Logistics Management System

A MINI-PROJECT BY:

KRITHIKA B 230701156

KOMMANA SAI LEELA YUKTHA 230701154

in partial fulfillment of the award of the degree

OF

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute

CHENNAI

NOVEMBER 2024

BONAFIDE CERTIFICATE

Certified that this project report "**Logistics Management System"** is a Bonafide work of "**KRITHIKA B (230701156) & KOMMANA SAI LEELA YUKTHA (230701154)" .**

Submitted for the Practical Examination held on ____26.11.2024_____

Signature

Mrs. K MahesMeena

Assistant Professor,

Computer Science and Engineering,

Rajalakshmi Engineering College (Autonomous),

Thandalam, Chennai-602 105

Internal Examiner **External Examiner**

LOGISTICS MANAGESMENT SYSTEM MINI PROJECT

ABSTARCT

Efficient logistics management is essential for businesses to maintain smooth operations and meet customer demands. This mini project presents a comprehensive solution to optimize inventory control, supplier coordination, and order processing. The system leverages real-time stock tracking with automatic low-stock alerts to ensure businesses can prevent stockouts or overstocking, thereby maintaining a balanced inventory.

The project integrates a centralized platform for supplier information, simplifying communication and restocking processes. The order management feature streamlines order placement and fulfillment, linking each order to specific customers and providing accurate tracking of payments and delivery statuses. These functionalities reduce errors and improve overall process efficiency.

Comprehensive inventory logs record all stock changes, including sales, restocking, and adjustments, ensuring transparency and accountability. Additionally, the system provides detailed analytics and reporting, offering insights into sales trends, inventory levels, and critical alerts. This empowers businesses to make informed, data-driven decisions.

By addressing common logistical challenges, this mini project aims to enhance operational efficiency, improve customer satisfaction, and support businesses in staying competitive in dynamic markets.

TABLE OF CONTENTS

INTRODUCTION

1. INTRODUCTION

The project streamlines logistics management by offering businesses a centralized platform for inventory tracking, supplier coordination, and order processing. Users can monitor stock levels in real-time with alerts, ensuring optimal inventory control. Supplier details are centralized for seamless restocking, while orders are accurately tracked with payment and delivery status updates. Comprehensive logs and detailed analytics empower businesses to make informed decisions, improve efficiency, and enhance customer satisfaction.

1.1 IMPLEMENTATION

The **LOGISTICS MANAGEMENT** project discussed here is implemented using the concepts of **JAVA SWINGS** and **MYSQL.**

1.2 SCOPE OF THE PROJECT

The logistics management system is designed for businesses to register and create accounts, allowing them to manage all their inventory and order data in one organized platform. This ensures streamlined operations, saves time, and promotes a sense of professionalism.

1.3 WEBSITE FEATURES

- Registering and login page.
- Income data chart .
- Dashboard showing possible actions.
- Overview of all the products **.**

