventory.fxml"));
            scene = new Scene(root, v: 600, v1: 600);
        }
        stage.setScene(scene);
        stage.show();
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.73: operationButtonOnAction ActionEvent handler in ManagerController

operationButtonOnAction(): called when registerButton / deregisterButton/ inventoryButton/ orderButton on action, load appropriate fxml file as scene, set scene on stage, show stage on centre of screen

- **PaymentConfirmationController:** controller for paymentconfirmation.fxml

```
public class PaymentConfirmationController {
    public void confirm(ActionEvent e){
        Button button= (Button) e.getSource();
        //pass confirm or cancel to inventory controller
        if(button.getText().equals("Confirm")){
            CashController.confirmation=true;
            ConnectbankController.confirmation=true;
        }
        else {
            CashController.confirmation=false;
            ConnectbankController.confirmation=false;
        }
        Stage stage= (Stage) button.getScene().getWindow();
        stage.close();
    }
}
```

Figure 3.2.74: PaymentConfirmationController

confirm(): called when confirmButton or cancelButton on action, set Boolean variable of CashController and ConnectBankController based on event source

- **RegisterController:** controller for register.fxml

```
public class RegisterController {
    //fxml id for register page
    @FXML
    private Button registerCancelButton,registerButton,registerHere,registerExit;
    @FXML
    private Label registerMessage;
    @FXML
    private TextField registerName,registerUsername;
    @FXML
    private PasswordField registerPassword,registerConPassword;
    @FXML
    private CheckBox managerCheckBox;
```

Figure 3.2.75: RegisterController

```
//attached to word here, switch scene to login when clicked
public void registerHereButtonOnAction(ActionEvent e){
    EndController.admin="Guest";
    Stage stage = (Stage) registerCancelButton.getScene().getWindow();
    try {
        Parent root = FXMLLoader.load(getClass().getResource( name: "login.fxml"));
        Scene scene = new Scene(root, v: 520, v1: 400);
        stage.setScene(scene);
        stage.show();
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.76: registerHereButtonOnAction ActionEvent handler in RegisterController

registerHereButtonOnAction(): called when registerHere button on action, set user as Guest, loads login page as scene onto current stage

```
//check content of field, attached to register button on register page called when mouse click, called when keyCode on any field is ENTER
public void registerButtonOnAction(){
    registerMessage.setTextFill(Paint.valueOf("RED"));
    //output error message if any field is empty
    if (registerUsername.getText().isEmpty() || registerName.getText().isEmpty() ||registerPassword.getText().isEmpty()
            || registerConPassword.getText().isEmpty())registerMessage.setText("All fields must not be empty");
        //output error message if different values entered into password and confirm password
    else if(!registerPassword.getText().equals(registerConPassword.getText()))registerMessage.setText("Password does not match");
        //output error message if length of password is too short for security
    else if (registerPassword.getText().length()<6)registerMessage.setText("Password should be at least 6 characters");
    else {
        if(registerUser()){
            Stage stage= (Stage) registerButton.getScene().getWindow();
            try {
                Parent root = FXMLLoader.load(getClass().getResource( name: "managerpage.fxml"));
                Scene scene = new Scene(root, v: 400, v1: 300);
                stage.setScene(scene);
                stage.show();
            } catch (IOException ie) {
                ie.printStackTrace();
            }
        }
    }
}
```

Figure 3.2.77: registerButtonOnAction method in RegisterController

registerButtonOnAction(): checks if all fields are not empty, checks if values in both Password fields matches, checks if length of String in passwordfield is not less than 6, or output error message, if return value of method registerUser is true, load manager page onto scene, set scene on current stage and show stage

```
public boolean registerUser(){
    DatabaseConnection connectNow = new DatabaseConnection(); //instantiate new connection
    Connection connectDb= connectNow.getConnection(); //establish connection with database connection returned
    String name=registerName.getText();
    String username=registerUsername.getText();
    boolean manager=managerCheckBox.isSelected();
    int managerInt;
    if(manager)managerInt=1;
    else managerInt=0;
    String password= String.valueOf(registerPassword.getText().hashCode()); //hash password before inserting into database
    String insertFields="INSERT INTO admin(adminName, username, password, manager, signed_up_on) VALUES('";
    String insertValues=name+"','"+username+"','"+password+"',"+managerInt+", now())";
    String insertQuery=insertFields+insertValues; //join strings to form insert query

    try{
        Statement statement=connectDb.createStatement();
        statement.executeUpdate(insertQuery); //execute insert query
        registerMessage.setTextFill(Paint.valueOf("Blue"));
        registerMessage.setText("Successful registration!"); //output blue message if query ok
        return true;
    }catch(Exception e){
        registerMessage.setTextFill(Paint.valueOf("RED")); //output red error message if error where username is not unique
        registerMessage.setText("Username " +username+" has been taken used");
    }
    return false;

}
```

Figure 3.2.78: registerUser method in RegisterController

registerUser(): connect to database, get String in fields and value in checkbox to build insert into query, insert current datetime into column signd_up_on, execute statement to update table, show error message if exception thrown due to non-unique value for username and return false, if query ok return true

```java
//keyboard event handler attached to all textfields and passwordfields
public void switchField(KeyEvent e){
    //set focused field to next field below if down key pressed
    if(e.getCode()== KeyCode.DOWN) {
        if(e.getSource()==registerName) registerUsername.requestFocus();
        else if(e.getSource()==registerUsername)registerPassword.requestFocus();
        else if(e.getSource()==registerPassword)registerConPassword.requestFocus();
        else if(e.getSource()==registerConPassword)registerButton.requestFocus();
        //set focused field to previous field above if up key pressed
    }else if(e.getCode()==KeyCode.UP){
        if(e.getSource()==registerUsername) registerName.requestFocus();
        else if(e.getSource()==registerPassword) registerUsername.requestFocus();
        else if(e.getSource()==registerConPassword) registerPassword.requestFocus();
        else if(e.getSource()==registerExit)registerHere.requestFocus();
        //call method to register or login when enter key pressed
    }else if(e.getCode().equals(KeyCode.ENTER)){
        if(e.getSource()==registerName ||e.getSource()==registerPassword ||e.getSource()==registerUsername||e.getSource()==registerConPassword)registerButtonOnAction();
    }
}
```

Figure 3.2.79: switchField Keyevent handler in RegisterController

switchField(): similar to switchFocus in appetizerController

```java
//event handler when cancel button on login or register page is clicked
public void cancelButtonOnAction(ActionEvent e){
    Stage stage;
    //get stage where button is clicked
    stage = (Stage) registerCancelButton.getScene().getWindow();
    Parent root;
    //set scene to startup scene
    try {
        root = FXMLLoader.load(getClass().getResource( name: "managerpage.fxml"));
        Scene scene = new Scene(root, v: 400, v1: 300);
        stage.setScene(scene);
        stage.show();
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.80: cancelButtonOnAction ActionEvent handler in RegisterController

cancelButtonOnAction(): called when registerCancel button on action, load manager page scene onto current stage and show, set stage to centre of screen

```java
//attached to all exit button, close current stage when clicked
public void exitButtonOnAction(ActionEvent e){
    Button exit= (Button) e.getSource();
    Stage stage = (Stage) exit.getScene().getWindow();
    stage.close();
}
```

Figure 3.2.81: exitButtonOnAction ActionEvent handler in RegisterController

exitButtonOnAction(): close current stage

StartupController: controller for startup.fxml

```java
public class StartupController{
    //fxml id for startup page
    @FXML
    private Button startupLogin,startupGuest;
```

Figure 3.2.82: StartupController

```java
//event handler when button clicked on startup page, attached to login and guest buttons
public void startupOnAction(ActionEvent e) {
    Stage stage = (Stage) startupLogin.getScene().getWindow();
    Parent root;
    Scene scene = null;
    try {
        //switch to login scene if source is login button
        if(e.getSource()==startupLogin) {
            root = FXMLLoader.load(getClass().getResource( name: "login.fxml"));
        }
        //switch to menu screen
        else{
            root = FXMLLoader.load(getClass().getResource( name: "appetizerMenu.fxml"));
            scene = new Scene(root, v: 1315, v1: 810);
        }
        //set root and size of scene
        if(e.getSource()!=startupGuest) {
            scene = new Scene(root,  v: 520,  v1: 400);
        }
        stage.setScene(scene);
        stage.show();
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        stage.setX((primScreenBounds.getWidth() - stage.getWidth()) / 2);
        stage.setY((primScreenBounds.getHeight() - stage.getHeight()) / 2);
    } catch (IOException ie) {
        ie.printStackTrace();
    }
}
```

Figure 3.2.83: startupOnAction ActionEvent handler in StartupController

startupOnAction(): load fxml file to scene based on button source, set scene to stage, show stage on centre of screen

- **TablePlanController:** controller for tableplan.fxml

```java
public class TablePlanController {

    //get table number selected and pass table no. to checkout scene
    public void tableButtonOnAction(ActionEvent e){
        Button table= (Button) e.getSource();
        Table t=new Table();
        t.setTableNo(table.getText());
        CheckoutController.table=t;
        Stage stage = (Stage) table.getScene().getWindow();
        stage.close();
    }
}
```

Figure 3.2.84: TablePlanController

tableButtonOnAction(): called when any button on the scene on action, set tableNo of table in CheckoutController as value on button source, close current stage

WarningController: controller for warning.fxml, similar to PaymentConfirmationController but set Boolean value of InventoryController and DeregisterController

Runnable Java Class

- **Main:** extends application, with static main method and start method

```
public class Main extends Application {

    public static void main(String[] args) { launch(args); }

    @Override
    public void start(Stage primaryStage) throws IOException {
        //Menu.writeMenu();  //only run on first run of application
        Parent root = FXMLLoader.load(getClass().getResource( name: "startup.fxml"));
        primaryStage.setTitle("Restaurant System");
        primaryStage.initStyle(StageStyle.UNDECORATED);  //remove the window operation around stage
        primaryStage.setScene(new Scene(
                //root,1315,810));
                root, v: 600, v1: 400));
        primaryStage.show();
        //set stage on centre of screen
        Rectangle2D primScreenBounds = Screen.getPrimary().getVisualBounds();
        primaryStage.setX((primScreenBounds.getWidth() - primaryStage.getWidth()) / 2);
        primaryStage.setY((primScreenBounds.getHeight() - primaryStage.getHeight()) / 2);
    }
}
```

Figure 3.3.1: Main Class

main(): launch to create thread

start(): load startup page onto scene, set scene on stage, set title of stage, remove decorations and panel around stage, show stage, set stage to centre of screen

CSS file

- **style.css:** Used to style components and pane

```
.selected-btn{
    -fx-background-color: #d9c1ff;
    -fx-background-radius: 10;
    -fx-border-radius: 10;
    -fx-border-color: rgb(113, 57, 228);
}
```

Figure 3.4.1: style class selected-btn

Set background colour using hex and radius to 10, set border colour using rgb and radius to 10

```
.button:focused,.button:hover{
     -fx-effect: dropshadow(gaussian, #8f1fea, 10, 0, 0, 0);
}
```

Figure 3.4.1: style class button

Set drop shadow effect to button when button is hovered or focused

SQL

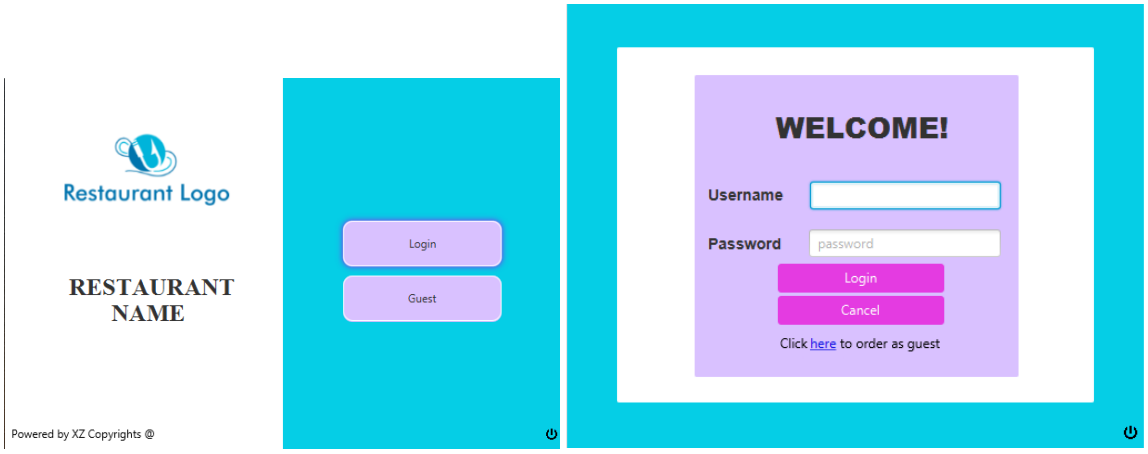
Create new database and import file at first run, connection created using DatabaseConnection Java Class

