Even Semester (2020)

BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:		**Surname**	**Given Names**	**Student ID Number**
	1.	**Wijaya**	**Alvian**	**2301891595**

Course Code	**: COMP6510**	**Course Name**	**: Programming Language**
Class	**: L2AC**	**Name of Lecturer**	**:** Jude Joseph Lamug Martinez
Major	**: CS**		
Title of Assignment (if any)	: Point of Sale And Inventory Management		
Type of Assignment	**: Final Project**		
Submission Pattern			
Due Date	**: 20 – 06 - 2020**	**Submission Date**	**: 20 – 06 - 2020**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I/we* understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I/we* declare that the work contained in this assignment is my/our* own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

Table of Contents

Project Specification

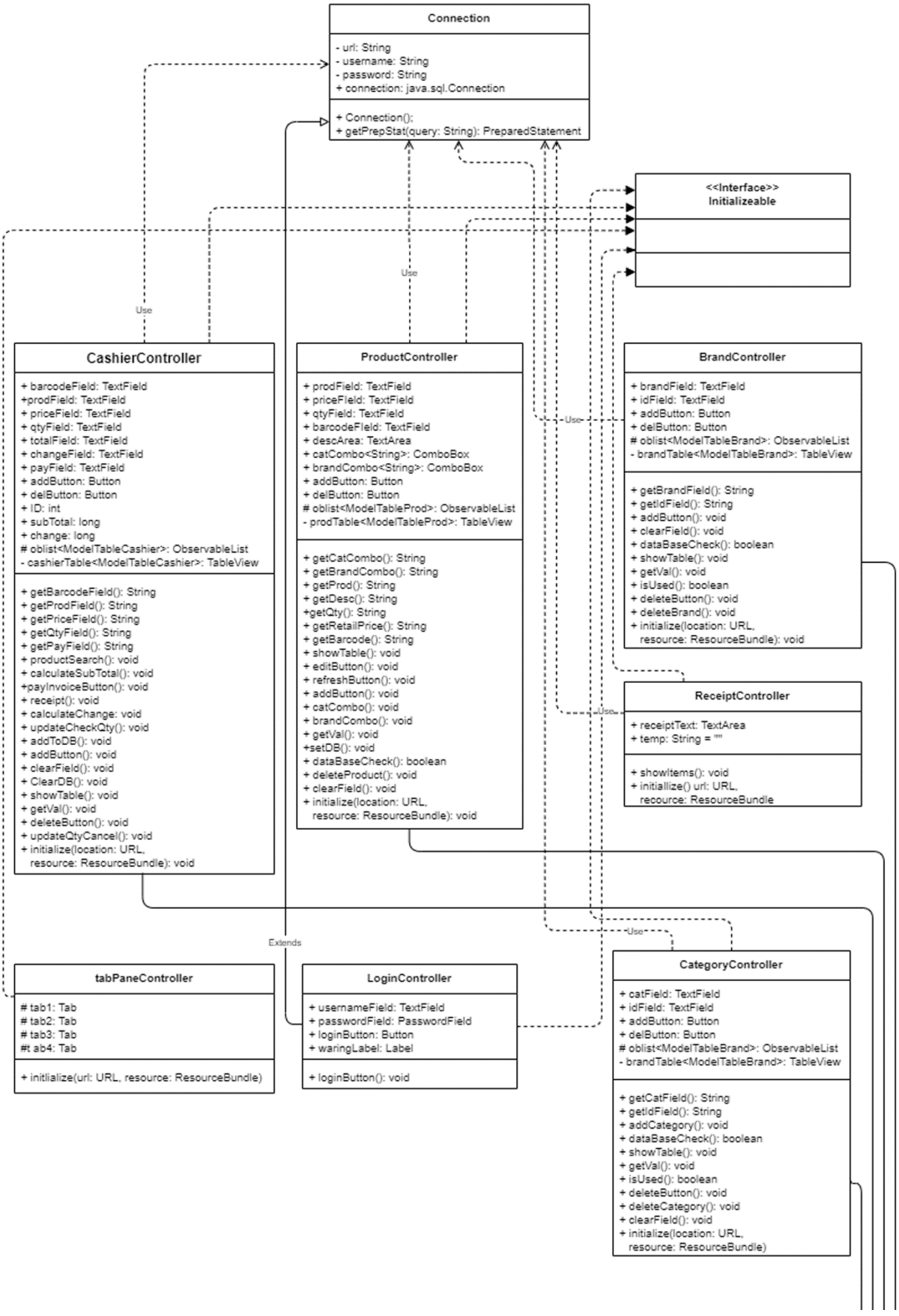
Problem

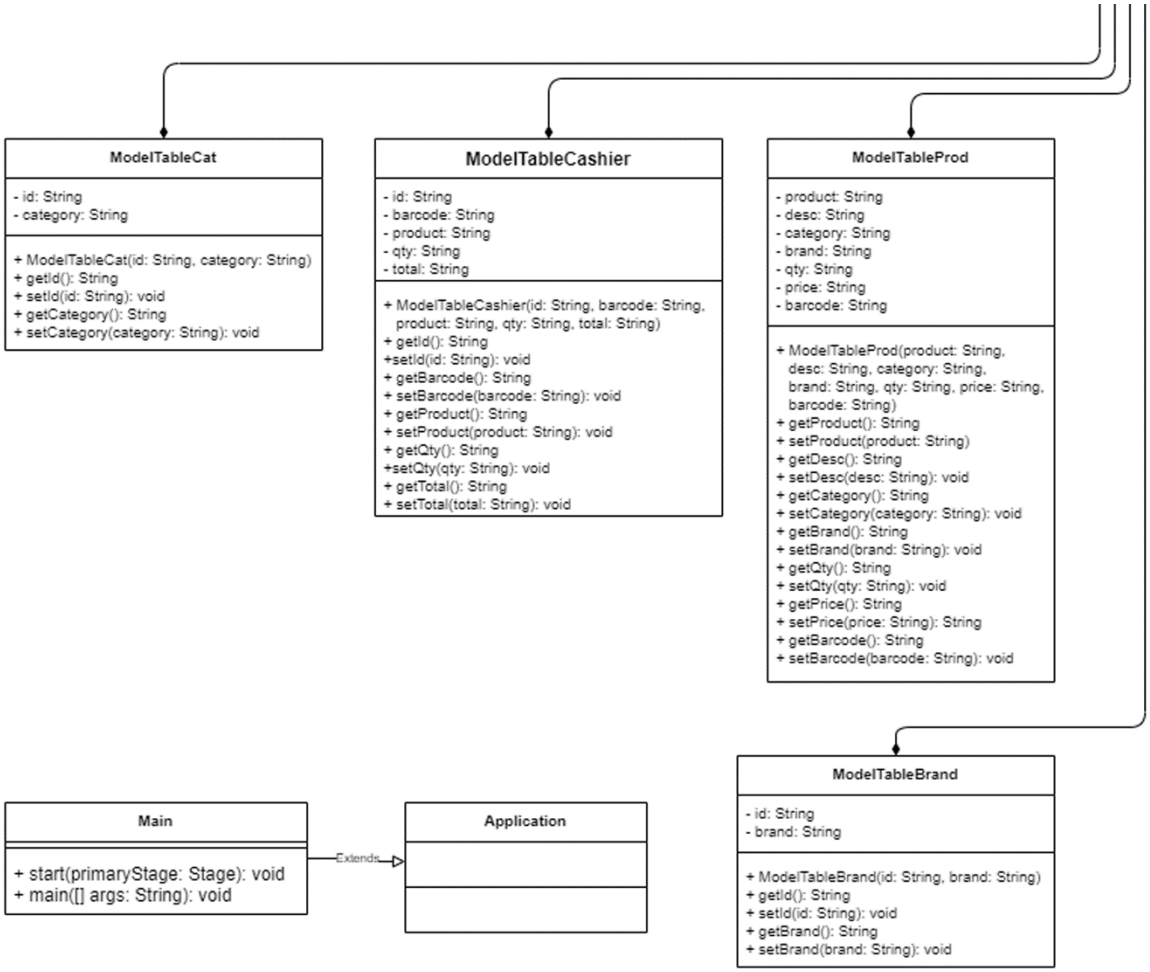
In this modern era, there is a lot of company (especially supermarket) use computer for their inventory system and also their cashier. Cashier can easily check the quantity of each product and they can calculate the receipt faster by using Point of Sale. So, it is very important now days to create a simple inventory management and POS program.