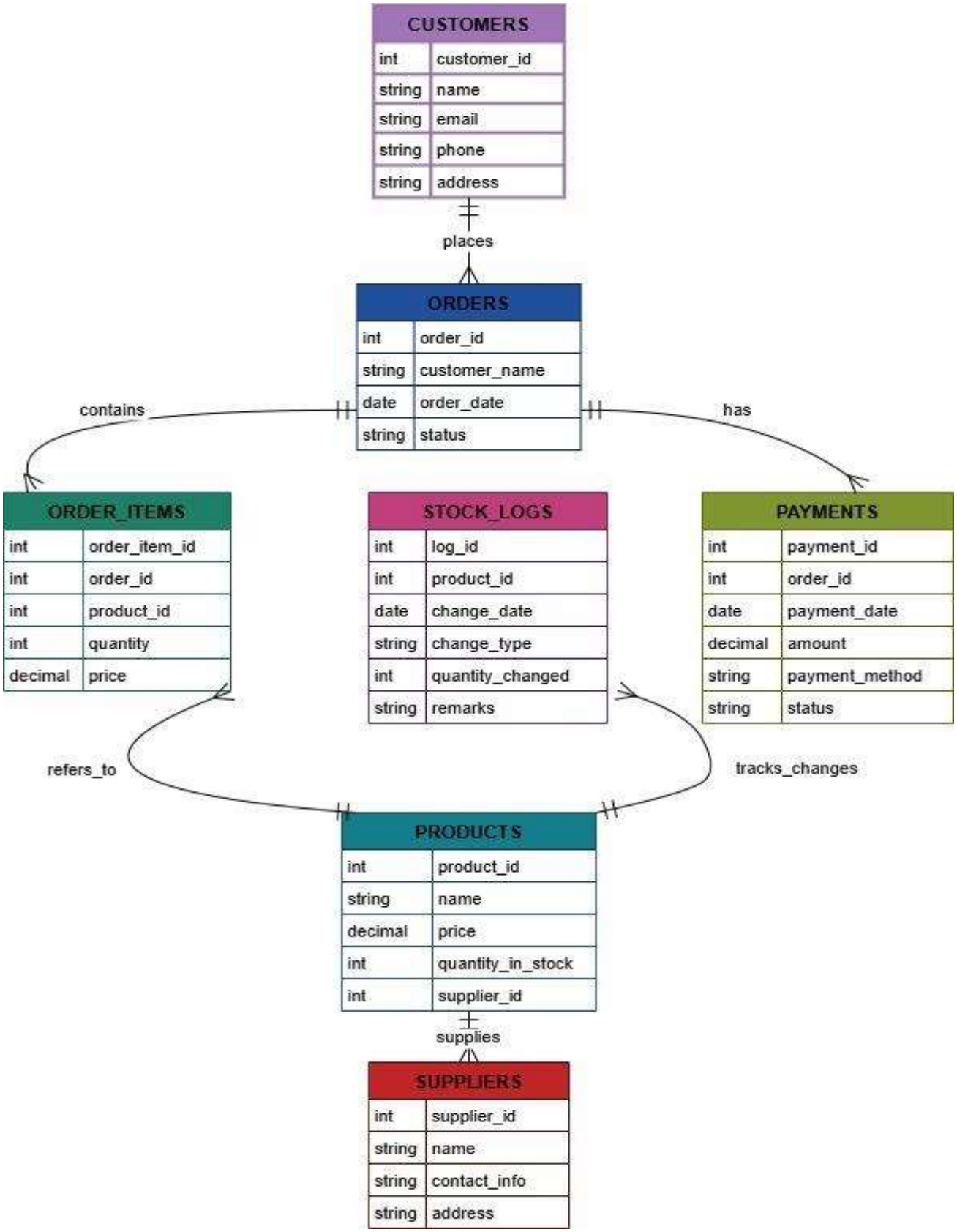
SYSTEM SPECIFICATIONS

2.1 HARDWARE SPECIFICATIONS:

PROCESSOR	:	Intel i5
MEMORY SIZE	:	4GB(Minimum)
HARD DISK	:	500 GB of free space

2.2 SOFTWARE SPECIFICATIONS:

PROGRAMMING LANGUAGE	:	Java, MySQL
FRONT-END	:	Java
BACK-END	:	MySQL
OPERATING SYSTEM	:	Windows 11


CUSTOMERS
int customer_id
string name
string email
string phone
string address
places
ORDERS
int order_id
string customer_name
date order_date
string status
contains
has
ORDER_ITEMS
int order_item_id
int order_id
int product_id
int quantity
decimal price
STOCK_LOGS
int log_id
int product_id
date change_date
string change_type
int quantity_changed
string remarks
PAYMENTS
int payment_id
int order_id
date payment_date
decimal amount
string payment_method
string status
refers_to
tracks_changes
PRODUCTS
int product_id
string name
decimal price
int quantity_in_stock
int supplier_id
supplies
SUPPLIERS
int supplier_id
string name
string contact_info
string address

SAMPLE CODE

4.1 LOGIN PAGE

```
3   package com.mycompany.yt_inventorymanagementsystem_javafx;
4
5   import java.net.URL;
6   import java.sql.Connection;
7   import java.sql.PreparedStatement;
8   import java.sql.ResultSet;
9   import java.util.ResourceBundle;
10  import javafx.fxml.FXML;
11  import javafx.fxml.FXMLLoader;
12  import javafx.fxml.Initializable;
13  import javafx.scene.Parent;
14  import javafx.scene.Scene;
15  import javafx.scene.control.Alert;
16  import javafx.scene.control.Button;
17  import javafx.scene.control.PasswordField;
18  import javafx.scene.control.TextField;
19  import javafx.scene.input.MouseEvent;
20  import javafx.scene.layout.AnchorPane;
21  import javafx.stage.Stage;
22  import javafx.stage.StageStyle;
23
24  /**
25   * FXML Controller class
26   *
27   * @author USER
28   */
29  public class LoginController implements Initializable {
30
31      @FXML
32      private Button close_btn;
33
34      @FXML
35      private Button login_btn;
36
37      @FXML
38      private AnchorPane main_form;
39
40      @FXML
41      private PasswordField password_txt;
42
43      @FXML
44      private TextField username_txt;
45
46      // DATABASE TOOLS
47      private Connection con;
48      private PreparedStatement prepare;
49      private ResultSet result;
50      private double x = 0;
51      private double y = 0;
52
53      /**
54       * Initializes the controller class.
55       */
56      @Override
57      public void initialize(URL url, ResourceBundle rb) {
```

```
60
61      // LOGIN METHOD
62      public void loginAdmin() {
63          String sql = "select * from users where username = ? and password = ?";
64
65          // Try to establish a connection
66          con = databaseHandler.connectDb();
67
68          if (con == null) {
69              showAlert(Alert.AlertType.ERROR, "Database Connection Error", "Failed to connect to the
    database.");
70              return;
71          }
72
73          try {
74              prepare = con.prepareStatement(sql);
75              prepare.setString(1, username_txt.getText());
76              prepare.setString(2, password_txt.getText());
77
78              result = prepare.executeQuery();
79
80              if (username_txt.getText().isEmpty() || password_txt.getText().isEmpty()) {
81                  showAlert(Alert.AlertType.ERROR, "Input Error", "Please fill all blank fields.");
82              } else {
83                  if (result.next()) {
84                      // If login is successful, store username for later use
85                      GetData.username = username_txt.getText();
86
87                      // Show success alert
88                      showAlert(Alert.AlertType.INFORMATION, "Login Success", "Successfully logged
    in!");
89
90                      // Hide the login window
91                      login_btn.getScene().getWindow().hide();
92
93                      // Load the dashboard scene
94                      loadDashboard();
95                  } else {
96                      // If username/password is wrong
97                      showAlert(Alert.AlertType.ERROR, "Login Failed", "Wrong Username or Password.");
98                  }
99              }
100         } catch (Exception e) {
101             e.printStackTrace();
102             showAlert(Alert.AlertType.ERROR, "Error", "An error occurred: " + e.getMessage());
103         }
104     }
105
106     // Helper method to show alerts
107     private void showAlert(Alert.AlertType type, String title, String content) {
108         Alert alert = new Alert(type);
109         alert.setTitle(title);
110         alert.setHeaderText(null);
111         alert.setContentText(content);
112         alert.showAndWait();
113     }
114
115     // Load the dashboard
116     private void loadDashboard() {
117         try {
```

```
121
122             // Drag functionality for moving the window
123             root.setOnMousePressed((MouseEvent event) -> {
124                 x = event.getSceneX();
125                 y = event.getSceneY();
126             });
127
128             root.setOnMouseDragged((MouseEvent event) -> {
129                 stage.setX(event.getScreenX() - x);
130                 stage.setY(event.getScreenY() - y);
131             });
132
133             stage.initStyle(StageStyle.TRANSPARENT);
134             stage.setScene(scene);
135             stage.show();
136         } catch (Exception e) {
137             e.printStackTrace();
138             showAlert(Alert.AlertType.ERROR, "Error", "Unable to load the dashboard: " +
    e.getMessage());
139         }
140     }
141
142     // Close the application
143     public void close() {
144         System.exit(0);
145     }
146 }
```