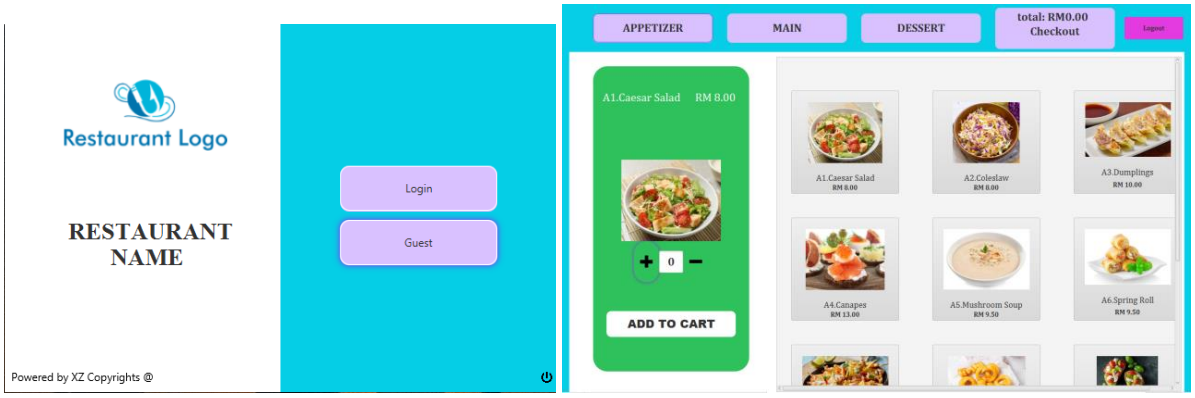
Output when Running System

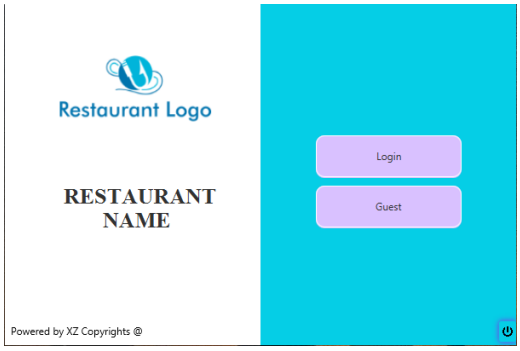
Screenshot on the left shows the source when focused

Screenshot on the right shows the output after event on source

- **startup.fxml**

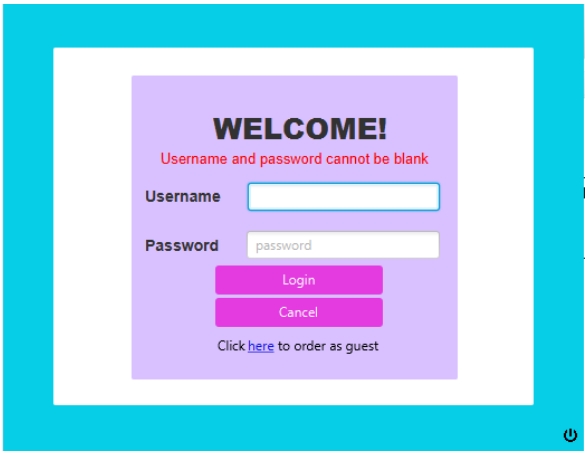


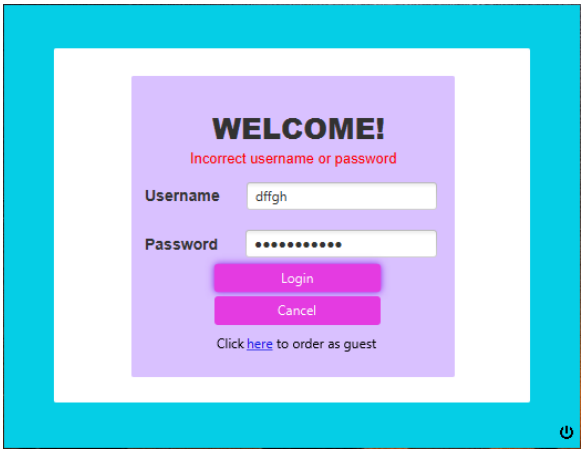


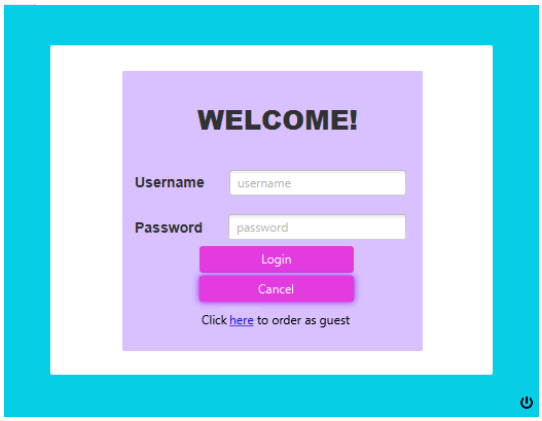


Stage Close

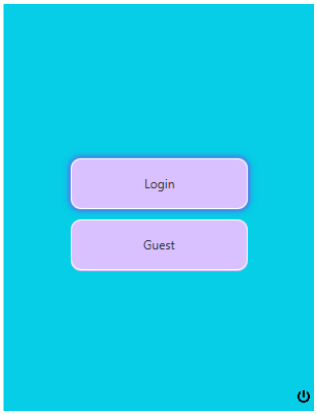
- **login.fxml**

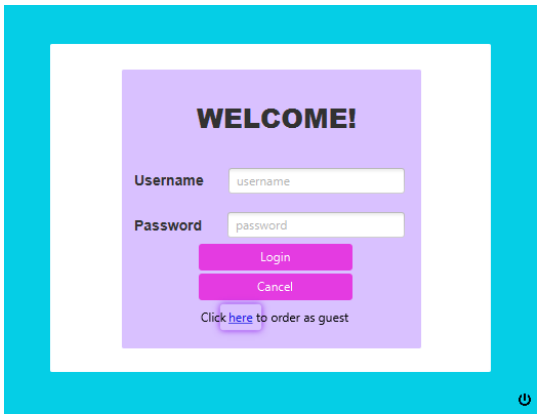

WELCOME!
Username and password cannot be blank
Username
Password
password
Login
Cancel
Click here to order as guest

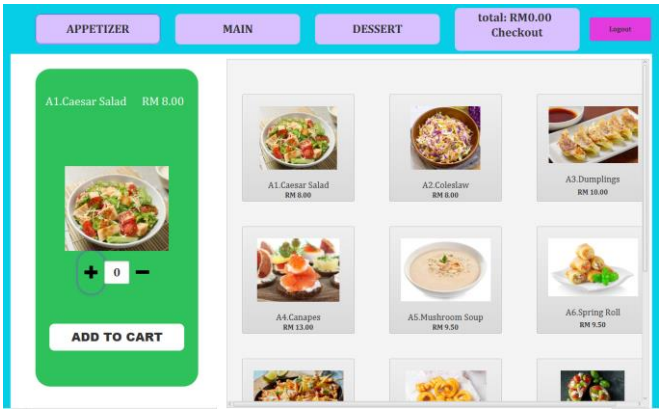

WELCOME!
Incorrect username or password
Username
dffgh
Password
Login
Cancel
Click here to order as guest

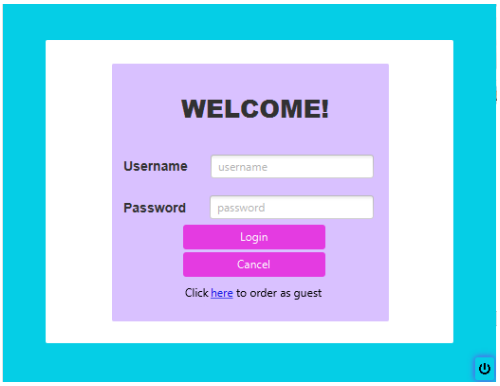

WELCOME!
Username
username
Password
password
Login
Cancel
Click here to order as guest


Restaurant Logo
RESTAURANT NAME
Powered by XZ Copyrights @

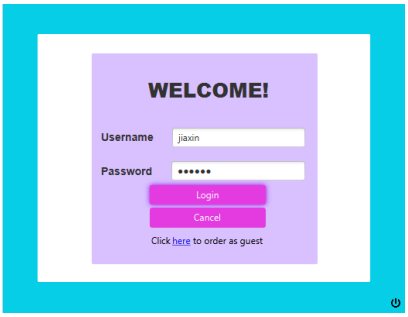

Login
Guest

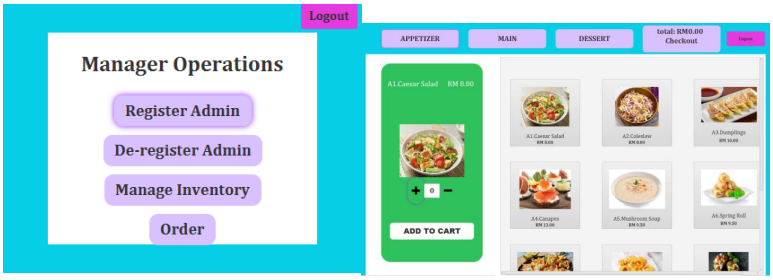

WELCOME!
Username
username
Password
password
Login
Cancel
Click here to order as guest


APPETIZER
MAIN
DESSERT
total: RM0.00
Checkout
Logout
A1.Caesar Salad RM 8.00
0
ADD TO CART
A1.Caesar Salad
RM 8.00
A2.Coleslaw
RM 8.00
A3.Dumplings
RM 10.00
A4.Canapes
RM 13.00
A5.Mushroom Soup
RM 9.50
A6.Spring Roll
RM 9.50