Solution

To solve the problem above, I am making a Point of Sale and Inventory Management System program that will answer those problem. This system use database and GUI. The database is used to store the inventory management data so when the program is closed, the data that has been already stored doesn't disappear. And, the GUI is used for simpler user interaction with the program.

Solution Design




ModelTableCat
- id: String
- category: String
+ ModelTableCat(id: String, category: String)
+ getId(): String
+ setId(id: String): void
+ getCategory(): String
+ setCategory(category: String): void
ModelTableCashier
- id: String
- barcode: String
- product: String
- qty: String
- total: String
+ ModelTableCashier(id: String, barcode: String, product: String, qty: String, total: String)
+ getId(): String
+setId(id: String): void
+ getBarcode(): String
+ setBarcode(barcode: String): void
+ getProduct(): String
+ setProduct(product: String): void
+ getQty(): String
+setQty(qty: String): void
+ getTotal(): String
+ setTotal(total: String): void
ModelTableProd
- product: String
- desc: String
- category: String
- brand: String
- qty: String
- price: String
- barcode: String
+ ModelTableProd(product: String, desc: String, category: String, brand: String, qty: String, price: String, barcode: String)
+ getProduct(): String
+ setProduct(product: String)
+ getDesc(): String
+ setDesc(desc: String): void
+ getCategory(): String
+ setCategory(category: String): void
+ getBrand(): String
+ setBrand(brand: String): void
+ getQty(): String
+ setQty(qty: String): void
+ getPrice(): String
+ setPrice(price: String): String
+ getBarcode(): String
+ setBarcode(barcode: String): void
ModelTableBrand
- id: String
- brand: String
+ ModelTableBrand(id: String, brand: String)
+ getId(): String
+ setId(id: String): void
+ getBrand(): String
+ setBrand(brand: String): void
Main
+ start(primaryStage: Stage): void
+ main([] args: String): void
Extends
Application

Discussion

Implementation

This project is made possible by using variety of built-in libraries in Java. This project use a lot of libraries, for example:

- javafx – To make the GUI.
- java.net – Use for initlializeable interface.
- java.util – Use for initilaizeable interface.
- java.sql – To help the program access data from and to the database.
- java.io. IOException – Use to control the try and catch in the program.

```
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;

import java.net.URL;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.*;
import java.io.IOException;
```

How It Works

- First, user have to input a valid credential(username and password) to log in. If user input a wrong username and/or password, it will show a label that inform the user for invalid credential. If the credential is correct, it will automatically pop up a new window with cashier, product, brand, and category tab.
- In cashier tab(POS), user can input the item that the consumer buy. It will automatically calculate the total price and how much the change that the user will get. It also update the quantity of the product in the database. If the program closed by accident and the “cart” is still full of product that consumer will buy, it will access the database and show the product again. After that, if user want to delete a product in the “cart”, it can remove it. If user want to pay and it’s smaller than Subtotal of the product, it will pop up a message window. If not, it will show a new window with “receipt” inside it.
- In Product Tab, user can add new, edit, and remove a product from the inventory system. It also check if the barcode is not duplicated from another product. If not, it will insert the product to the database.
- In Brand and Category tabs, user can add new brand/category of product and check for duplicate data(either {brand/category} or ID). If there is no duplicate in the database. It will insert the data to the database.

Code Explanation

- Connection class

  This class is specifically made to connect the java code to the database and also to prevent code duplication in other classes.

```
private String url = "jdbc:mysql://localhost:3306/pos_system?useSSL=false";
    private String username = "root";
    private String password = "AlvianWijaya";
    public java.sql.Connection connection = null;


    // Class constructor of Connection class
    public Connection() {
        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(url, username, password);
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


    // Function that return prepStat
    public PreparedStatement getPrepStat(String query) {
        try {
            PreparedStatement prepStat = connection.prepareStatement(query);
            return prepStat;
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return null;
    }


    // Function that return value of username
    public String getUsername() {
        return username;
    }


    // Function that return value of username
    public String getPassword() {
        return password;
    }
}
```