4.2 Dashboard

```
5  package com.mycompany.yt_inventorymanagementsystem_javafx;
6
7  import de.jensd.fx.glyphs.fontawesome.FontAwesomeIconView;
8  import java.io.File;
9  import java.net.URL;
10 import java.sql.Connection;
11 import java.sql.PreparedStatement;
12 import java.sql.ResultSet;
13 import java.sql.Statement;
14 import java.util.ArrayList;
15 import java.util.Date;
16 import java.util.HashMap;
17 import java.util.List;
18 import java.util.Optional;
19 import java.util.ResourceBundle;
20 import javafx.collections.FXCollections;
21 import javafx.collections.ObservableList;
22 import javafx.collections.transformation.FilteredList;
23 import javafx.collections.transformation.SortedList;
24 import javafx.event.ActionEvent;
25 import javafx.fxml.FXML;
26 import javafx.fxml.FXMLLoader;
27 import javafx.fxml.Initializable;
28 import javafx.scene.Parent;
29 import javafx.scene.Scene;
30 import javafx.scene.chart.AreaChart;
31 import javafx.scene.chart.BarChart;
32 import javafx.scene.chart.XYChart;
33 import javafx.scene.control.Alert;
34 import javafx.scene.control.Button;
35 import javafx.scene.control.ButtonType;
36 import javafx.scene.control.ComboBox;
```

```
39 import javafx.scene.control.SpinnerValueFactory;
40 import javafx.scene.control.TableColumn;
41 import javafx.scene.control.TableView;
42 import javafx.scene.control.TextField;
43 import javafx.scene.control.cell.PropertyValueFactory;
44 import javafx.scene.image.Image;
45 import javafx.scene.image.ImageView;
46 import javafx.scene.input.MouseEvent;
47 import javafx.scene.layout.AnchorPane;
48 import javafx.stage.FileChooser;
49 import javafx.stage.Stage;
50 import javafx.stage.StageStyle;
51 import net.sf.jasperreports.engine.JRException;
52 import net.sf.jasperreports.engine.JasperCompileManager;
53 import net.sf.jasperreports.engine.JasperFillManager;
54 import net.sf.jasperreports.engine.JasperPrint;
55 import net.sf.jasperreports.engine.JasperReport;
56 import net.sf.jasperreports.engine.design.JasperDesign;
57 import net.sf.jasperreports.engine.xml.JRXmlLoader;
58 import net.sf.jasperreports.view.JasperViewer;
59 

60 /**
61  * FXML Controller class
62  *
63  * @author USER
64  */
65 public class DashboardController implements Initializable {
66 
67     /**
68      * Initializes the controller class.
69      */
70     @FXML
71     private Button addProduct_add_btn;
72 
73     @FXML
74     private TextField addProduct_brand_txt;
75 
76     @FXML
77     private FontAwesomeIconView addProduct_btn;
78 
79     @FXML
80     private Button addProduct_delete_btn;
81 
82     @FXML
83     private ImageView addProduct_imgView;
84 
85     @FXML
86     private Button addProduct_imort_btn;
87 
88     @FXML
89     private TextField addProduct_price_txt;
90 
91     @FXML
92     private TextField addProduct_productId_txt;
93 
94     @FXML
95     private TextField addProduct_productName_txt;
96 
97     @FXML
```

```
101     private Button addProduct_reset_btn;
102
103     @FXML
104     private TextField addProduct_search_txt;
105
106     @FXML
107     private ComboBox<?> addProduct_status_combo;
108
109     @FXML
110     private TableView<ProductsModel> addProduct_tableView;
111
112     @FXML
113     private TableColumn<ProductsModel, String> addProduct_tbvCol_brand;
114
115     @FXML
116     private TableColumn<ProductsModel, String> addProduct_tbvCol_price;
117
118     @FXML
119     private TableColumn<ProductsModel, String> addProduct_tbvCol_productID;
120
121     @FXML
122     private TableColumn<ProductsModel, String> addProduct_tbvCol_productName;
123
124     @FXML
125     private TableColumn<ProductsModel, String> addProduct_tbvCol_status;
126
127     @FXML
128     private TableColumn<ProductsModel, String> addProduct_tbvCol_type;
129
130     @FXML
131     private Button addProduct_update_btn;
132
133     @FXML
134     private Button addProducts_btn;
135
136     @FXML
137     private AnchorPane addProducts_form;
138
139     @FXML
140     private Button close_btn;
141
142     @FXML
143     private Label home_availableProducts_lbl;
144
145     @FXML
146     private Button home_btn;
147
148     @FXML
149     private AnchorPane home_form;
150
151     @FXML
152     private AreaChart<?, ?> home_incomeChart;
153
154     @FXML
155     private Label home_numberOfOrders_lbl;
156
157     @FXML
158     private BarChart<?, ?> home_orderChart;
159
160     @FXML
```

```
164     private Button logout;
165
166     @FXML
167     private AnchorPane main_form;
168
169     @FXML
170     private FontAwesomeIconView minize_btn;
171
172     @FXML
173     private Button orders_add_btn;
174
175     @FXML
176     private TextField orders_amount_txt;
177
178     @FXML
179     private Label orders_balance_lbl;
180
181     @FXML
182     private ComboBox<?> orders_brandName_combo;
183
184     @FXML
185     private Button orders_btn;
186
187     @FXML
188     private TableColumn<CustomerData, String> orders_col_brand;
189
190     @FXML
191     private TableColumn<CustomerData, String> orders_col_price;
192
193     @FXML
194     private TableColumn<CustomerData, String> orders_col_productName;
195
196     @FXML
197     private TableColumn<CustomerData, String> orders_col_quantity;
198
199     @FXML
200     private TableColumn<CustomerData, String> orders_col_type;
201
202     @FXML
203     private AnchorPane orders_form;
204
205     @FXML
206     private Button orders_pay_btn;
207
208     @FXML
209     private ComboBox<?> orders_productName_combo;
210
211     @FXML
212     private ComboBox<?> orders_productType_combo;
213
214     @FXML
215     private Spinner<Integer> orders_quantity_spinner;
216
217     @FXML
218     private Button orders_receipt_btn;
219
220     @FXML
221     private Button orders_reset_btn;
222
223     @FXML
```

```
227     private Label orders_total_lbl;
228 
229     @FXML
230     private Label username_lbl;
231 
232     // DATABASE TOOLS
233     private Connection conn;
234     private PreparedStatement prepare;
235     private Statement statement;
236     private ResultSet resultSet;
237 
238 
239     public void homeDisplayTotalOrders(){
240 //
241 //        Date date = new Date();
242 //                    java.sql.Date sqlDate = new java.sql.Date(date.getTime());
243 //                    prepare.setString(8, String.valueOf(sqlDate));
244 
245         Date date = new Date();
246         java.sql.Date sqlDate = new java.sql.Date(date.getTime());
247 
248 //        String sql = "SELECT COUNT(id) AS count_of_id FROM customer_receipt";
249         String sql = "SELECT COUNT(id) FROM customer_receipt WHERE date = '"+sqlDate+"' ";
250 
251         conn = databaseHandler.connectDb();
252 
253         int countOrders = 0;
254 
255         try {
256 
257             prepare = conn.prepareStatement(sql);
258             resultSet = prepare.executeQuery();
259 
260             if(resultSet.next()){
261 //                countOrders = resultSet.getInt("count_of_id");
262                 countOrders = resultSet.getInt("COUNT(id)");
263             }
264 
265             home_numberOfOrders_lbl.setText(String.valueOf(countOrders));
266 
267         } catch (Exception e) {e.printStackTrace();}
268 
269     }
270 
271     public void homeTotalIncome(){
272 
273         String sql = "SELECT SUM(total) AS total_income FROM customer_receipt";
274 
275         conn = databaseHandler.connectDb();
276 
277         double totalIncome = 0;
278 
279         try {
280 