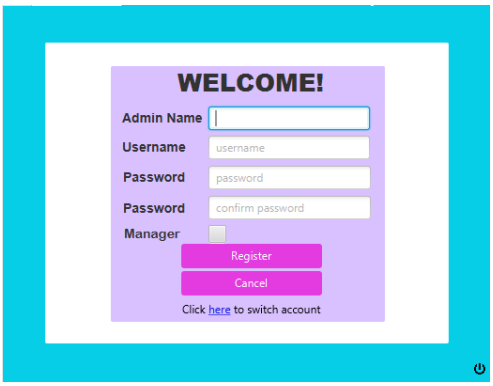


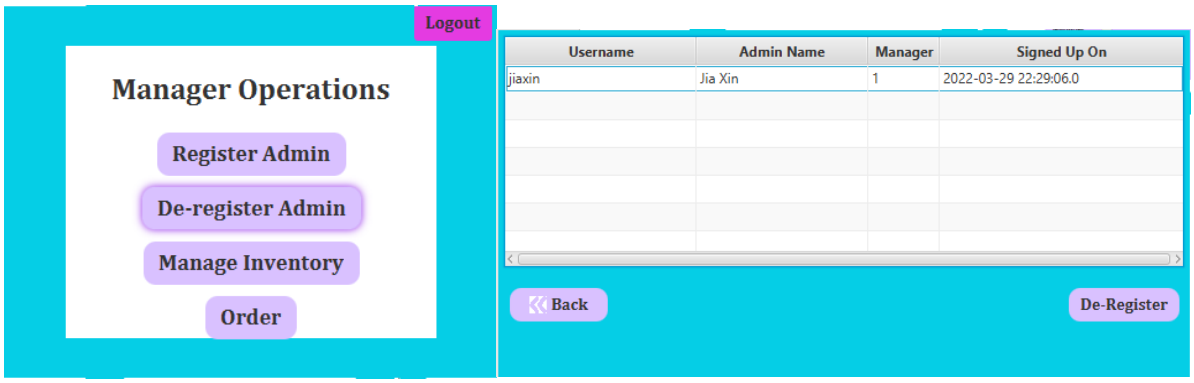
Stage Close

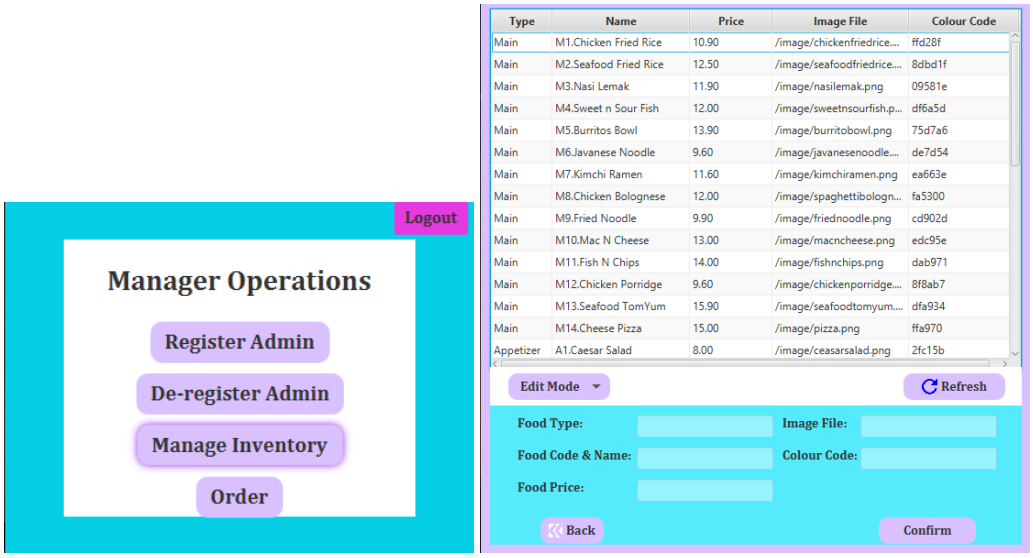


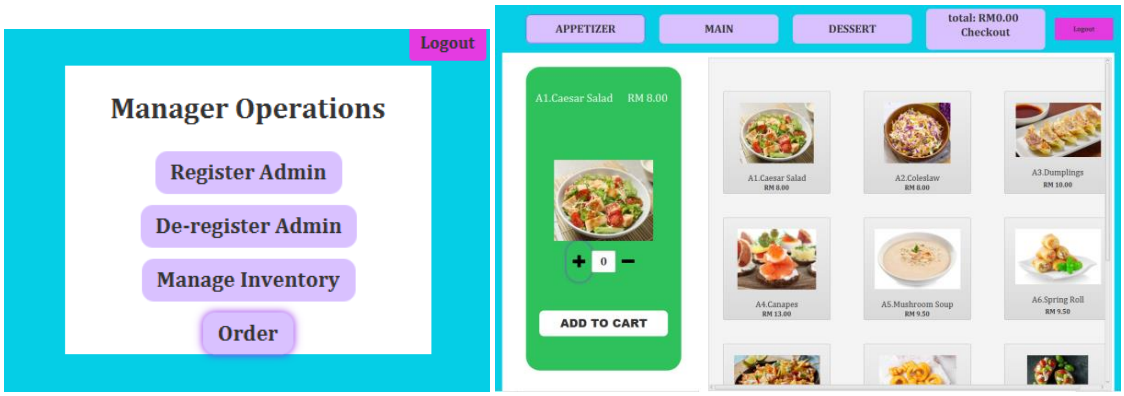


- **managerpage.fxml**

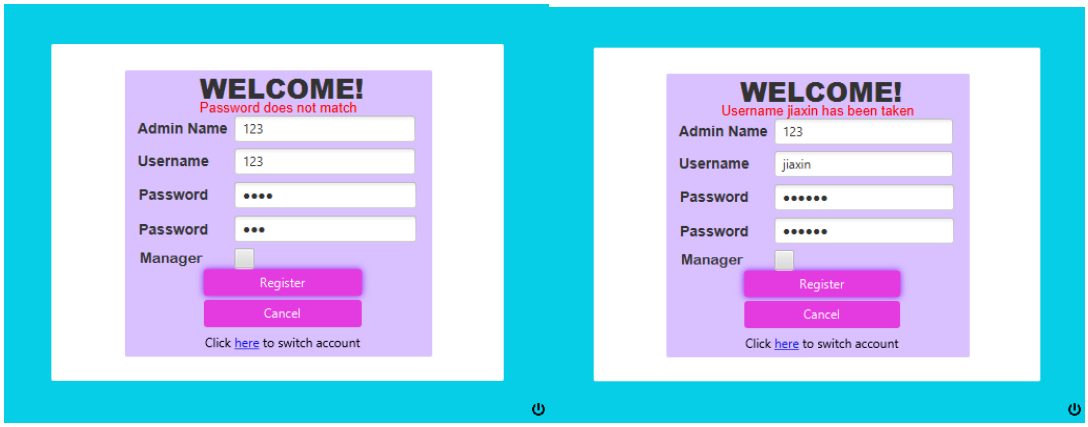


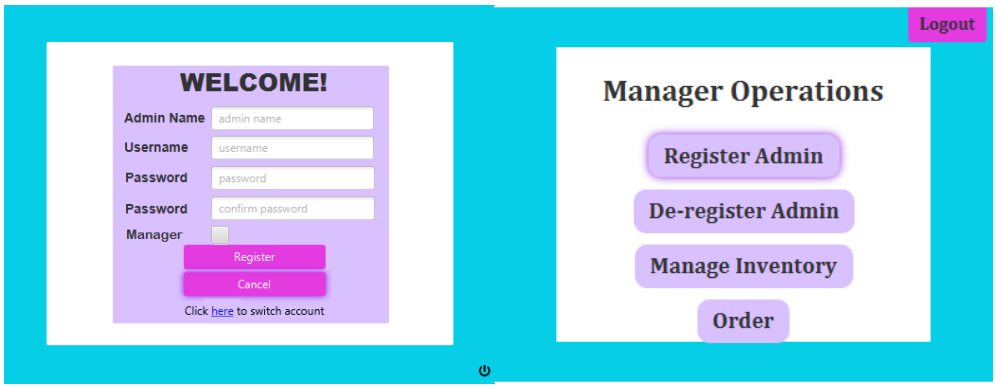


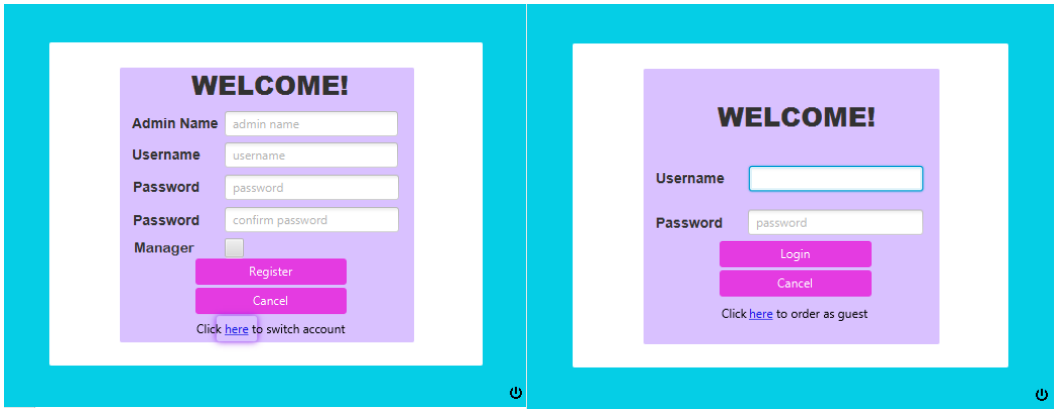


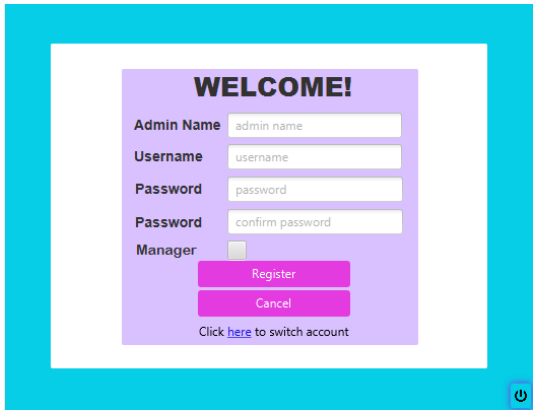


- **register.fxml**

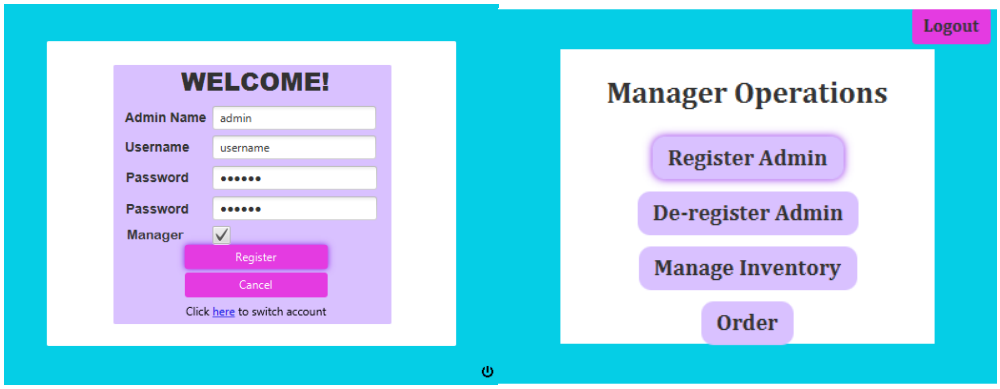

WELCOME!
Password does not match
Admin Name 123
Username 123
Password ••••
Password •••
Manager
Register
Cancel
Click here to switch account
WELCOME!
Username jiaxin has been taken
Admin Name 123
Username jiaxin
Password ••••••
Password ••••••
Manager
Register
Cancel
Click here to switch account


WELCOME!
Admin Name admin name
Username username
Password password
Password confirm password
Manager
Register
Cancel
Click here to switch account
Logout
Manager Operations
Register Admin
De-register Admin
Manage Inventory
Order

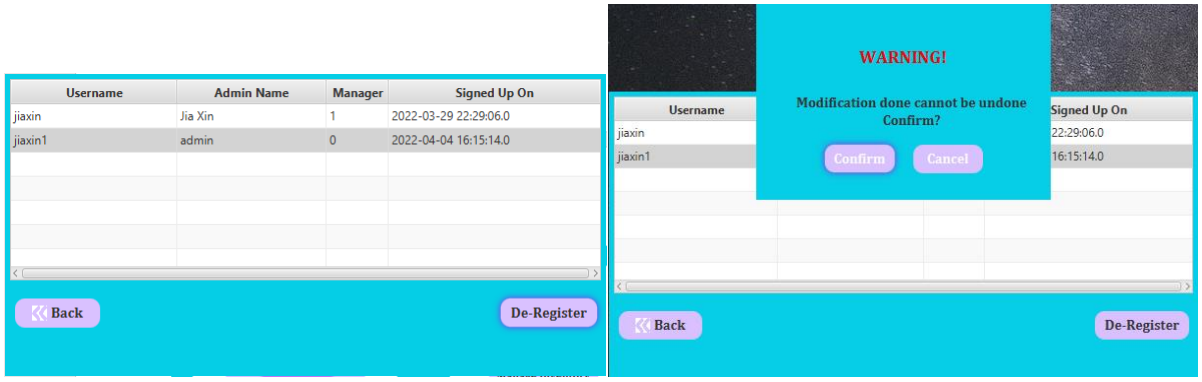

WELCOME!
Admin Name admin name
Username username
Password password
Password confirm password
Manager
Register
Cancel
Click here to switch account
WELCOME!
Username
Password password
Login
Cancel
Click here to order as guest