- Brand Controller, Cashier Controller, Category Controller, Product Controller

  All of this class is pretty much have the same basic function. This class is made to control the FXML file of the GUI. It also control all input and output of the program. It is also connected to the database via Connection Class. Here is an example of the code from ProductController class:

```
public class ProductController implements Initializable {
    Connection connect = new Connection();
    public TextField prodField, priceField, qtyField, barcodeField;
    public TextArea descArea;
    public ComboBox<String> catCombo;
    public ComboBox<String> brandCombo;
    public Button addButton, delButton;
    ObservableList<ModelTableProd> oblist = FXCollections.observableArrayList();


    // ProductController class associated with ModelTableProd class
    @FXML
    private TableView<ModelTableProd> prodTable;
```

```
    // Function that return value of catCombo combobox
    public String getCatCombo() {
        return catCombo.getValue();
    }

    // Function that return value of brandCombo combobox
    public String getBrandCombo() {
        return brandCombo.getValue();
    }

    // Function that return value of prodField textfield
    public String getProd() {
        return prodField.getText();
    }

    // Function that return value of deascArea textarea
    public String getDesc() {
        return descArea.getText();
    }


    // Function that return value of qtyField textfield
    public String getQty() {
        try {
            Integer.parseInt(qtyField.getText());
            return qtyField.getText();
        } catch (Exception e) {
            return "";
        }
    }


    // Function that return value of priceField textfield
    public String getRetailPrice() {
        try {
            Integer.parseInt(priceField.getText());
            return priceField.getText();
        } catch (Exception e) {
            return "";
        }
    }


    // Function that return value of barcodeField textfield
    public String getBarcode() {
        return barcodeField.getText();
    }


    // Function that fill prodTable
    public void showTable() {
        try {
            PreparedStatement prepStat = connect.getPrepStat("SELECT * FROM product");
            ResultSet rs = prepStat.executeQuery();

            while (rs.next()) {
                oblist.add((new ModelTableProd(rs.getString("Product"), rs.getString("Description"),
rs.getString("Category"), rs.getString("Brand"), rs.getString("Qty"), rs.getString("RetailPrice"),
rs.getString("Barcode"))));
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


    // editButton function
    // This function is used to update the values of the product in database
    public void editButton() {
        PreparedStatement prepStat = connect.getPrepStat("UPDATE product SET Product = ?, Description
= ?, Category = ?, Brand = ?, Qty = ?, RetailPrice = ?, Barcode = ?  WHERE Barcode = '" +
getBarcode() + "'");
        try {
            prepStat.setString(1, getProd());
            prepStat.setString(2, getDesc());
            prepStat.setString(3, getCatCombo());
```

```
            prepStat.setString(4, getBrandCombo());
            prepStat.setString(5, getQty());
            prepStat.setString(6, getRetailPrice());
            prepStat.setString(7, getBarcode());
            prepStat.executeUpdate();

            // Pop up a message
            Alert alert1 = new Alert(Alert.AlertType.INFORMATION);
            alert1.setTitle("Info");
            alert1.setContentText("CHANGE SAVED");
            alert1.setHeaderText("Success updated product!");
            alert1.show();
            clearField();
            prodTable.getItems().clear();
            showTable();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


    // This function is used to refresh data inside table
    public void refreshButton() {
        barcodeField.setEditable(true);
        clearField();
        prodTable.getItems().clear();
        showTable();
    }


    // addButton Function
    public void addButton() {

        // Check if the user already filled all of input
        // If it return true, it will show a pop up message
        if (getProd().isEmpty() || getCatCombo() == null || getBrandCombo() == null ||
getBarcode().isEmpty() ||
                getDesc().isEmpty() || getQty().isEmpty() || getRetailPrice().isEmpty()) {
            Alert a = new Alert(Alert.AlertType.WARNING);
            a.setTitle("Warning");
            a.setContentText("Check data input!");
            a.show();
        } else {
            getRetailPrice();

            // Check the user for appropriate input
            // If input is appropriate, it will show a pop up message
            if (getProd().length() <= 30 & getBarcode().length() == 4 && getDesc().length() <= 255 &&
                    getQty().length() <= 255 && getRetailPrice().length() <= 255) {

                // Check if data is already in database
                // If yes, it will show a pop up message
                if (dataBaseCheck()) {
                    Alert alert1 = new Alert(Alert.AlertType.ERROR);
                    alert1.setTitle("ERROR");
                    alert1.setContentText("Barcode cannot be repeated!");
                    alert1.setHeaderText("SOMETHING WRONG");
                    alert1.show();

                    // If no, it will insert the data to database
                } else {
                    setDB();
                    clearField();
                    prodTable.getItems().clear();
                    showTable();
                }

            } else {
                Alert alert3 = new Alert(Alert.AlertType.WARNING);
                alert3.setTitle("Warning");
                alert3.setContentText("Check all inputs!");
                alert3.show();
            }
        }
    }
```

```
    // Function that control catCombo combobox
    public void catCombo() {
        try {
            PreparedStatement prepStat = connect.getPrepStat("SELECT * FROM category");
            ResultSet rs = prepStat.executeQuery();
            catCombo.getItems().clear();
            while (rs.next()) {
                catCombo.getItems().add(rs.getString("Category"));
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }

    }


    // Function that control brandCombo combobox
    public void brandCombo() {
        try {
            PreparedStatement prepStat = connect.getPrepStat("SELECT * FROM brand");
            ResultSet rs = prepStat.executeQuery();
            brandCombo.getItems().clear();
            while (rs.next()) {
                brandCombo.getItems().add(rs.getString("Brand"));
            }


        } catch (SQLException e) {
            e.printStackTrace();
        }

    }


    // Function that get value from the table
    // and set value of all textfield, textarea,combobox
    // to selected value
    public void getVal() {
        barcodeField.setEditable(false);
        ModelTableProd product = prodTable.getSelectionModel().getSelectedItem();
        prodField.setText(product.getProduct());
        brandCombo.setValue(product.getBrand());
        catCombo.setValue(product.getCategory());
        qtyField.setText(product.getQty());
        priceField.setText(product.getPrice());
        descArea.setText(product.getDesc());
        barcodeField.setText(product.getBarcode());
    }


    // Function that insert the data to database
    public void setDB() {
        try {
            PreparedStatement prepStat = connect.getPrepStat("INSERT INTO product VALUES(?, ?, ?, ?,
?, ?,?)");
            prepStat.setString(1, getProd());
            prepStat.setString(2, getDesc());
            prepStat.setString(3, getCatCombo());
            prepStat.setString(4, getBrandCombo());
            prepStat.setString(5, getQty());
            prepStat.setString(6, getRetailPrice());
            prepStat.setString(7, getBarcode());
            prepStat.executeUpdate();

            Alert alert1 = new Alert(Alert.AlertType.INFORMATION);
            alert1.setTitle("Info");
            alert1.setContentText("Success add new category!");
            alert1.setHeaderText("SUCCESS");
            alert1.show();

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

```
    // Function that check database if the data is already inside database
    // If true, it will show a pop up message
    public boolean dataBaseCheck() {
        PreparedStatement prepStat = connect.getPrepStat("SELECT * FROM product WHERE Barcode='" +
getBarcode() + "'");
        try {
            ResultSet rs = prepStat.executeQuery();
            return rs.next();

        } catch (SQLException e) {
            e.printStackTrace();
            return false;
        }

    }


    // Function that delete product from database
    public void deleteProduct() {
        try {
            // show confirmation of deletion
            Alert alert4 = new Alert(Alert.AlertType.CONFIRMATION);
            alert4.setTitle("Confirmation");
            alert4.setContentText("This will remove it permanently from the database.");
            alert4.setHeaderText("Are you sure want to delete this product?");
            Optional<ButtonType> result = alert4.showAndWait();

            // If user press "OK" button
            if (result.isPresent() && result.get() == ButtonType.OK) {
                PreparedStatement prepStat = connect.getPrepStat("DELETE FROM product WHERE Barcode =
?");
                prepStat.setString(1, getBarcode());
                prepStat.executeUpdate();
                clearField();
                prodTable.getItems().clear();
                showTable();
            } else {
                alert4.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