281             prepare = conn.prepareStatement(sql);
282             resultSet = prepare.executeQuery();
283 
284             if(resultSet.next()){
285                 totalIncome = resultSet.getDouble("total_income");
```

```
289
290         } catch (Exception e) {e.printStackTrace();}
291
292     }
293
294     public void homeAvailableProducts(){
295
296         String sql = "SELECT COUNT(id) AS ava_products FROM product WHERE status = 'Available'";
297
298         conn = databaseHandler.connectDb();
299
300         int availableProducts = 0;
301
302         try {
303
304             prepare = conn.prepareStatement(sql);
305             resultSet = prepare.executeQuery();
306
307             if(resultSet.next()){
308                 availableProducts = resultSet.getInt("ava_products");
309             }
310
311             home_availableProducts_lbl.setText(String.valueOf(availableProducts));
312
313         } catch (Exception e) {e.printStackTrace();}
314
315     }
316
317     public void homeIncomeChart(){
318
319         home_incomeChart.getData().clear();
320
321 //        String sql = "SELECT date, SUM(total) FROM customer_receipt GROUP BY date GROUP BY
    TIMESTAMP(date) ASC LIMIT 6";
322
323         String sql = "SELECT date, SUM(total) FROM customer_receipt GROUP BY date ORDER BY
    TIMESTAMP(date) ASC LIMIT 6;";
324
325         conn = databaseHandler.connectDb();
326
327         try {
328
329             XYChart.Series chart = new XYChart.Series();
330
331             prepare = conn.prepareStatement(sql);
332             resultSet = prepare.executeQuery();
333
334             while(resultSet.next()){
335                 chart.getData().add(new XYChart.Data(resultSet.getString(1), resultSet.getInt(2)));
336             }
337
338             home_incomeChart.getData().add(chart);
339
340         } catch (Exception e) {e.printStackTrace();}
341
342     }
343
344     public void homeOrdersChart(){
345
346         home_orderChart.getData().clear();
```

```
349
350         /*
351         SELECT date, SUM(total): We ask for two things: the date and the sum of income for each date.
    The SUM(total) adds up all income values for each date, so we get the total income on that day.
352         FROM customer_receipt: This tells the database to look in the customer_receipt table.
353         GROUP BY date: It groups the data by date, so it only shows one total amount per date.
354         ORDER BY TIMESTAMP(date) ASC: Orders the data by date from oldest to newest.
355         LIMIT 6: Only shows the last 6 days (or dates) of data.
356         */
357
358         conn = databaseHandler.connectDb();
359
360         try {
361
362             XYChart.Series chart = new XYChart.Series<>(); // This creates an empty series, which is
    like a container that will hold all the data points for the chart.
363
364             prepare = conn.prepareStatement(sql);
365             resultSet = prepare.executeQuery();
366
367             while(resultSet.next()){
368                 chart.getData().add(new XYChart.Data(resultSet.getString(1), resultSet.getInt(2)));
369
370                 /*
371                 resultSet.next(): Moves to the next row of data.
372                 resultSet.getString(1): Gets the date from the first column.
373                 resultSet.getInt(2): Gets the total income for that date from the second column.
374                 */
375
376             }
377
378             home_orderChart.getData().add(chart);
379
380         } catch (Exception e) {e.printStackTrace();}
381
382     }
383
384     public void addProductsAdd() {
385
386         String sql = "INSERT INTO product (product_id, type, brand, product_name, price, status, image,
    date) "
387                 + "VALUES(?,?,?,?,?,?,?,?)";
388
389         conn = databaseHandler.connectDb();
390
391         try {
392
393             Alert alert;
394
395             if (addProduct_productId_txt.getText().isEmpty()
396                     || addProduct_productType_combo.getSelectionModel().getSelectedItem() == null
397                     || addProduct_brand_txt.getText().isEmpty()
398                     || addProduct_productName_txt.getText().isEmpty()
399                     || addProduct_price_txt.getText().isEmpty()
400                     || addProduct_status_combo.getSelectionModel().getSelectedItem() == null
401                     || GetData.path == "") {
402
403                 alert = new Alert(Alert.AlertType.ERROR);
404                 alert.setTitle("Error Message");
405                 alert.setHeaderText(null);
```

```
409             } else {
410 
411                 // CHECK IF THE PRODUCT ID IS ALREADY EXIST
412                 String checkData = "SELECT product_id FROM product WHERE product_id = '"
413                         + addProduct_productId_txt.getText() + "'";
414 
415                 statement = conn.createStatement();
416                 resultSet = statement.executeQuery(checkData);
417 
418                 if (resultSet.next()) {
419 
420                     alert = new Alert(Alert.AlertType.ERROR);
421                     alert.setTitle("Error Message");
422                     alert.setHeaderText(null);
423                     alert.setContentText("Product ID: " + addProduct_productId_txt.getText() + " was
    already exist!");
424                     alert.showAndWait();
425                 } else {
426 
427                     prepare = conn.prepareStatement(sql);
428                     prepare.setString(1, addProduct_productId_txt.getText());
429                     prepare.setString(2, (String)
    addProduct_productType_combo.getSelectionModel().getSelectedItem());
430                     prepare.setString(3, addProduct_brand_txt.getText());
431                     prepare.setString(4, addProduct_productName_txt.getText());
432                     prepare.setString(5, addProduct_price_txt.getText());
433                     prepare.setString(6, (String)
    addProduct_status_combo.getSelectionModel().getSelectedItem());
434 
435                     String uri = GetData.path;
436                     uri = uri.replace("\\", "\\\\");
437                     prepare.setString(7, uri);
438 
439                     Date date = new Date();
440                     java.sql.Date sqlDate = new java.sql.Date(date.getTime());
441                     prepare.setString(8, String.valueOf(sqlDate));
442 
443                     prepare.executeUpdate();
444 
445                     /*
446                     Difference Between executeUpdate() and executeQuery():
447                     executeUpdate():
448 
449                     This method is used for modifying the database. It is typically used with SQL
    statements such as INSERT, UPDATE, DELETE, and CREATE. These types of statements alter the contents or
    structure of the database but do not return data in the form of a result set.
450                     It returns an integer representing the number of rows affected by the operation.
451                     executeQuery():
452 
453                     This method is used for retrieving data from the database. It is typically used
    with SELECT statements, where the query returns a result set (data retrieved from the database).
454                     It returns a ResultSet object, which contains the data retrieved by the query.
455                      */
456                     // TO BECOME UPDATED YOUR TABLEVIEW
457                     addProductsShowListData();