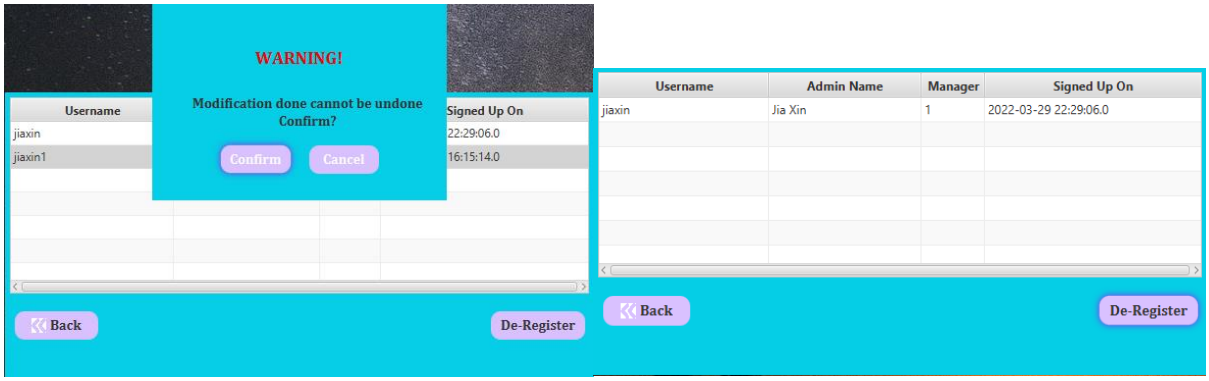

WELCOME!
Admin Name admin name
Username username
Password password
Password confirm password
Manager
Register
Cancel
Click here to switch account