    // Function that clear of textfield and textarea
    public void clearField() {
        prodField.setText("");
        priceField.setText("");
        qtyField.setText("");
        barcodeField.setText("");
        descArea.setText("");
        catCombo.setValue("");
        brandCombo.setValue("");
    }


    // Initialize function
    @Override
    public void initialize(URL location, ResourceBundle resource) {
        showTable();

        // Create necessary table column
        TableColumn prodCol = new TableColumn("Product");
        prodCol.setMinWidth(100);
        prodCol.setCellValueFactory(
                new PropertyValueFactory<ModelTableProd, String>("product"));

        TableColumn descCol = new TableColumn("Description");
        descCol.setMinWidth(180);
        descCol.setCellValueFactory(
                new PropertyValueFactory<ModelTableProd, String>("desc"));

        TableColumn catCol = new TableColumn("Category");
        catCol.setMinWidth(100);
        catCol.setCellValueFactory(
                new PropertyValueFactory<ModelTableProd, String>("category"));
```

```
        TableColumn brandCol = new TableColumn("Brand");
        brandCol.setMinWidth(100);
        brandCol.setCellValueFactory(
                new PropertyValueFactory<ModelTableProd, String>("brand"));

        TableColumn priceCol = new TableColumn("Price");
        priceCol.setMinWidth(100);
        priceCol.setCellValueFactory(
                new PropertyValueFactory<ModelTableProd, String>("price"));

        TableColumn qtyCol = new TableColumn("Qty");
        qtyCol.setMinWidth(50);
        qtyCol.setCellValueFactory(
                new PropertyValueFactory<ModelTableProd, String>("qty"));

        TableColumn barcodeCol = new TableColumn("Barcode");
        barcodeCol.setMinWidth(100);
        barcodeCol.setCellValueFactory(
                new PropertyValueFactory<ModelTableProd, String>("barcode"));

        prodTable.setItems(oblist);
        prodTable.getColumns().addAll(prodCol, descCol, catCol, brandCol, qtyCol, priceCol,
barcodeCol);
    }
}
```

- Receipt Contoller

  This class made to “print out” a receipt from the cashier tab. It also access the same database table as the CashierController Class.

```
// ReceiptController class which implements Initializable
public class ReceiptController implements Initializable {

    Connection connect = new Connection();

    public TextArea receiptText;

    String temp = "";


    // Function that print all items from database
    // and print it to the "receipt"
    public void showItems() {
        try {
            PreparedStatement prepStat = connect.getPrepStat("SELECT * FROM cashier");
            ResultSet rs = prepStat.executeQuery();
            while (rs.next()) {
                temp += rs.getString("Product") + "\n" + "    Qty: " + rs.getString("Quantity") +
                        " ..................................................................... " +
rs.getString("Total") + "\n";
            }
            receiptText.setText(
                    "-------------- WELCOME TO BINBIN STORE --------------\n" +
                            "                                      YOUR RECEIPT\n" + temp +
                            "\n########################################" +
                            "\n----------------------- THANK YOU -----------------------"
            );

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle resource) {
        showItems();
    }
}
```

- tabPaneController Class

  This class is made to combine 4 FXML files into one FXML. It creates 4 tabs, each tab is accessed one of the FXML file.

```
// tabPaneController class which implements Initializable
public class tabPaneController implements Initializable {

    @FXML
    Tab tab1;

    @FXML
    Tab tab2;

    @FXML
    Tab tab3;

    @FXML
    Tab tab4;


    @Override
    public void initialize(URL url, ResourceBundle resource) {

        try {
            AnchorPane anch1 = FXMLLoader.load(getClass().getResource("Cashier.fxml"));
            tab1.setContent(anch1);
        } catch (IOException ex) {
            System.out.println("File not found!");
        }

        try {
            AnchorPane anch2 = FXMLLoader.load(getClass().getResource("Product.fxml"));
            tab2.setContent(anch2);
        } catch (IOException ex) {
            System.out.println("File not found!");
        }

        try {

            AnchorPane anch3 = FXMLLoader.load(getClass().getResource("Brand.fxml"));
            tab3.setContent(anch3);
        } catch (IOException ex) {
            System.out.println("File not found!");
        }

        try {

            AnchorPane anch4 = FXMLLoader.load(getClass().getResource("Category.fxml"));
            tab4.setContent(anch4);
        } catch (IOException ex) {
            System.out.println("File not found!");
        }
    }
}
```

- LoginController class

  This calss control the GUI of Login.fxml file. It is also extends Connection class for the credentials(username and password)

```
// Login class which extends Connection class
public class LoginController extends Connection {

    public TextField usernameField;
    public PasswordField passwordField;
    public Button loginButton;
    public Label warningLabel;
```

```
    // Function that "loginButton" execute
    public void loginButton() {

        // Check if user input a correct credential
        if (usernameField.getText().equals(getUsername()) &&
passwordField.getText().equals(getPassword())) {
            Parent root = null;
            try {
                root = FXMLLoader.load(getClass().getResource("tabPane.fxml"));
            } catch (IOException e) {
                System.out.println("File not found!");
            }
            assert root != null;
            Scene scene = new Scene(root);
            Stage stage = new Stage();
            stage.setScene(scene);
            stage.show();
            Stage closeWindow = (Stage) loginButton.getScene().getWindow();
            closeWindow.close();
            warningLabel.setVisible(false);

            // If not correct
            // It will show a message to the user
        } else {
            warningLabel.setVisible(true);
        }
    }
}
```

- ModelTableBrand, ModelTableCashier, ModelTableCat, ModelTableProd

  This class basically support the corresponding class for “ObservableList” which is “A list that enables listeners to track changes when they occur”. It also provide the data that will be inserted to the “TableView”. Here is an example from ModelTableProd

```
public class ModelTableProd {
    private String product, desc, category, brand, qty, price, barcode;

    public ModelTableProd(String product, String desc, String category, String brand, String qty,
String price, String barcode) {
        this.product = product;
        this.desc = desc;
        this.category = category;
        this.brand = brand;
        this.qty = qty;
        this.price = price;
        this.barcode = barcode;
    }

    public String getProduct() {
        return product;
    }

    public void setProduct(String product) {
        this.product = product;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    public String getCategory() {
        return category;
    }

    public void setCategory(String category) {
        this.category = category;
    }
```