458 
459                     // CLEAR THE FIELDS
460                     addProductsReset();
461 
462                 }
```

```
466             e.printStackTrace();
467         }
468 
469     }
470 
471     private String[] listType = {"Snacks", "Drinks", "Dessert", "Gadgets", "Personal Product",
    "Others"};
472 
473     public void addProductsListTypes() {
474 
475         List<String> listT = new ArrayList<>();
476 
477         for (String data : listType) {
478             listT.add(data);
479         }
480 
481         ObservableList listData = FXCollections.observableArrayList(listT);
482         addProduct_productType_combo.setItems(listData);
483 
484     }
485 
486     private String[] listStatus = {"Available", "Not Available"};
487 
488     public void addProductsListStatus() {
489         List<String> listS = new ArrayList<>();
490 
491         for (String data : listStatus) {
492             listS.add(data);
493         }
494 
495         ObservableList listData = FXCollections.observableArrayList(listS);
496         addProduct_status_combo.setItems(listData);
497     }
498 
499     public void addproductsSelectedItemsToInputFields() {
500 
501         ProductsModel productsData = addProduct_tableView.getSelectionModel().getSelectedItem(); //
    TableView<ProductsModel> addProduct_tableView; nisa return wenneth ProductsModel object ekak
502         int num = addProduct_tableView.getSelectionModel().getSelectedIndex();
503 
504         if ((num - 1) < -1) {
505             return;
506         } // This condition checks if the selected index (num) is invalid. If no row is selected (i.e.,
    num is -1), the method returns and exits without doing anything.The num - 1 < -1 check prevents trying
    to access an invalid row.
507 
508         addProduct_productId_txt.setText(String.valueOf(productsData.getProductId())); //
    addProductsShowListData() Read this method to understand how the variable of the ProductsModel.java
    class matched to the column of the table
509 //        addProduct_productType_combo.getSelectionModel().clearSelection();
510         addProduct_brand_txt.setText(productsData.getBrand());
511         addProduct_productName_txt.setText(productsData.getProductName());
512         addProduct_price_txt.setText(String.valueOf(productsData.getPrice()));
513 //        addProduct_status_combo.getSelectionModel().clearSelection();
514 
515         String uri = "file:" + productsData.getImage();
516 
517         image = new Image(uri, 115, 128, false, true);
518         addProduct_imgView.setImage(image);
519 
```

```
    the path is empty, by clicking a row in the table, the path of the image can be set to the path of
    GetData.java.
521     }
522 
523     public void addProductsUpdate() {
524 
525         String uri = GetData.path;
526         uri = uri.replace("\\", "\\\\");
527 
528         Date date = new Date();
529         java.sql.Date sqlDate = new java.sql.Date(date.getTime());
530 
531         String sql = "UPDATE product SET type = '"
532                 + addProduct_productType_combo.getSelectionModel().getSelectedItem()
533                 + "', brand = '" + addProduct_brand_txt.getText()
534                 + "', product_name = '" + addProduct_productName_txt.getText()
535                 + "', price = '" + addProduct_price_txt.getText()
536                 + "', status = '" + addProduct_status_combo.getSelectionModel().getSelectedItem()
537                 + "', image = '" + uri
538                 + "', date = '" + sqlDate
539                 + "' WHERE product_id = '" + addProduct_productId_txt.getText() + "'";
540 
541         conn = databaseHandler.connectDb();
542 
543         try {
544 
545             Alert alert;
546 
547             if (addProduct_productId_txt.getText().isEmpty()
548                     || addProduct_productType_combo.getSelectionModel().getSelectedItem() == null
549                     || addProduct_brand_txt.getText().isEmpty()
550                     || addProduct_productName_txt.getText().isEmpty()
551                     || addProduct_price_txt.getText().isEmpty()
552                     || addProduct_status_combo.getSelectionModel().getSelectedItem() == null
553                     || GetData.path == "") {
554 
555                 alert = new Alert(Alert.AlertType.ERROR);
556                 alert.setTitle("Error Message");
557                 alert.setHeaderText(null);
558                 alert.setContentText("Please fill all blank fields");
559                 alert.showAndWait();
560 
561             } else {
562 
563                 alert = new Alert(Alert.AlertType.CONFIRMATION);
564                 alert.setTitle("Confirmation Message");
565                 alert.setHeaderText(null);
566                 alert.setContentText("Are you sure you want to UPDATE Product ID: " +
    addProduct_productId_txt.getText() + " ?");
567 
568                 Optional<ButtonType> option = alert.showAndWait();
569 
570                 if (option.get().equals(ButtonType.OK)) {
571 
572                     statement = conn.createStatement();
573                     statement.executeUpdate(sql);
574 
575                     alert = new Alert(Alert.AlertType.INFORMATION);
576                     alert.setTitle("Information Message");
577                     alert.setHeaderText(null);
```

```
                    // TO BECOME UPDATED YOUR TABLEVIEW
                    addProductsShowListData();

                    // CLEAR THE FIELDS
                    addProductsReset();
                }
            }

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

    public void addProductsDelete() {

        String sql = "DELETE FROM product WHERE product_id = '" + addProduct_productId_txt.getText() + "'";
        conn = databaseHandler.connectDb();

        try {

            Alert alert;

            if (addProduct_productId_txt.getText().isEmpty()
                    || addProduct_productType_combo.getSelectionModel().getSelectedItem() == null
                    || addProduct_brand_txt.getText().isEmpty()
                    || addProduct_productName_txt.getText().isEmpty()
                    || addProduct_price_txt.getText().isEmpty()
                    || addProduct_status_combo.getSelectionModel().getSelectedItem() == null
                    || GetData.path == "") {

                alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("Please fill all blank fields");
                alert.showAndWait();

            } else {

                alert = new Alert(Alert.AlertType.CONFIRMATION);
                alert.setTitle("Confirmation Message");
                alert.setHeaderText(null);
                alert.setContentText("Are you sure you want to DELETE Product ID: " + addProduct_productId_txt.getText() + " ?");

                Optional<ButtonType> option = alert.showAndWait();

                if (option.get().equals(ButtonType.OK)) {

                    statement = conn.createStatement();
                    statement.executeUpdate(sql);

                    alert = new Alert(Alert.AlertType.INFORMATION);
                    alert.setTitle("Information Message");
                    alert.setHeaderText(null);
                    alert.setContentText("Successfully Deleted");
                    alert.showAndWait();

                    // TO BECOME UPDATED YOUR TABLEVIEW
                    addProductsShowListData();
```