Stage Close

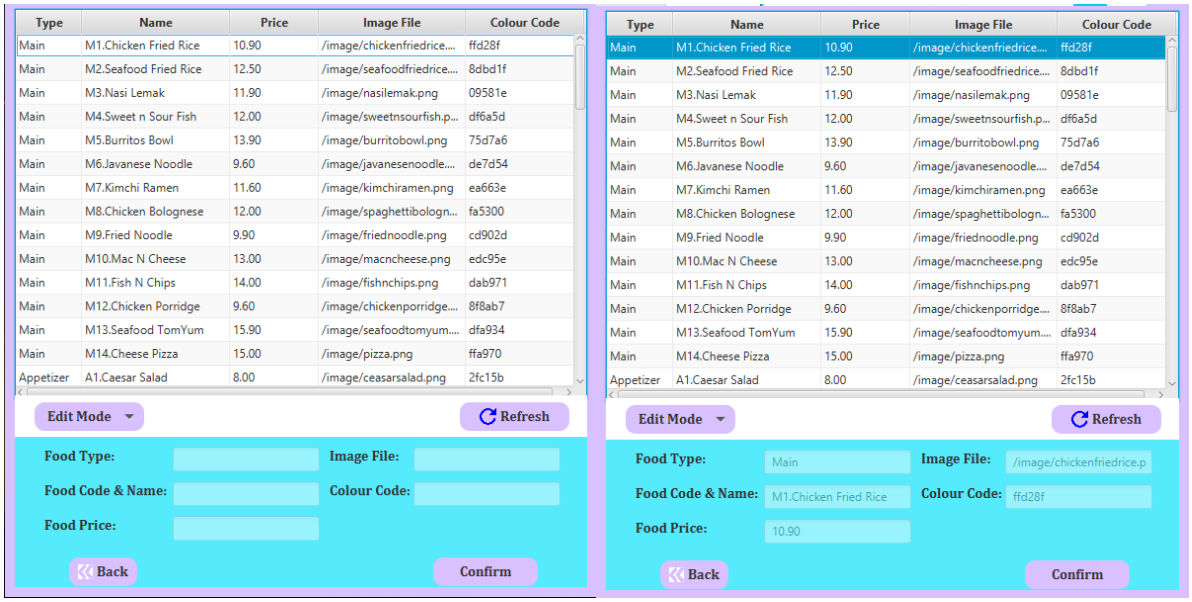


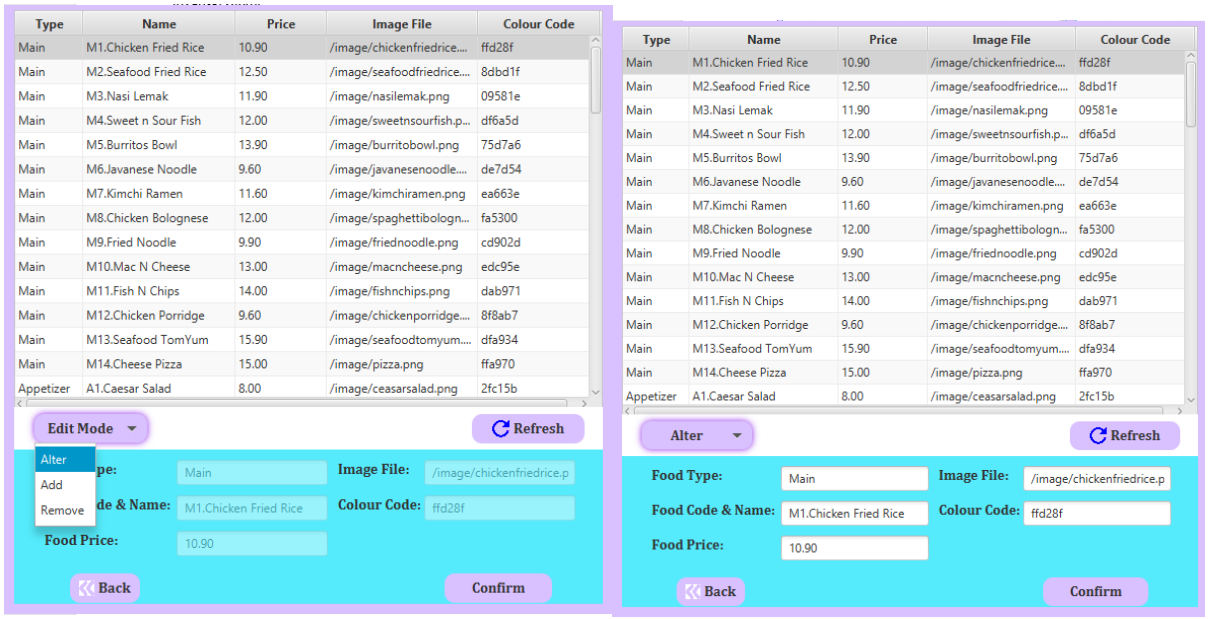
- **deregister.fxml & warning.fxml**

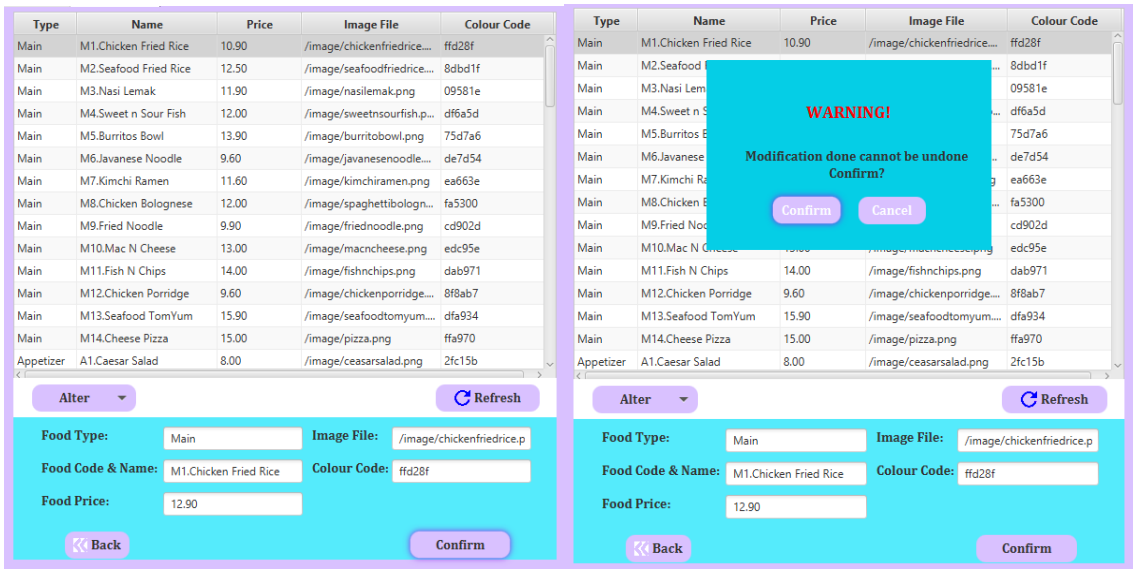


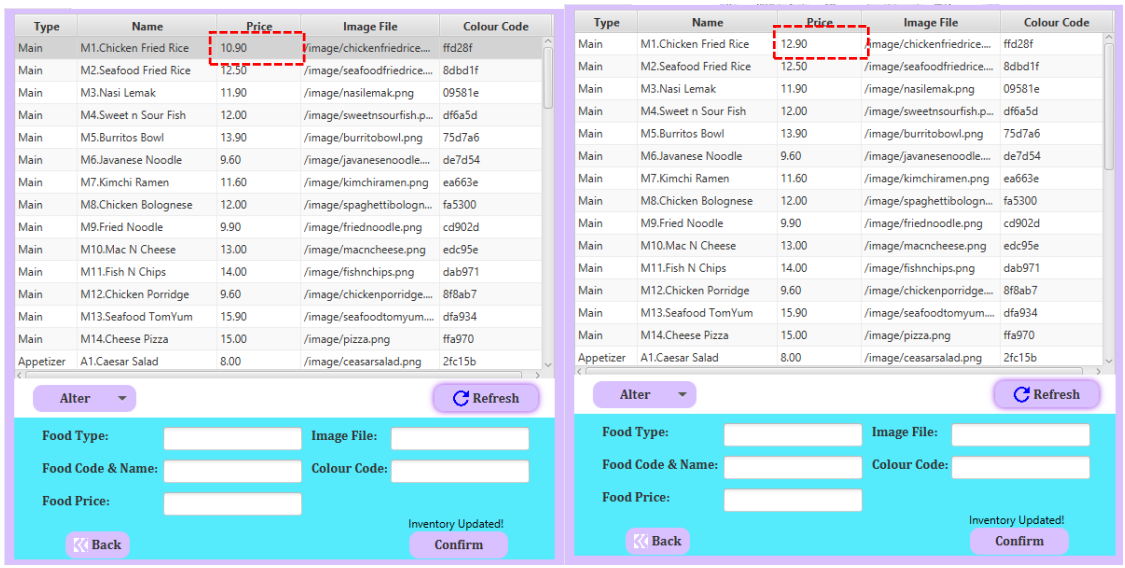


- **inventory.fxml**





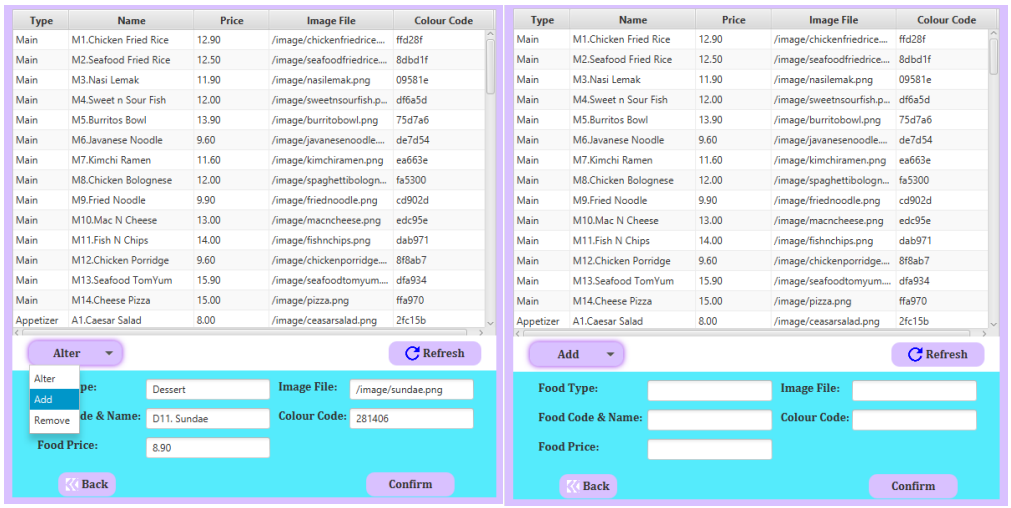

Type Name Price Image File Colour Code
Main M1.Chicken Fried Rice 10.90 /image/chickenfriedrice.... ffd28f
Main M2.Seafood Fried Rice 12.50 /image/seafoodfriedrice.... 8dbd1f
Main M3.Nasi Lemak 11.90 /image/nasilemak.png 09581e
Main M4.Sweet n Sour Fish 12.00 /image/sweetnsourfish.p... df6a5d
Main M5.Burritos Bowl 13.90 /image/burritobowl.png 75d7a6
Main M6.Javanese Noodle 9.60 /image/javanesenoodle.... de7d54
Main M7.Kimchi Ramen 11.60 /image/kimchiramen.png ea663e
Main M8.Chicken Bolognese 12.00 /image/spaghettibologn... fa5300
Main M9.Fried Noodle 9.90 /image/friednoodle.png cd902d
Main M10.Mac N Cheese 13.00 /image/macncheese.png edc95e
Main M11.Fish N Chips 14.00 /image/fishnchips.png dab971
Main M12.Chicken Porridge 9.60 /image/chickenporridge.... 8f8ab7
Main M13.Seafood TomYum 15.90 /image/seafoodtomyum.... dfa934
Main M14.Cheese Pizza 15.00 /image/pizza.png ffa970
Appetizer A1.Caesar Salad 8.00 /image/ceasarsalad.png 2fc15b
Alter
Refresh
Food Type: Main
Image File: /image/chickenfriedrice.p
Food Code & Name: M1.Chicken Fried Rice
Colour Code: ffd28f
Food Price: 12.90
Back
Confirm
WARNING!
Modification done cannot be undone
Confirm?
Confirm
Cancel

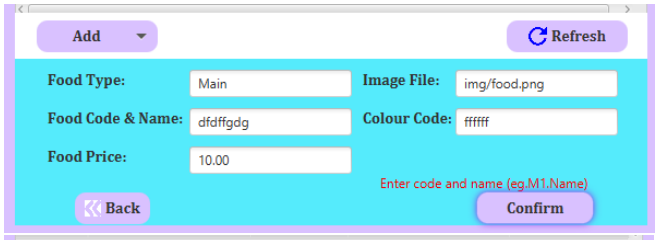

Type Name Price Image File Colour Code
Main M1.Chicken Fried Rice 10.90 /image/chickenfriedrice.... ffd28f
Main M2.Seafood Fried Rice 12.50 /image/seafoodfriedrice.... 8dbd1f
Main M3.Nasi Lemak 11.90 /image/nasilemak.png 09581e
Main M4.Sweet n Sour Fish 12.00 /image/sweetnsourfish.p... df6a5d
Main M5.Burritos Bowl 13.90 /image/burritobowl.png 75d7a6
Main M6.Javanese Noodle 9.60 /image/javanesenoodle.... de7d54
Main M7.Kimchi Ramen 11.60 /image/kimchiramen.png ea663e
Main M8.Chicken Bolognese 12.00 /image/spaghettibologn... fa5300
Main M9.Fried Noodle 9.90 /image/friednoodle.png cd902d
Main M10.Mac N Cheese 13.00 /image/macncheese.png edc95e
Main M11.Fish N Chips 14.00 /image/fishnchips.png dab971
Main M12.Chicken Porridge 9.60 /image/chickenporridge.... 8f8ab7
Main M13.Seafood TomYum 15.90 /image/seafoodtomyum.... dfa934
Main M14.Cheese Pizza 15.00 /image/pizza.png ffa970
Appetizer A1.Caesar Salad 8.00 /image/ceasarsalad.png 2fc15b
Alter
Refresh
Food Type:
Image File:
Food Code & Name:
Colour Code:
Food Price:
Inventory Updated!
Back
Confirm

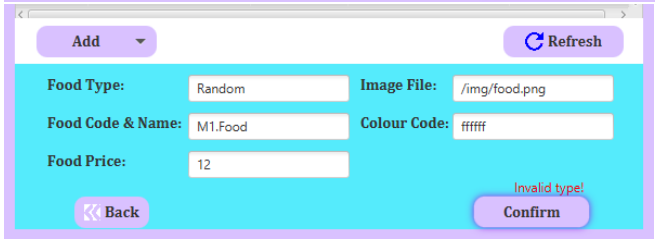

Type Name Price Image File Colour Code
Main M1.Chicken Fried Rice 12.90 /image/chickenfriedrice.... ffd28f
Main M2.Seafood Fried Rice 12.50 /image/seafoodfriedrice.... 8dbd1f
Main M3.Nasi Lemak 11.90 /image/nasilemak.png 09581e
Main M4.Sweet n Sour Fish 12.00 /image/sweetnsourfish.p... df6a5d
Main M5.Burritos Bowl 13.90 /image/burritobowl.png 75d7a6
Main M6.Javanese Noodle 9.60 /image/javanesenoodle.... de7d54
Main M7.Kimchi Ramen 11.60 /image/kimchiramen.png ea663e
Main M8.Chicken Bolognese 12.00 /image/spaghettibologn... fa5300
Main M9.Fried Noodle 9.90 /image/friednoodle.png cd902d
Main M10.Mac N Cheese 13.00 /image/macncheese.png edc95e
Main M11.Fish N Chips 14.00 /image/fishnchips.png dab971
Main M12.Chicken Porridge 9.60 /image/chickenporridge.... 8f8ab7
Main M13.Seafood TomYum 15.90 /image/seafoodtomyum.... dfa934
Main M14.Cheese Pizza 15.00 /image/pizza.png ffa970
Appetizer A1.Caesar Salad 8.00 /image/ceasarsalad.png 2fc15b
Alter
Refresh
Food Type:
Image File:
Food Code & Name:
Colour Code:
Food Price:
Inventory Updated!
Back
Confirm

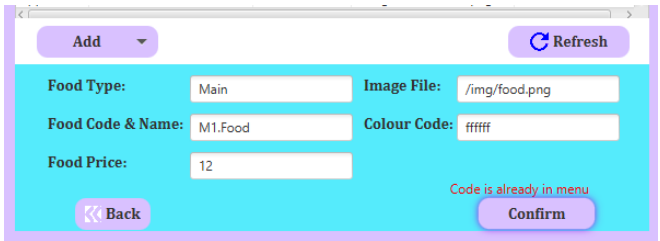
Type Name Price Image File Colour Code
Main M1.Chicken Fried Rice 12.90 /image/chickenfriedrice.... ffd28f
Main M2.Seafood Fried Rice 12.50 /image/seafoodfriedrice.... 8dbd1f
Main M3.Nasi Lemak 11.90 /image/nasilemak.png 09581e
Main M4.Sweet n Sour Fish 12.00 /image/sweetnsourfish.p... df6a5d
Main M5.Burritos Bowl 13.90 /image/burritobowl.png 75d7a6
Main M6.Javanese Noodle 9.60 /image/javanesenoodle.... de7d54
Main M7.Kimchi Ramen 11.60 /image/kimchiramen.png ea663e
Main M8.Chicken Bolognese 12.00 /image/spaghettibologn... fa5300
Main M9.Fried Noodle 9.90 /image/friednoodle.png cd902d
Main M10.Mac N Cheese 13.00 /image/macncheese.png edc95e
Main M11.Fish N Chips 14.00 /image/fishnchips.png dab971
Main M12.Chicken Porridge 9.60 /image/chickenporridge.... 8f8ab7
Main M13.Seafood TomYum 15.90 /image/seafoodtomyum.... dfa934
Main M14.Cheese Pizza 15.00 /image/pizza.png ffa970
Appetizer A1.Caesar Salad 8.00 /image/ceasarsalad.png 2fc15b
Alter
Alter
Add
Remove
Refresh
Food Type:
Image File:
Food Code & Name:
Colour Code:
Food Price:
Back
Confirm

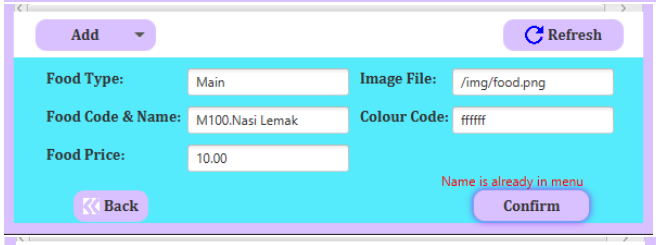
Type Name Price Image File Colour Code
Main M1.Chicken Fried Rice 12.90 /image/chickenfriedrice.... ffd28f
Main M2.Seafood Fried Rice 12.50 /image/seafoodfriedrice.... 8dbd1f
Main M3.Nasi Lemak 11.90 /image/nasilemak.png 09581e
Main M4.Sweet n Sour Fish 12.00 /image/sweetnsourfish.p... df6a5d
Main M5.Burritos Bowl 13.90 /image/burritobowl.png 75d7a6
Main M6.Javanese Noodle 9.60 /image/javanesenoodle.... de7d54
Main M7.Kimchi Ramen 11.60 /image/kimchiramen.png ea663e
Main M8.Chicken Bolognese 12.00 /image/spaghettibologn... fa5300
Main M9.Fried Noodle 9.90 /image/friednoodle.png cd902d
Main M10.Mac N Cheese 13.00 /image/macncheese.png edc95e
Main M11.Fish N Chips 14.00 /image/fishnchips.png dab971
Main M12.Chicken Porridge 9.60 /image/chickenporridge.... 8f8ab7
Main M13.Seafood TomYum 15.90 /image/seafoodtomyum.... dfa934
Main M14.Cheese Pizza 15.00 /image/pizza.png ffa970
Appetizer A1.Caesar Salad 8.00 /image/ceasarsalad.png 2fc15b
Remove
Refresh
Food Type:
Image File:
Food Code & Name:
Colour Code:
Food Price:
Back
Confirm

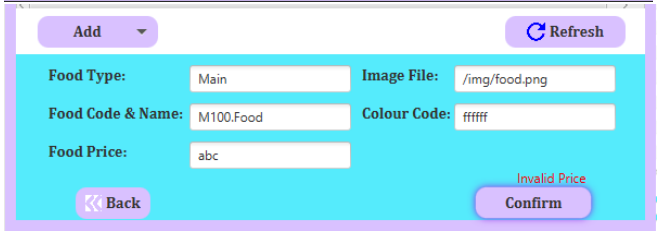

Type Name Price Image File Colour Code
Main M1.Chicken Fried Rice 12.90 /image/chickenfriedrice.... ffd28f
Main M2.Seafood Fried Rice 12.50 /image/seafoodfriedrice.... 8dbd1f
Main M3.Nasi Lemak 11.90 /image/nasilemak.png 09581e
Main M4.Sweet n Sour Fish 12.00 /image/sweetnsourfish.p... df6a5d
Main M5.Burritos Bowl 13.90 /image/burritobowl.png 75d7a6
Main M6.Javanese Noodle 9.60 /image/javanesenoodle.... de7d54
Main M7.Kimchi Ramen 11.60 /image/kimchiramen.png ea663e
Main M8.Chicken Bolognese 12.00 /image/spaghettibolognese.... fa5300
Main M9.Fried Noodle 9.90 /image/friednoodle.png cd902d
Main M10.Mac N Cheese 13.00 /image/macncheese.png edc95e
Main M11.Fish N Chips 14.00 /image/fishnchips.png dab971
Main M12.Chicken Porridge 9.60 /image/chickenporridge.... 8f8ab7
Main M13.Seafood TomYum 15.90 /image/seafoodtomyum.... dfa934
Main M14.Cheese Pizza 15.00 /image/pizza.png ffa970
Appetizer A1.Caesar Salad 8.00 /image/ceasarsalad.png 2fc15b
Alter
Alter
Add
Remove
Refresh
Dessert
Image File:
/image/sundae.png
D11. Sundae
Colour Code:
281406
Food Price:
8.90
Back
Confirm
Add
Refresh
Food Type:
Image File:
Food Code & Name:
Colour Code:
Food Price:
Back
Confirm