```
    public String getBrand() {
        return brand;
    }

    public void setBrand(String brand) {
        this.brand = brand;
    }

    public String getQty() {
        return qty;
    }

    public void setQty(String qty) {
        this.qty = qty;
    }

    public String getPrice() {
        return price;
    }

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

    public String getBarcode() {
        return barcode;
    }

    public void setBarcode(String barcode) {
        this.barcode = barcode;
    }
}
```

Evidence

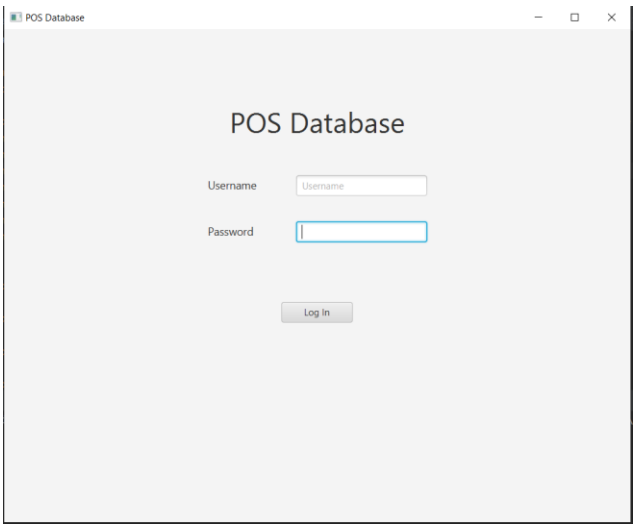


Figure1.1, Login window

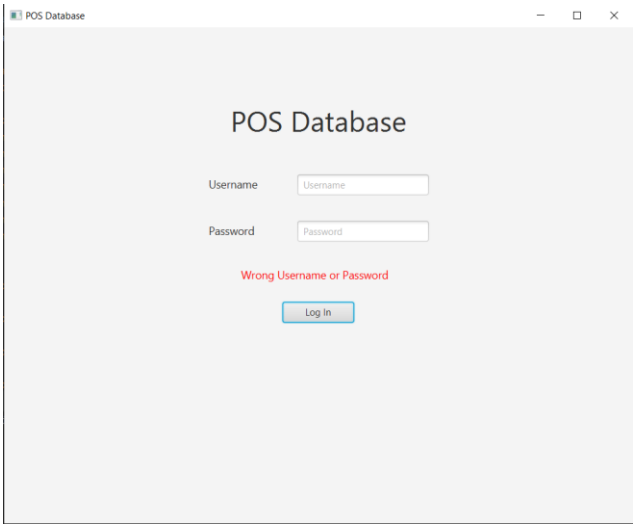


Figure1.2, Login window, when user input wrong username or password

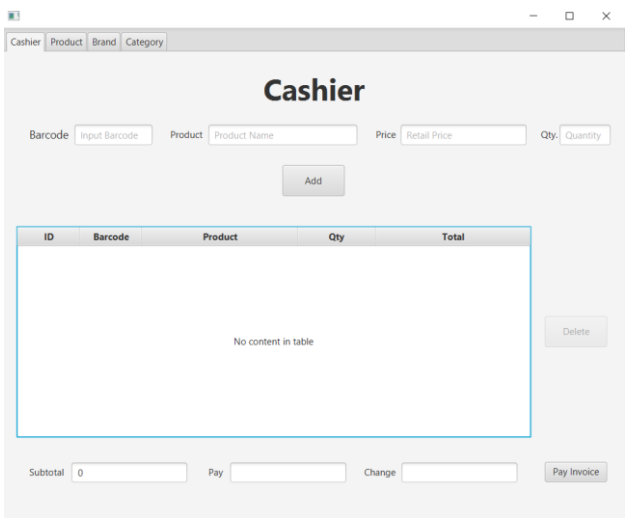


Figure2.1, Cashier Tab

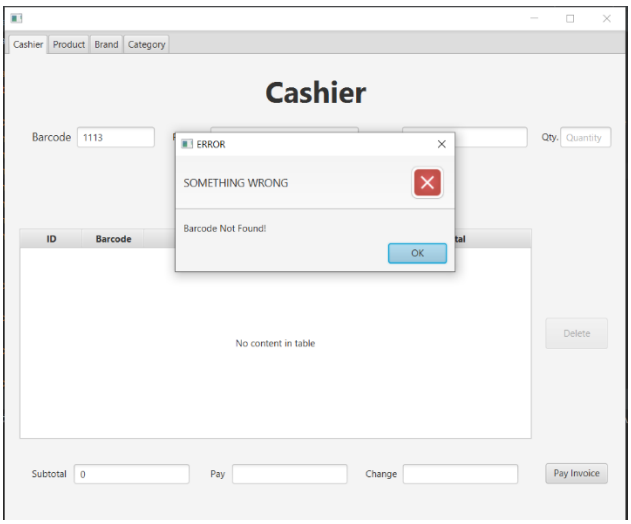


Figure2.2, Cashier Tab, when user input wrong Barcode