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

    public void addProductsReset() {

        addProduct_productId_txt.setText("");
        addProduct_productType_combo.getSelectionModel().clearSelection();
        addProduct_brand_txt.setText("");
        addProduct_productName_txt.setText("");
        addProduct_price_txt.setText("");
        addProduct_status_combo.getSelectionModel().clearSelection();
        addProduct_imgView.setImage(null);
        GetData.path = "";

    }

    private Image image;

    public void addProductsImportImage() {

        FileChooser open = new FileChooser(); // This creates a new FileChooser object, which is a JavaFX component that opens a dialog window to let the user browse and choose files from their computer.
        open.setTitle("Open Image File");
        open.getExtensionFilters().add(new FileChooser.ExtensionFilter("Image File", "*jpg", "*png")); // This line adds a filter to the FileChooser, so only image files with the extensions .jpg and .png are shown in the file browser. The user will only be able to select these types of image files.

        File file = open.showOpenDialog(main_form.getScene().getWindow()); // This is the method that opens the file dialog window, allowing the user to browse and select an image file.

        /*
        --What is a Modal Window?--
        A modal window is a type of window that temporarily blocks the interaction with the parent window (the main application window) until the modal window is closed. This means:

        While the modal window (in this case, the file dialog) is open, the user cannot interact with the main window (e.g., click buttons, type in text fields) until they either select a file or close the dialog.
        This is helpful to ensure that the user completes the current action (choosing an image) before interacting with the rest of the application.

        ---Why Specify the Parent Window?--
        The argument main_form.getScene().getWindow() is used to indicate the parent window of the file dialog, which ties the file dialog to the main application window. Here’s why this is important:

        Blocking interaction with the main window:

        When the file dialog is modal, it prevents the user from interacting with the parent window until they complete the task in the dialog (choosing a file or closing the dialog). This ensures the user focuses on selecting an image before doing anything else in the app.
        Dialog behavior and position:

        By specifying the parent window, the file dialog will appear centered relative to the main window. It also ensures that when you minimize or move the main window, the file dialog will minimize or move along with it.
```

```
690         If the user could interact with the main window while the file dialog was open, they might
    trigger unintended behaviors (such as clicking other buttons or opening additional dialogs), which
    could lead to confusion or errors in the application.
691
692          */
693         if (file != null) {
694
695             GetData.path = file.getAbsolutePath(); // On Windows, an absolute path might look like
    this: C:\Users\John\Pictures\image.jpg
696
697             image = new Image(file.toURI().toString(), 115, 128, false, true);
698             addProduct_imgView.setImage(image);
699
700         }
701     }
702
703     public void addProductsSearch() {
704
705         FilteredList<ProductsModel> filter = new FilteredList<>(addProductList, e -> true);
706
707         addProduct_search_txt.textProperty().addListener((Observable, oldValue, newValue) -> {
708
709             filter.setPredicate(predicateProductData -> {
710
711                 if (newValue == null || newValue.isEmpty()) {
712                     return true;
713                 }
714
715                 String searchKey = newValue.toLowerCase();
716
717                 if (predicateProductData.getProductId().toString().contains(searchKey)) {
718                     return true;
719                 } else if (predicateProductData.getType().toLowerCase().contains(searchKey)) {
720                     return true;
721                 } else if (predicateProductData.getBrand().toLowerCase().contains(searchKey)) {
722                     return true;
723                 } else if (predicateProductData.getProductName().toLowerCase().contains(searchKey)) {
724                     return true;
725                 } else if (predicateProductData.getPrice().toString().contains(searchKey)) {
726                     return true;
727                 } else if (predicateProductData.getStatus().toLowerCase().contains(searchKey)) {
728                     return true;
729                 } else {
730                     return false;
731                 }
732             });
733         });
734
735         SortedList<ProductsModel> sortList = new SortedList<>(filter);
736
737         sortList.comparatorProperty().bind(addProduct_tableView.comparatorProperty());
738         addProduct_tableView.setItems(sortList);
739
740     }
741
742     public ObservableList<ProductsModel> addProductListDataGet() {
743
744         ObservableList<ProductsModel> productList = FXCollections.observableArrayList();
745
746         String sql = "SELECT * FROM product";
```

```
750
751             prepare = conn.prepareStatement(sql);
752             resultSet = prepare.executeQuery();
753             ProductsModel proData;
754
755             while (resultSet.next()) {
756
757                 proData = new ProductsModel(
758                         resultSet.getInt("product_id"),
759                         resultSet.getString("type"),
760                         resultSet.getString("brand"),
761                         resultSet.getString("product_name"),
762                         resultSet.getDouble("price"),
763                         resultSet.getString("status"),
764                         resultSet.getString("image"),
765                         resultSet.getDate("date")
766                 );
767
768                 productList.add(proData);
769             }
770
771         } catch (Exception e) {
772             e.printStackTrace();
773         }
774
775         return productList;
776
777     }
778
779     private ObservableList<ProductsModel> addProductList;
780 //    public void addProductsShowListData(){
781 //
782 //        addProductList = addProductListDataGet();
783 //
784 //        addProduct_tbvCol_productID.setCellValueFactory(new PropertyValueFactory<>("product_id")); //
    table colomns name of database
785 //        addProduct_tbvCol_type.setCellValueFactory(new PropertyValueFactory<>("type"));
786 //        addProduct_tbvCol_brand.setCellValueFactory(new PropertyValueFactory<>("brand"));
787 //        addProduct_tbvCol_productName.setCellValueFactory(new
    PropertyValueFactory<>("product_name"));
788 //        addProduct_tbvCol_price.setCellValueFactory(new PropertyValueFactory<>("price"));
789 //        addProduct_tbvCol_status.setCellValueFactory(new PropertyValueFactory<>("status"));
790 //
791 //        addProduct_tableView.setItems(addProductList);
792 //    }
793
794     public void addProductsShowListData() { // show on table
795
796         addProductList = addProductListDataGet();
797
798         addProduct_tbvCol_productID.setCellValueFactory(new PropertyValueFactory<>("productId"));  //
    productId is the variable name of the ProductsModel class
799         addProduct_tbvCol_type.setCellValueFactory(new PropertyValueFactory<>("type")); // this does
    not immediately "put all the data related to the product ID at once." Instead, what it does is link the
    productId field from each product in your list to the Product ID column of the TableView
800         addProduct_tbvCol_brand.setCellValueFactory(new PropertyValueFactory<>("brand"));
801         addProduct_tbvCol_productName.setCellValueFactory(new PropertyValueFactory<>("productName"));
802         addProduct_tbvCol_price.setCellValueFactory(new PropertyValueFactory<>("price"));
803         addProduct_tbvCol_status.setCellValueFactory(new PropertyValueFactory<>("status"));
804
```