Add
Refresh
Food Type:
Main
Image File:
img/food.png
Food Code & Name:
dfdffgdg
Colour Code:
ffffff
Food Price:
10.00
Enter code and name (eg.M1.Name)
Back
Confirm

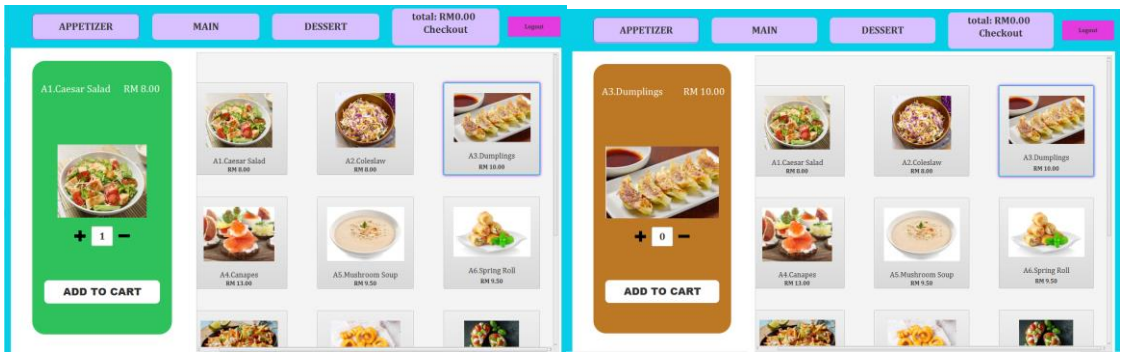

Add
Refresh
Food Type:
Random
Image File:
/img/food.png
Food Code & Name:
M1.Food
Colour Code:
ffffff
Food Price:
12
Invalid type!
Back
Confirm

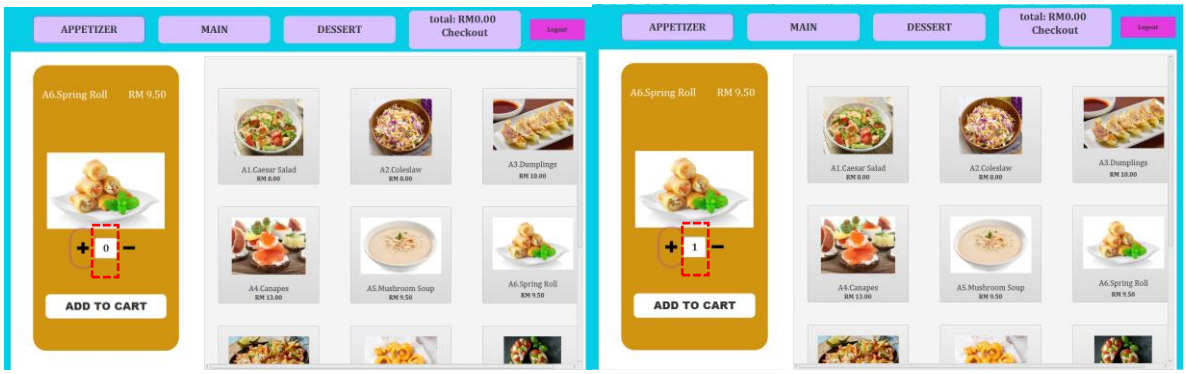

Add
Refresh
Food Type:
Main
Image File:
/img/food.png
Food Code & Name:
M1.Food
Colour Code:
ffffff
Food Price:
12
Code is already in menu
Back
Confirm

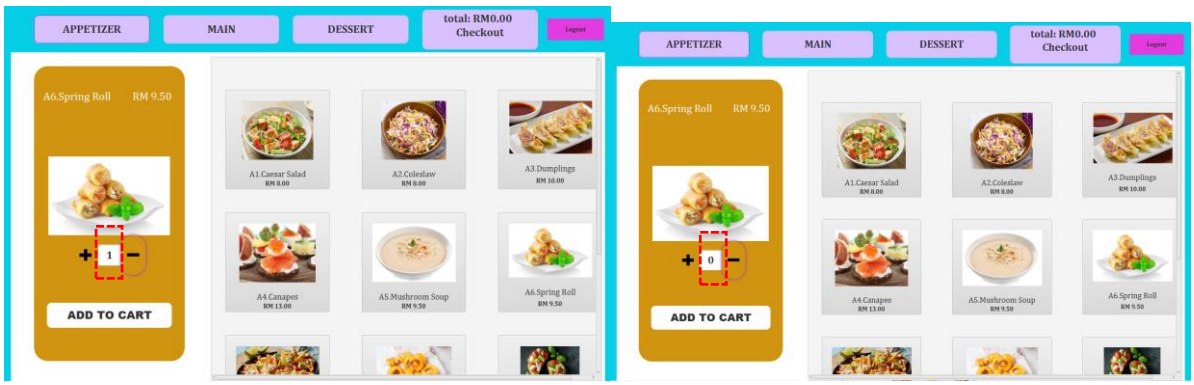

Add
Refresh
Food Type:
Main
Image File:
/img/food.png
Food Code & Name:
M100.Nasi Lemak
Colour Code:
ffffff
Food Price:
10.00
Name is already in menu
Back
Confirm

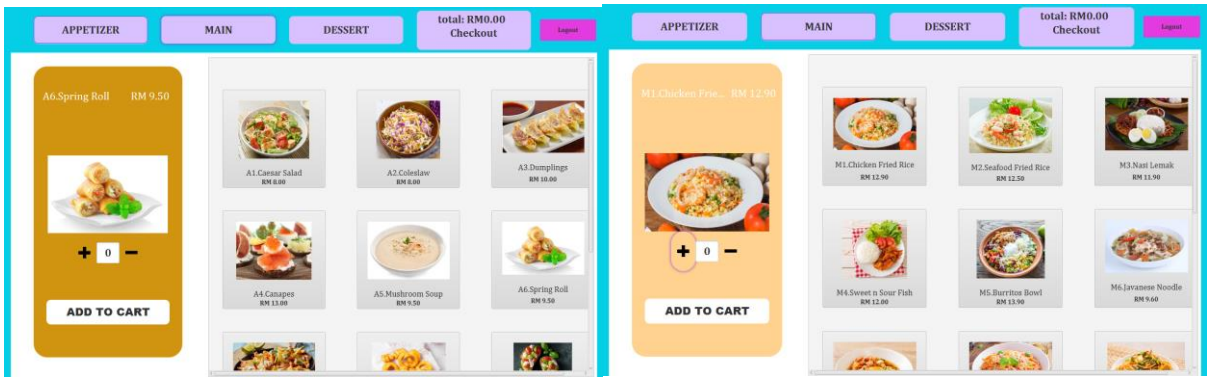

Add
Refresh
Food Type:
Main
Image File:
/img/food.png
Food Code & Name:
M100.Food
Colour Code:
ffffff
Food Price:
abc
Invalid Price
Back
Confirm

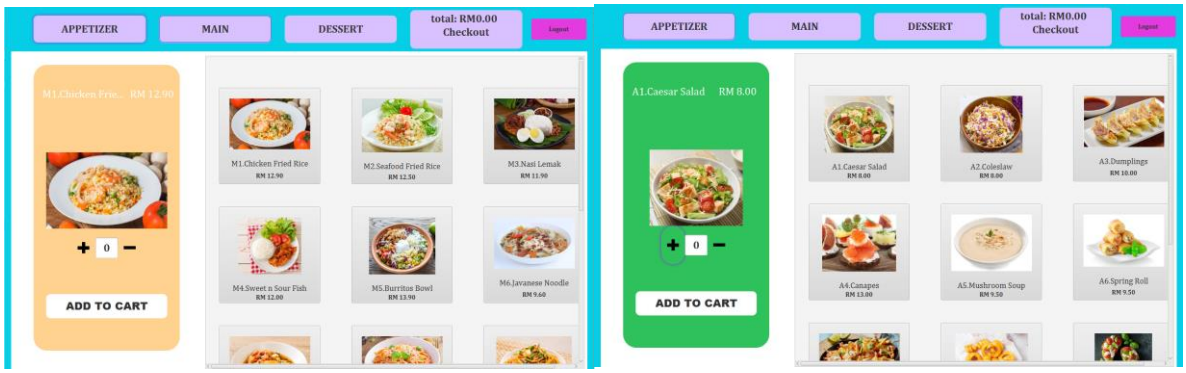
- **appetizermenu.fxml & mainmenu.fxml & dessertmenu.fxml**