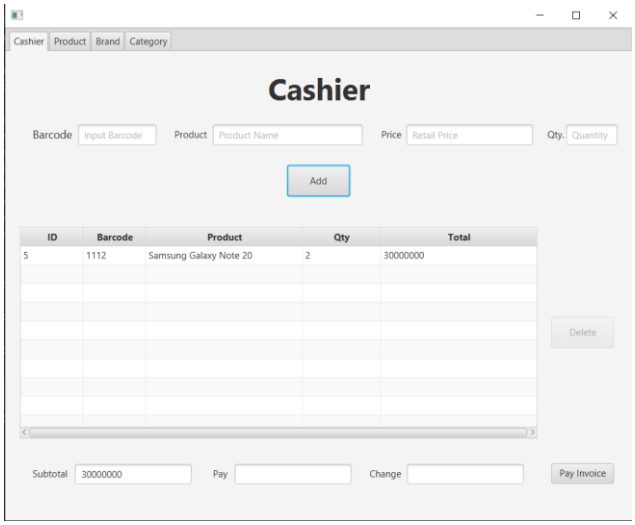


Figure2.3, Cashier Tab, If user input valid barcode and press "Add" button

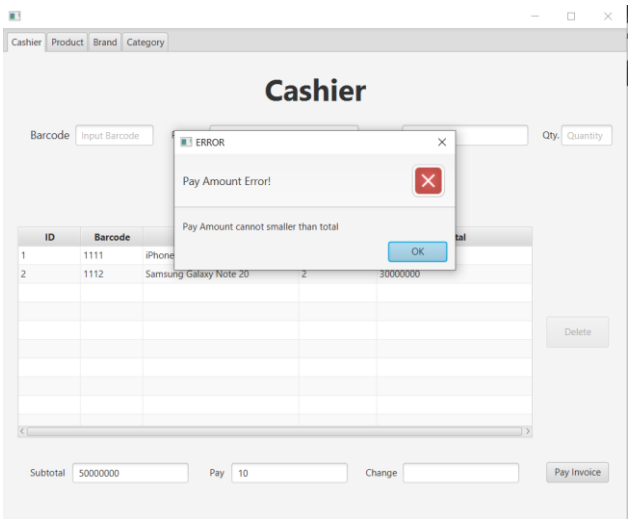


Figure2.4, Cashier Tab, If user input "pay" textfield smaller than "subtotal" textfield

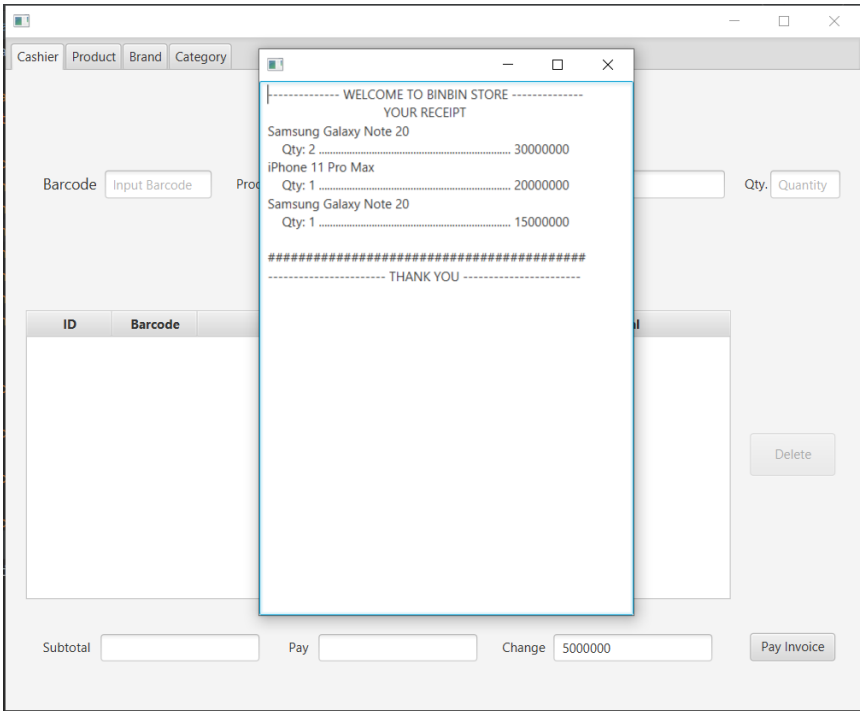


Figure2.5, Cashier Tab, If user press "Pay Invoice" Button, it will pop up a new window with the receipt.

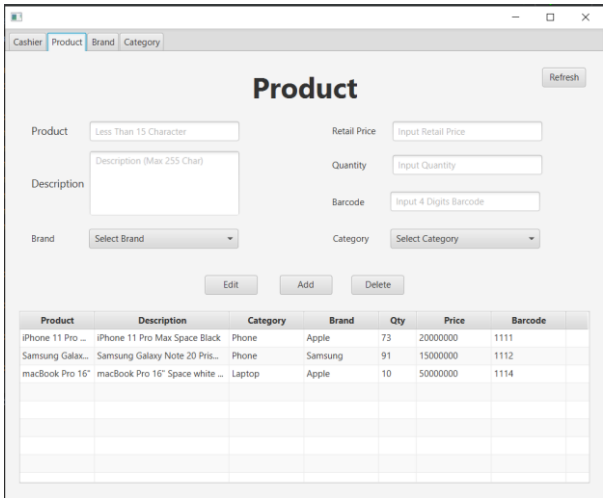


Figure3.1, Product Tab

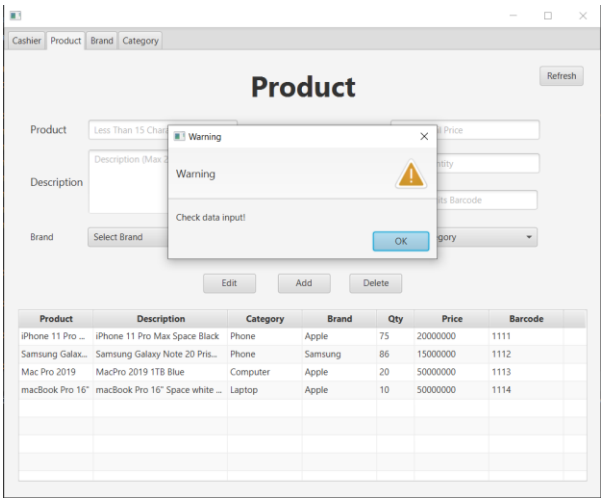


Figure3.2, Product Tab, If user don't fill the data

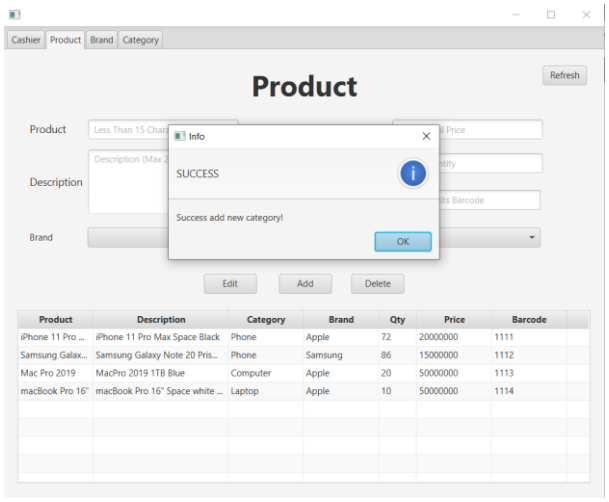


Figure3.3, Product Tab, If user input all valid data

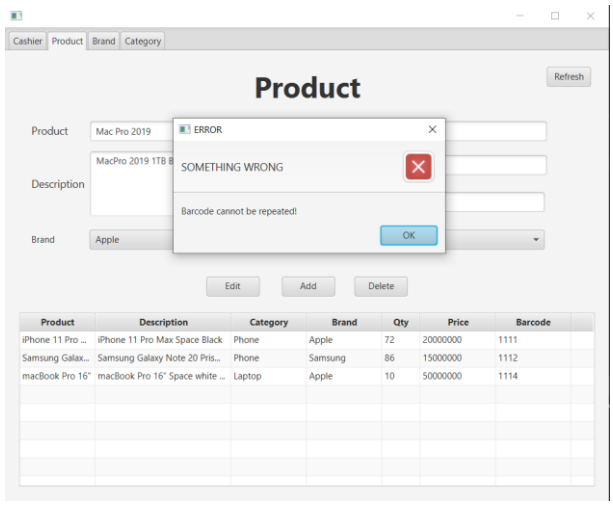


Figure3.4, Product Tab, If user type the same barcode

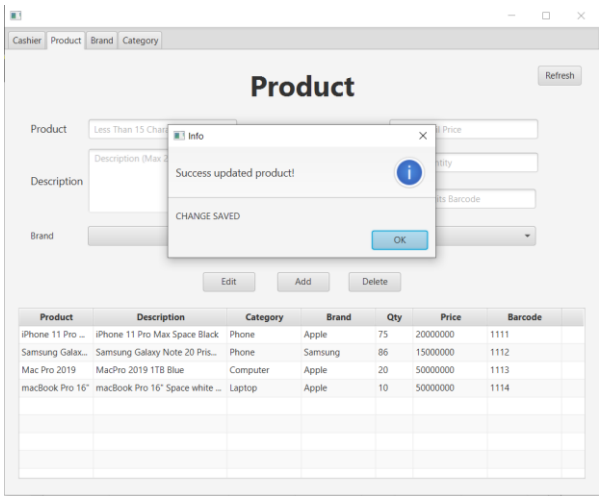


Figure3.5, Product Tab, If user edit a product