```
808     public void ordersAdd() {
809 
810         customerId();
811         String sql = "INSERT INTO customer (customer_id, type, brand, productName, quantity, price,
    date) "
812                 + "VALUES(?, ?, ?, ?, ?, ?, ?)";
813 
814         conn = databaseHandler.connectDb();
815 
816         try {
817 
818             String checkData = "SELECT * FROM product WHERE product_name = '"
819                     + orders_productName_combo.getSelectionModel().getSelectedItem() + "'";
820 
821             double priceData = 0;
822 
823             statement = conn.createStatement();
824             resultSet = statement.executeQuery(checkData);
825 
826             if (resultSet.next()) {
827                 priceData = resultSet.getDouble("price"); // if there are more prices more than 1 then
    what happen?????
828 //                ---------------
829 // ------------------
830             }
831 
832             double totalPdata = (priceData * qty);
833 
834             Alert alert;
835             if (orders_productType_combo.getSelectionModel().getSelectedItem() == null
836                     || orders_brandName_combo.getSelectionModel().getSelectedItem() == null
837                     || orders_productName_combo.getSelectionModel().getSelectedItem() == null
838                     || totalPdata == 0) {
839                 alert = new Alert(Alert.AlertType.ERROR);
840                 alert.setTitle("Error Message");
841                 alert.setHeaderText(null);
842                 alert.setContentText("Please choose product first");
843                 alert.showAndWait();
844             } else {
845 
846                 prepare = conn.prepareStatement(sql);
847                 prepare.setInt(1, customer_Id);
848                 prepare.setString(2, (String)
    orders_productType_combo.getSelectionModel().getSelectedItem());
849                 prepare.setString(3, (String)
    orders_brandName_combo.getSelectionModel().getSelectedItem());
850                 prepare.setString(4, (String)
    orders_productName_combo.getSelectionModel().getSelectedItem());
851                 prepare.setString(5, String.valueOf(qty));
852 
853                 prepare.setString(6, String.valueOf(totalPdata));
854 
855                 Date date = new Date();
856                 java.sql.Date sqlDate = new java.sql.Date(date.getTime());
857                 prepare.setString(7, String.valueOf(sqlDate));
858 
859                 prepare.executeUpdate();
860 
861                 ordersShowListData();
862 //              ordersDisplayTotal();
```

```
866             e.printStackTrace();
867         }
868 
869     }
870 
871     public void ordersPay() {
872 
873         customerId();
874         String sql = "INSERT INTO customer_receipt (customer_id, total, amount, balance, date) "
875                 + "VALUES(?, ?, ?, ?, ?)"; // WHERE customer_id = '"+ customer_Id +"'  // change this I
876 
877         conn = databaseHandler.connectDb();
878 
879         try {
880 
881             Alert alert;
882             if (totalP > 0) { // || orders_amount_txt.getText().isEmpty() || amount == 0
883 
884                 alert = new Alert(Alert.AlertType.CONFIRMATION);
885                 alert.setTitle("Confirmation Message");
886                 alert.setHeaderText(null);
887                 alert.setContentText("Are you sure?");
888                 Optional<ButtonType> optional = alert.showAndWait();
889 
890                 if (optional.get().equals(ButtonType.OK)) {
891 
892                     prepare = conn.prepareStatement(sql);
893 
894                     prepare.setInt(1, customer_Id); // I change this
895                     prepare.setString(2, String.valueOf(totalP));
896                     prepare.setString(3, String.valueOf(amount));
897                     prepare.setString(4, String.valueOf(balance));
898 
899                     Date date = new Date();
900                     java.sql.Date sqlDate = new java.sql.Date(date.getTime());
901                     prepare.setString(5, String.valueOf(sqlDate));
902 
903                     prepare.executeUpdate();
904 
905                     alert = new Alert(Alert.AlertType.INFORMATION);
906                     alert.setTitle("Information Message");
907                     alert.setHeaderText(null);
908                     alert.setContentText("Successfully Payed!");
909                     alert.showAndWait();
910 
911 //                    ordersShowListData();
912 
913 //                    totalP = 0;
914                     balance = 0;
915                     amount = 0;
916 
917                     orders_balance_lbl.setText("Rs 0.0");
918                     orders_total_lbl.setText("Rs 0.0");
919                     orders_amount_txt.setText("");
920 
921                 } else {
922                     return;
923                 }
924 
925             } else {
```

```
                alert.setHeaderText(null);
                alert.setContentText("Invalid : 3");
                alert.showAndWait();

            }

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

    }

    public void ordersReceipt() throws JRException{

        try {

//            customerId();

            HashMap hash = new HashMap();
            hash.put("inventoryP", customer_Id);

            if(totalP == 0){

                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("Invalid :3"); // kalin order eka pay karala iwara nam den totalP = 0 ema nisa receipt ekak print kranna denne aye order ekak daddi
                alert.showAndWait();

            }else{

                JasperDesign jDesign = JRXmlLoader.load("C:\\Users\\krith\\Downloads\\YT_InventoryManagementSystem_JavaFx-main\\YT_InventoryManagementSystem_JavaFx-main\\report.jrxml");
                JasperReport jReport = JasperCompileManager.compileReport(jDesign);
                JasperPrint jPrint = JasperFillManager.fillReport(jReport, hash, conn);

                JasperViewer.viewReport(jPrint, false);


            }

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

    }

    public void ordersReset(){

        customerId();

        String sql = "DELETE FROM customer WHERE customer_id = '"+ customer_Id +"'";

        conn = databaseHandler.connectDb();

        try {
            if(!orders_tableView.getItems().isEmpty()){

                Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
                alert.setTitle("Confirmation Message");
```