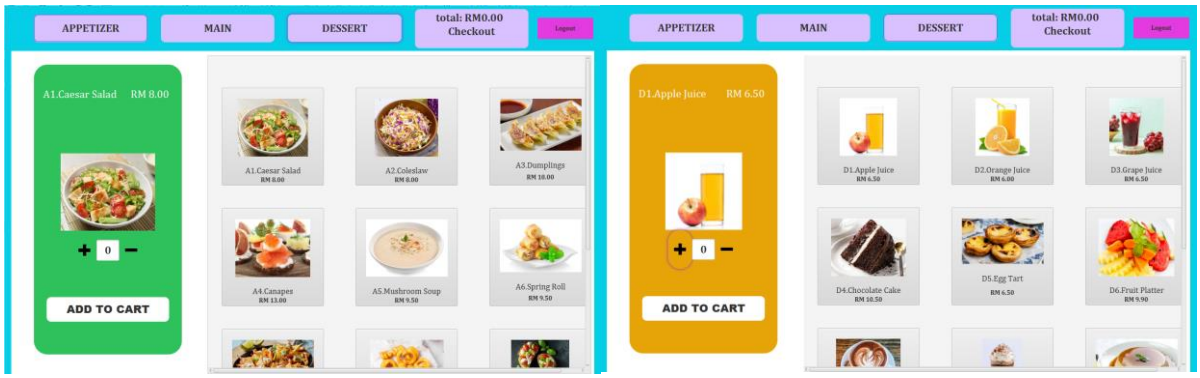


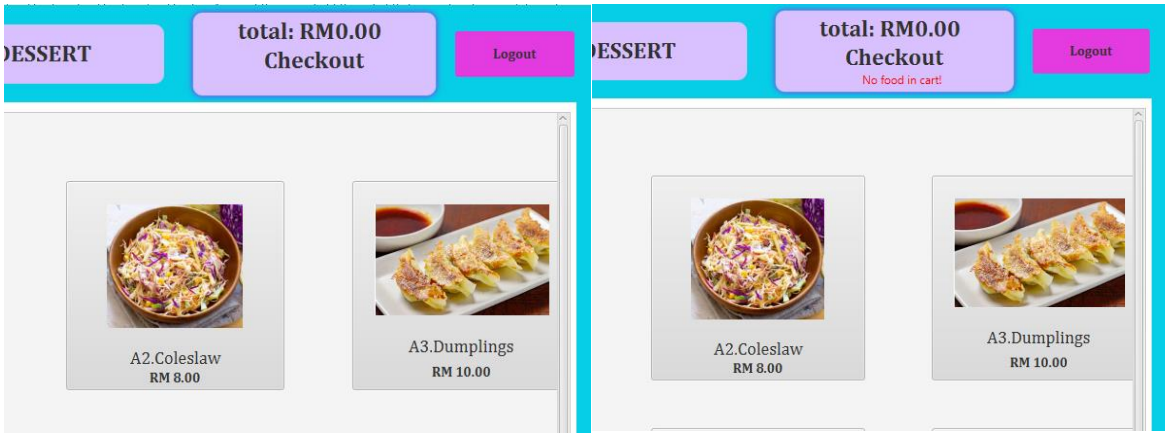


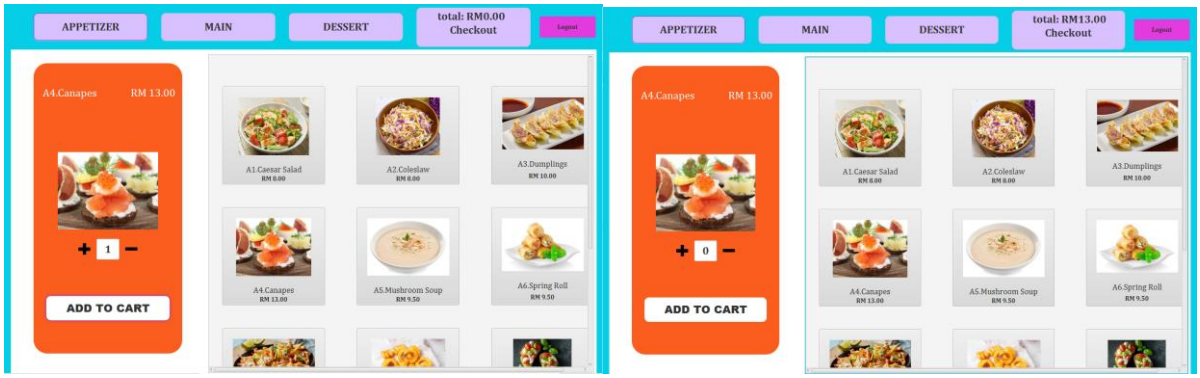


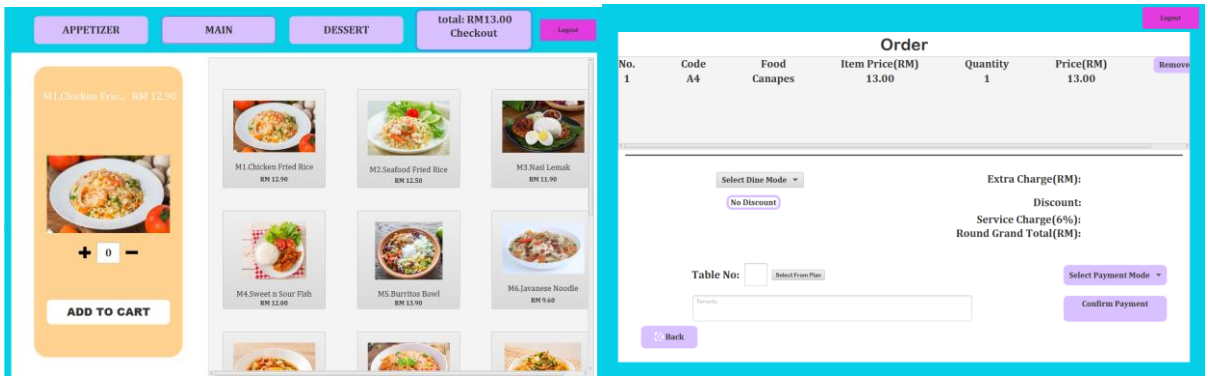

APPETIZER
MAIN
DESSERT
total: RM0.00
Checkout
Logout
A6.Spring Roll RM 9.50
ADD TO CART
A1.Caesar Salad
A2.Coleslaw
A3.Dumplings
A4.Canapes
A5.Mushroom Soup
A6.Spring Roll
M1.Chicken Fried Rice
M2.Seafood Fried Rice
M3.Nasi Lemak
M4.Sweet n Sour Fish
M5.Burritos Bowl
M6.Javanese Noodle

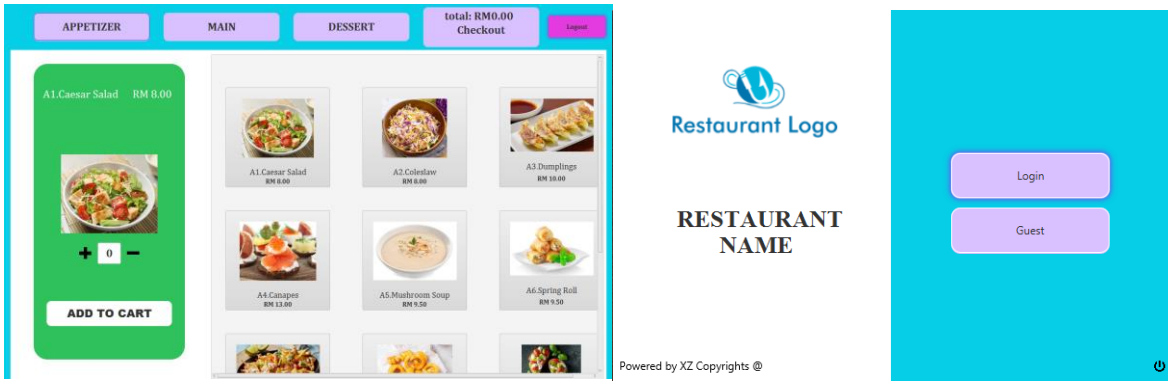

APPETIZER
MAIN
DESSERT
total: RM0.00
Checkout
Logout
ADD TO CART
A1.Caesar Salad RM 8.00
A1.Caesar Salad
A2.Coleslaw
A3.Dumplings
A4.Canapes
A5.Mushroom Soup
A6.Spring Roll
M1.Chicken Fried Rice
M2.Seafood Fried Rice
M3.Nasi Lemak
M4.Sweet n Sour Fish
M5.Burritos Bowl
M6.Javanese Noodle


APPETIZER
MAIN
DESSERT
total: RM0.00
Checkout
Logout
ADD TO CART
A1.Caesar Salad RM 8.00
D1.Apple Juice RM 6.50
A1.Caesar Salad
A2.Coleslaw
A3.Dumplings
A4.Canapes
A5.Mushroom Soup
A6.Spring Roll
D1.Apple Juice
D2.Orange Juice
D3.Grape Juice
D4.Chocolate Cake
D5.Egg Tart
D6.Fruit Platter

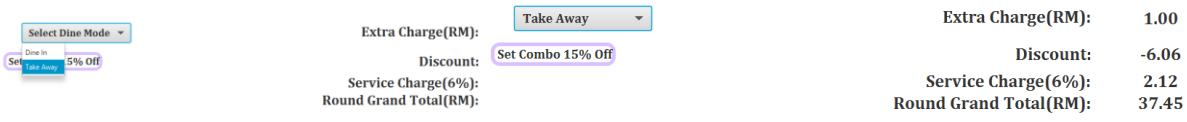

DESSERT
total: RM0.00
Checkout
Logout
A2.Coleslaw
RM 8.00
A3.Dumplings
RM 10.00
DESSERT
total: RM0.00
Checkout
No food in cart!
Logout
A2.Coleslaw
RM 8.00
A3.Dumplings
RM 10.00



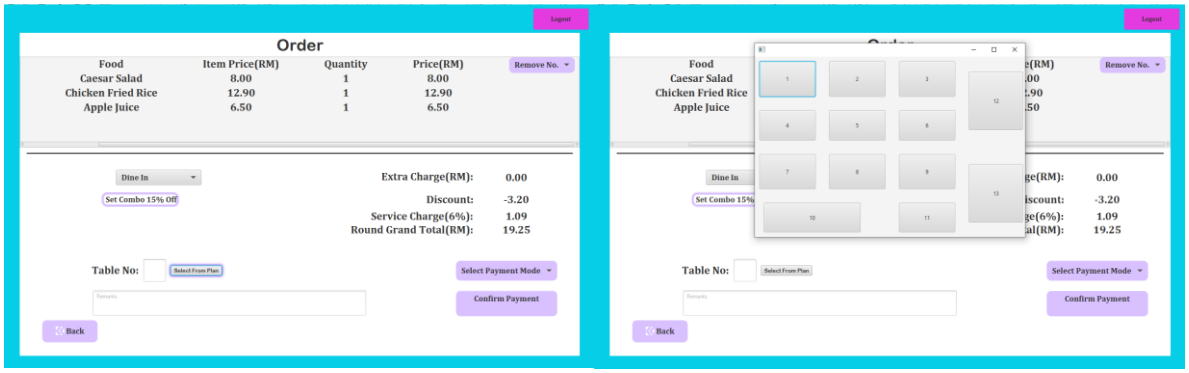


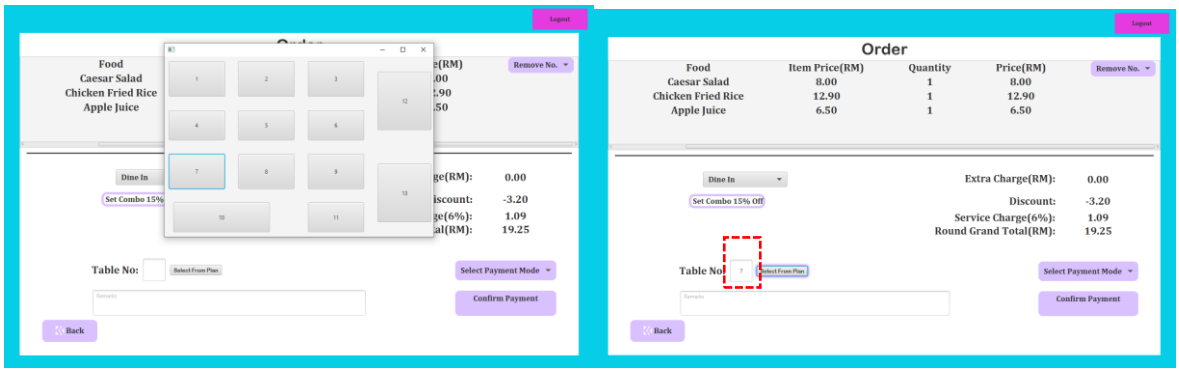


- **checkout.fxml & tableplan.fxml**

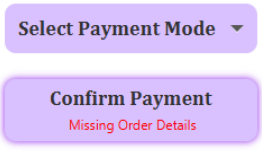


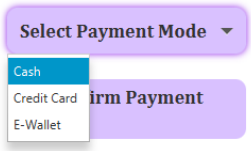

Food
Item Price(RM)
Quantity
Price(RM)
Remove No.
Canapes
13.00
1
13.00
Caesar Salad
8.00
1
8.00
Chicken Fried Rice
12.90
1
12.90
Apple Juice
6.50
1
6.50

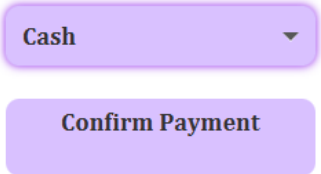

Logout
Order
Food
Item Price(RM)
Quantity
Price(RM)
Remove No.
Caesar Salad
8.00
1
8.00
Chicken Fried Rice
12.90
1
12.90
Apple Juice
6.50
1
6.50
Dine In
Set Combo 15% Off
Extra Charge(RM):
0.00
Discount:
-3.20
Service Charge(6%):
1.09
Round Grand Total(RM):
19.25
Table No:
Select From Plan
Select Payment Mode
Confirm Payment
Back


Logout
Order
Food
Item Price(RM)
Quantity
Price(RM)
Remove No.
Caesar Salad
8.00
1
8.00
Chicken Fried Rice
12.90
1
12.90
Apple Juice
6.50
1
6.50
Dine In
Set Combo 15% Off
Extra Charge(RM):
0.00
Discount:
-3.20
Service Charge(6%):
1.09
Round Grand Total(RM):
19.25
Table No:
Select From Plan
Select Payment Mode
Confirm Payment
Back