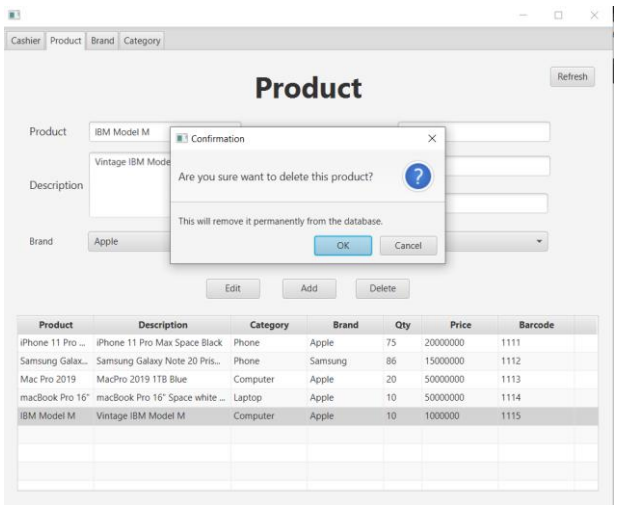


Figure3.6, Product Tab, If user want to delete a product

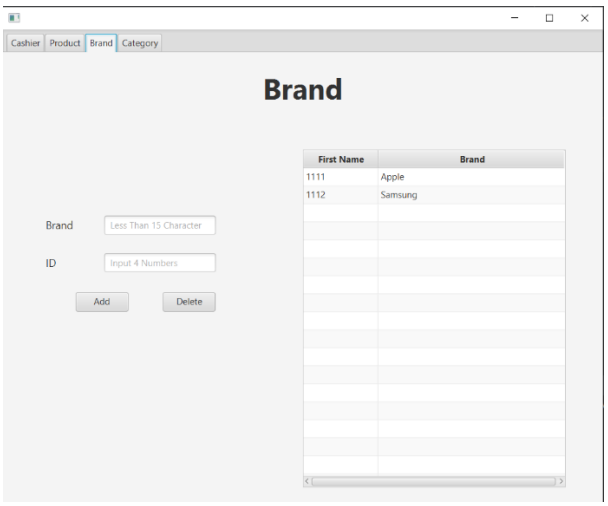


Figure4.1, Brand Tab

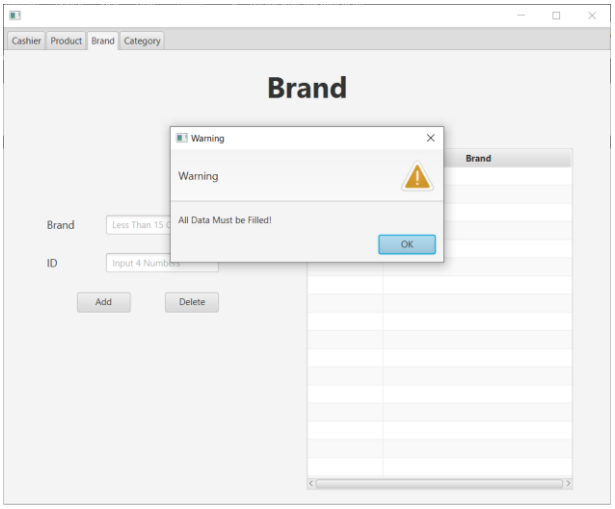


Figure4.2, Brand Tab, If user don't input all data

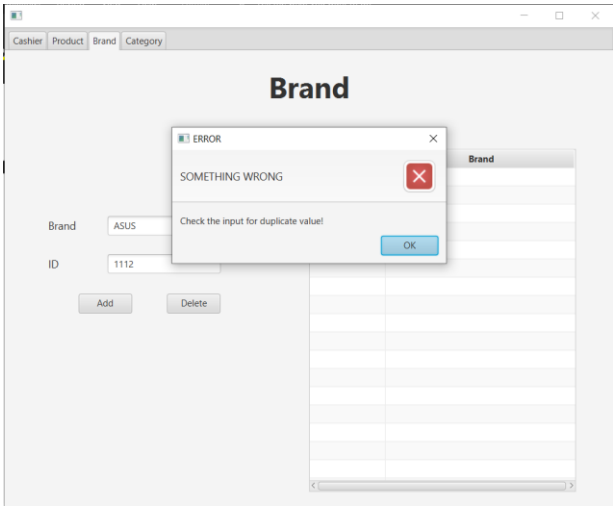


Figure4.3, Brand Tab, If input the same brand and/or ID

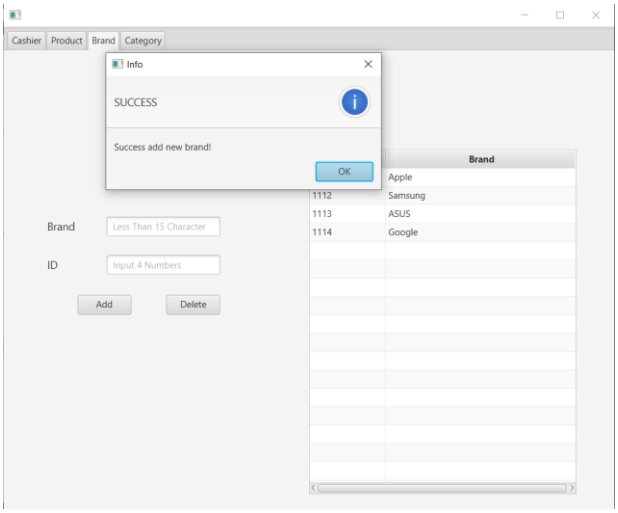


Figure4.3, Brand Tab, If user input all valid data

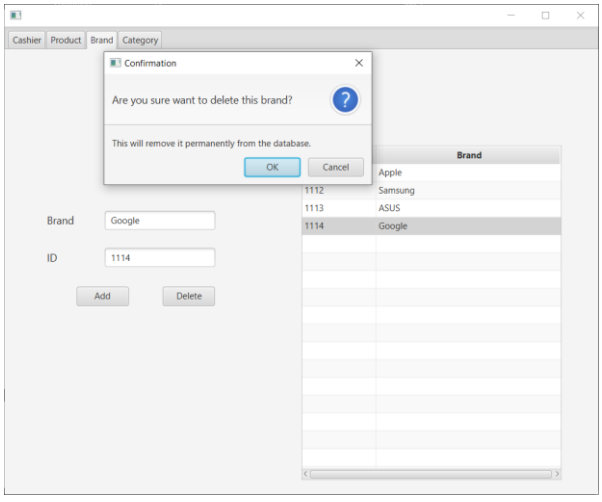


Figure4.4, Brand Tab, If user want to delete a brand

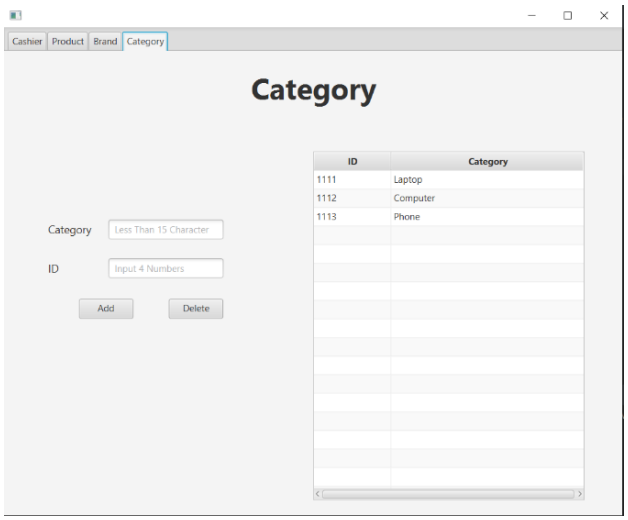


Figure5.1, Category Tab

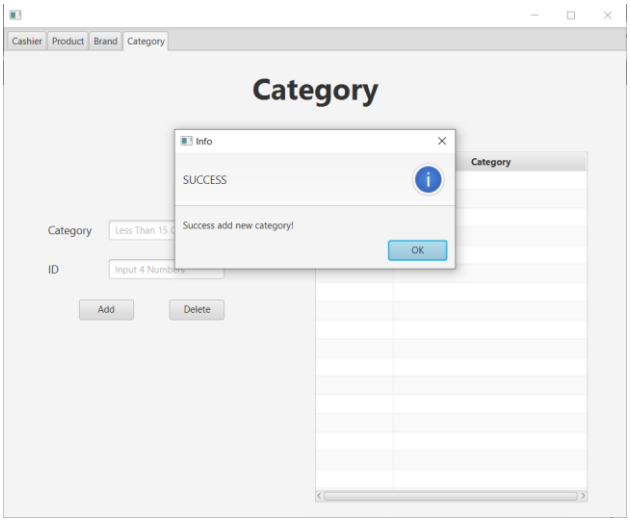


Figure5.2, Category Tab, If user input all valid data

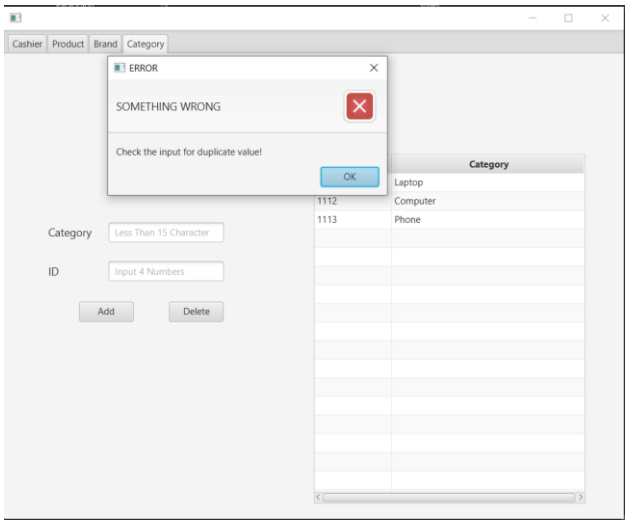


Figure5.3, Category Tab, If user input the same category and/or ID

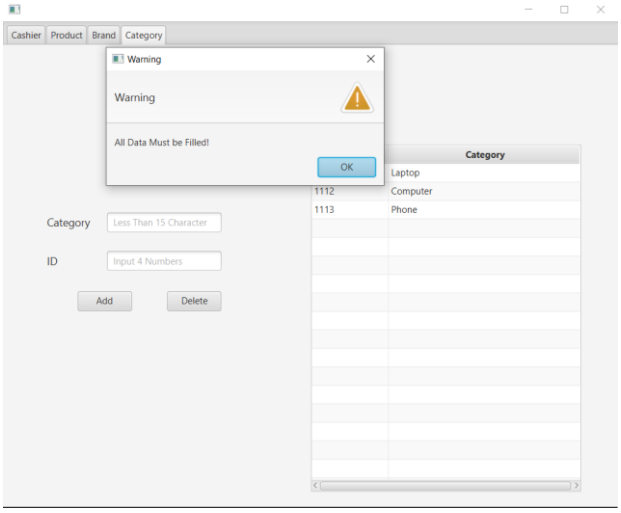


Figure5.4, Category Tab, If user don't fill all data

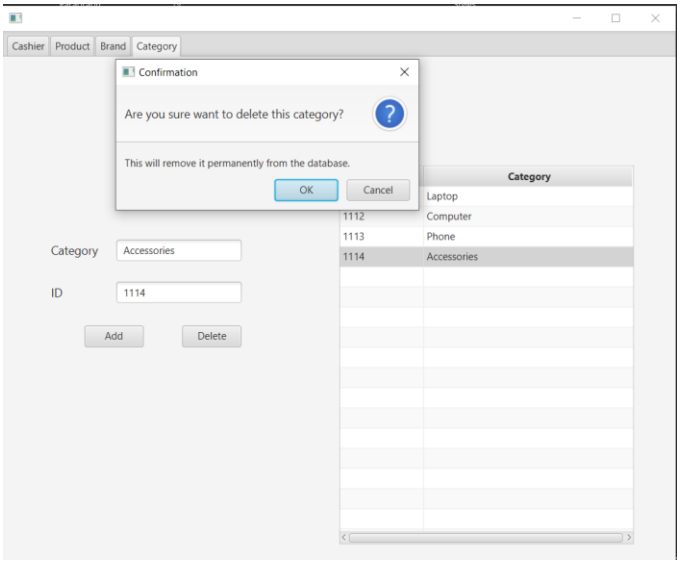


Figure5.5, Category Tab, If user want to delete a category

Link for Demo Video: https://youtu.be/rFESCvfLvdg

Conclusion

To conclude, creating program that use database is not easy. It takes some times for me to understand how to access and use database. I think this program is very useful for small store like minimarket to manage their inventory system and cashier system. I am pretty satisfied with this project because it push me to the limit and use all of the material that I know about java programming.

Reference

- https://youtu.be/2i4t-SL1VsU
- https://www.youtube.com/watch?v=Ksf0383Hhdc&feature=youtu.be&list=PLuji25yj7oIKWUxnb3GeRfql9s5C26CyD
- https://noblecodemonkeys.com/javafx-tab-pane-and-nested-fxml-files/
- https://youtu.be/m_dH8rQ4_Bc