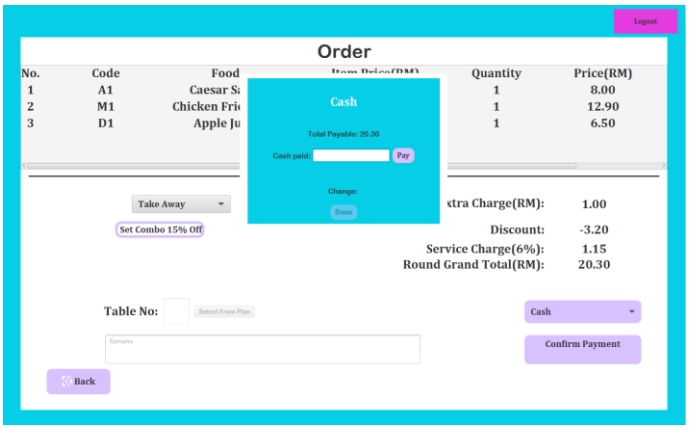
Select Payment Mode

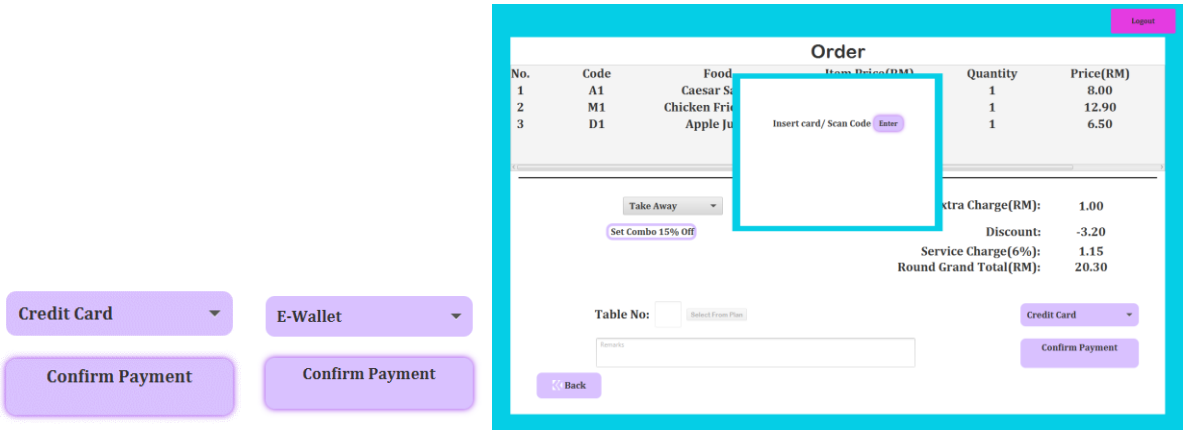

Confirm Payment
Missing Order Details

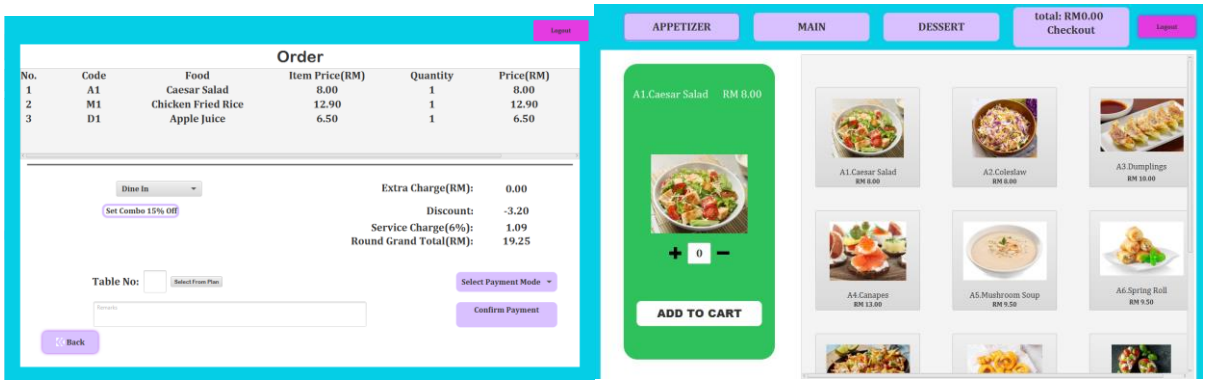

Select Payment Mode
Cash
Credit Card
E-Wallet


Cash

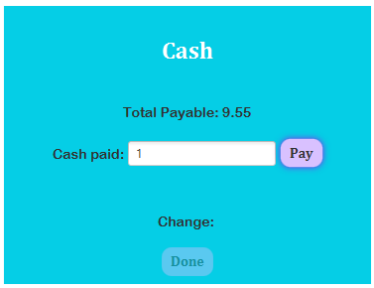
Confirm Payment

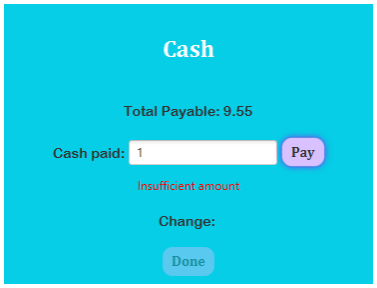

Logout
Order
No.
Code
Food
Quantity
Price(RM)
1
A1
Caesar S
1
8.00
2
M1
Chicken Fri
1
12.90
3
D1
Apple Ju
1
6.50
Cash
Total Payable: 20.30
Cash paid:
Pay
Change:
Done
Take Away
Set Combo 15% Off
xtra Charge(RM):
1.00
Discount:
-3.20
Service Charge(6%):
1.15
Round Grand Total(RM):
20.30
Table No:
Select From Plan
Cash
Confirm Payment
Back

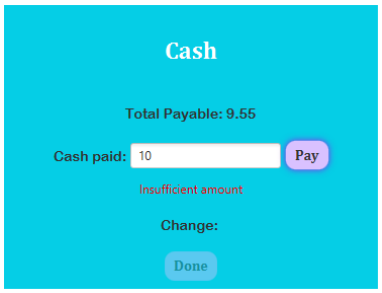


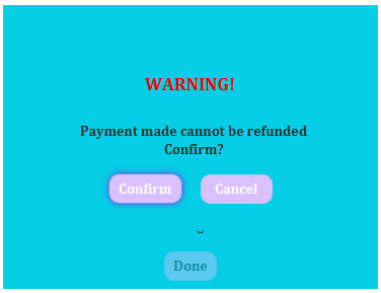


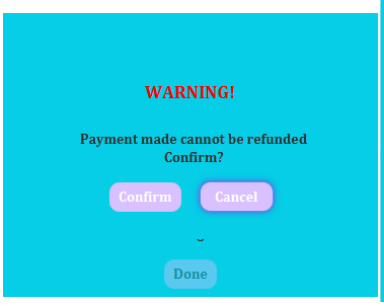
- **cash.fxml & paymentconfirmation.fxml**

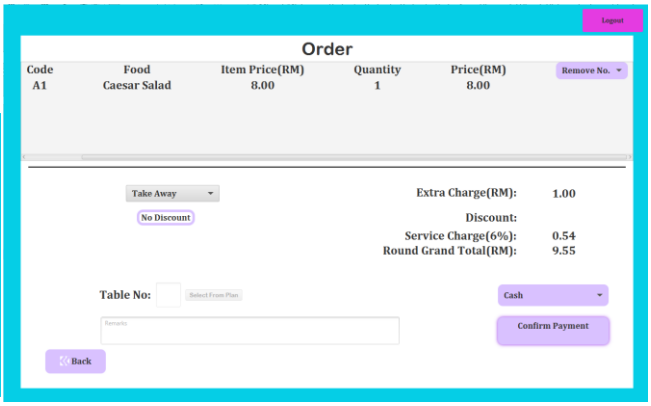


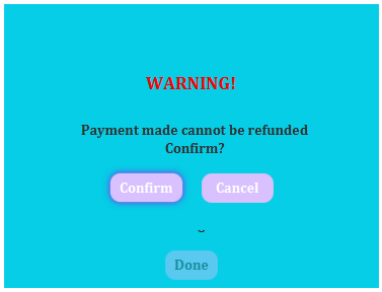


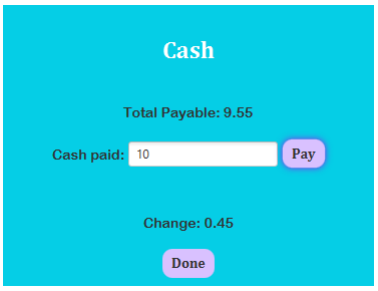


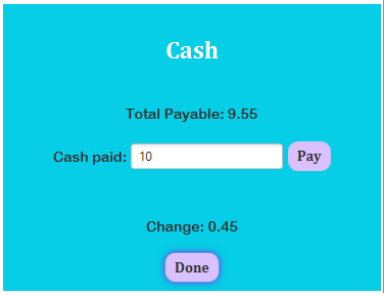


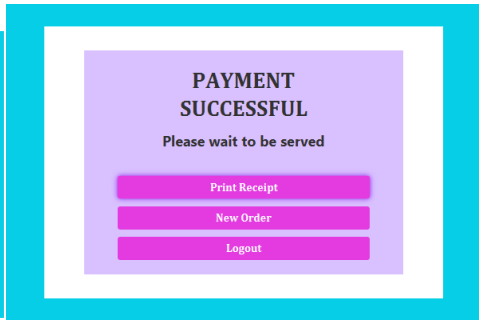




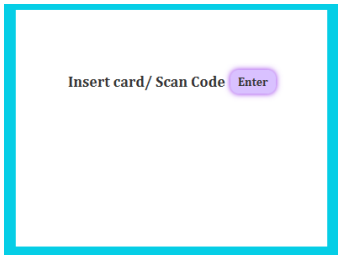


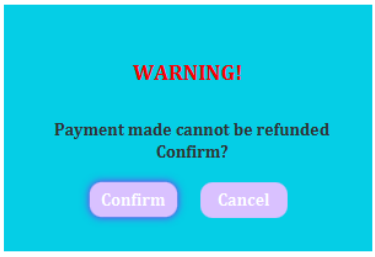


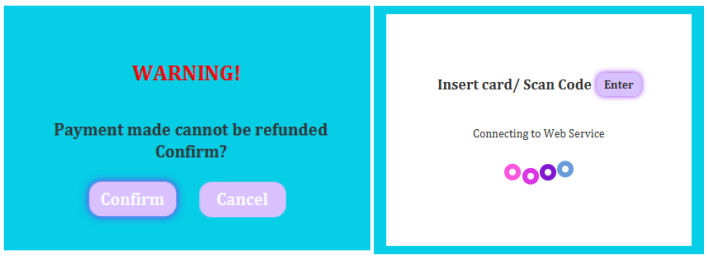




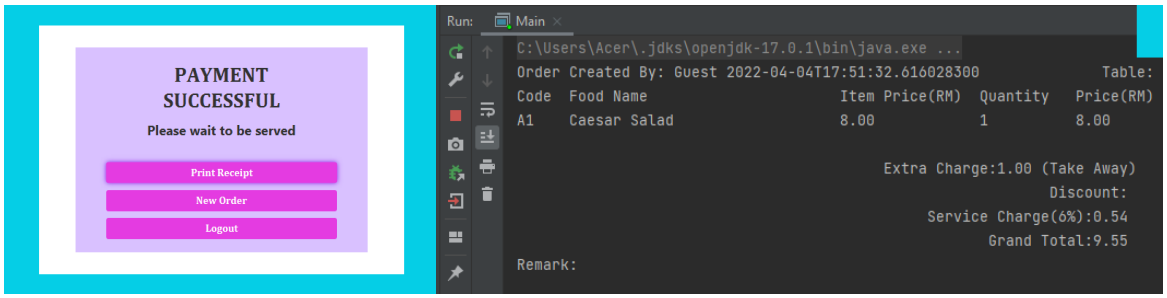
- **connectbank.fxml**

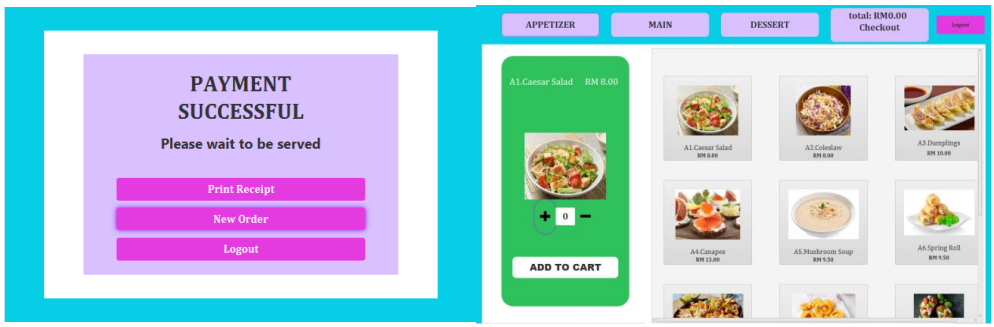






- **endpage.fxml**





Conclusion

Through the development process of this system, I am able to familiarise and put my knowledge on Javafx in practice, as well as gain more knowledge about this framework. I have learnt how to set-up a scene by implementing Initializable, connect project to database, display collection of objects on TableView, creating animation and using EventListener. I really enjoyed trying out and exploring on new approach to a problem. There were minor difficulties faced such as having to using values from another scene, and all were solved by some trial and error or researches done on the Internet. This system is yet far from being implemented in real life, but I have done my best attempt to achieve the basic requirements within the one-month time allocated for this project.