```
989
990                 if(option.get().equals(ButtonType.OK)){
991
992                     statement = conn.createStatement();
993                     statement.executeUpdate(sql);
994
995                     ordersShowListData();
996
997                     totalP = 0;
998                     balance = 0;
999                     amount = 0;
1000
1001                        orders_balance_lbl.setText("Rs 0.0");
1002                        orders_total_lbl.setText("Rs 0.0");
1003                        orders_amount_txt.setText("");
1004                    }
1005
1006                }
1007            } catch (Exception e) {e.printStackTrace();}
1008
1009        }
1010
1011        private double amount;
1012        private double balance;
1013
1014        public void ordersAmount() {
1015
1016            Alert alert;
1017
1018            if (!orders_amount_txt.getText().isEmpty()) {
1019
1020                amount = Double.parseDouble(orders_amount_txt.getText());
1021
1022                if (totalP > 0) {
1023                    if (amount >= totalP) {
1024                        balance = (amount - totalP);
1025                        orders_balance_lbl.setText("Rs " + String.valueOf(balance));
1026                    } else {
1027                        alert = new Alert(Alert.AlertType.ERROR);
1028                        alert.setHeaderText(null);
1029                        alert.setContentText("Enter Enough Amount");
1030                        alert.showAndWait();
1031
1032                        orders_amount_txt.setText("");
1033                    }
1034                } else {
1035
1036                    alert = new Alert(Alert.AlertType.ERROR);
1037                    alert.setHeaderText(null);
1038                    alert.setContentText("Invalid :3");
1039                    alert.showAndWait();
1040
1041                }
1042            } else {
1043
1044                    alert = new Alert(Alert.AlertType.ERROR);
1045                    alert.setHeaderText(null);
1046                    alert.setContentText("Amount Field is Empty");
1047                    alert.showAndWait();
1048
```

```
1052
1053        private double totalP;
1054
1055        public void ordersDisplayTotal() {
1056
1057            customerId();
1058
1059            String sql = "SELECT SUM(price) FROM customer WHERE customer_id = '" + customer_Id + "'
    ";   // mehidee cus id of customer table is 1, and cus id of customer_receipt is 0, customerId() ekedi
    customer table eke customer id eka select krala gannwa, eka thamay methana thiyenne '"+customer_Id+"'
    lesa.
1060
1061            conn = databaseHandler.connectDb();
1062
1063            try {
1064
1065                prepare = conn.prepareStatement(sql);
1066                resultSet = prepare.executeQuery();
1067
1068                while (resultSet.next()) {
1069                    totalP = resultSet.getDouble("SUM(price)");
1070                }
1071
1072                orders_total_lbl.setText("Rs " + String.valueOf(totalP));
1073
1074            } catch (Exception e) {
1075                e.printStackTrace();
1076            }
1077
1078        }
1079
1080        private String[] OrderslistType = {"Snacks", "Drinks", "Dessert", "Gadgets", "Personal
    Product", "Others"};
1081
1082        public void OrdersListTypes() {
1083
1084    //        orders_productType_combo.getSelectionModel().clearSelection();
1085            List<String> listT = new ArrayList<>();
1086
1087            for (String data : OrderslistType) {
1088                listT.add(data);
1089            }
1090
1091            ObservableList listData = FXCollections.observableArrayList(listT);
1092            orders_productType_combo.setItems(listData);
1093
1094            ordersListBrands();
1095        }
1096
1097        public void ordersListBrands() { // this method call after type selected, but when selected
    brand name combo we want call again this mehtod, because we want to show product name combo list
1098
1099            // Clear the product name combo box when a new type is selected
1100            orders_productName_combo.getSelectionModel().clearSelection();
1101            orders_productName_combo.getItems().clear();
1102
1103            // Check if brand selection exists before executing the query
1104            if (orders_productType_combo.getSelectionModel().getSelectedItem() == null) {
1105                return; // Exit the method if no brand is selected
1106            }
```

```
1110                     + "' and status = 'Available'"; // GROUP BY brand
1111
1112             conn = databaseHandler.connectDb();
1113
1114             try {
1115
1116                 prepare = conn.prepareStatement(sql);
1117                 resultSet = prepare.executeQuery();
1118
1119                 ObservableList listData = FXCollections.observableArrayList();
1120
1121                 while (resultSet.next()) {
1122                     listData.add(resultSet.getString("brand"));
1123                 }
1124
1125                 orders_brandName_combo.setItems(listData);
1126
1127                 ordersListProductName();
1128             } catch (Exception e) {
1129                 e.printStackTrace();
1130             }
1131
1132         }
1133
1134         public void ordersListProductName() {
1135
1136             // Check if brand selection exists before executing the query
1137             if (orders_brandName_combo.getSelectionModel().getSelectedItem() == null) {
1138                 return; // Exit the method if no brand is selected
1139             }
1140
1141             String sql = "SELECT * FROM product WHERE brand = '"
1142                     + orders_brandName_combo.getSelectionModel().getSelectedItem() + "'";
1143
1144             conn = databaseHandler.connectDb();
1145
1146             try {
1147
1148                 prepare = conn.prepareStatement(sql);
1149                 resultSet = prepare.executeQuery();
1150
1151                 ObservableList listData = FXCollections.observableArrayList();
1152
1153                 while (resultSet.next()) {
1154                     listData.add(resultSet.getString("product_name"));
1155                 }
1156
1157                 orders_productName_combo.setItems(listData);
1158
1159     //            while(resultSet.next()){
1160     //                customerData = new CustomerData(
1161     //                        resultSet.getInt("customer_id"),
1162     //                        resultSet.getString("type"),
1163     //                        resultSet.getString("brand"),
1164     //                        resultSet.getString("productName"),
1165     //                        resultSet.getInt("quantity"),
1166     //                        resultSet.getDouble("price"),
1167     //                        resultSet.getDate("date"));
1168     //
1169     //                listData.add(customerData);
```

```
1173            }
1174        }
1175
1176        private SpinnerValueFactory<Integer> spinner;
1177
1178        public void ordersSpinner() {
1179
1180            spinner = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 20, 0);
1181            orders_quantity_spinner.setValueFactory(spinner);
1182
1183            /*
1184
1185            --- ordersSpinner() Method ---
1186
1187            This method is used to set up a Spinner for ordering quantities, allowing the user to
    select a number between 0 and 20. Here's a detailed explanation of each part of this method:
1188
1189            1. Spinner Initialization:
1190
1191
1192            spinner = new SpinnerValueFactory.IntegerSpinnerValueFactory(0, 20, 0);
1193
1194            SpinnerValueFactory.IntegerSpinnerValueFactory(0, 20, 0) creates an integer-based spinner
    with a minimum value of 0, a maximum value of 20, and an initial/default value of 0.
1195            The SpinnerValueFactory helps manage the range and value of the spinner for quantity
    selection.
1196
1197
1198            2. Assigning the Spinner to the orders_quantity_spinner:
1199
1200            orders_quantity_spinner.setValueFactory(spinner);
1201
1202            Here, orders_quantity_spinner is a spinner UI element (likely created in the FXML file or
    elsewhere in your JavaFX UI code).
1203            This line assigns the spinner factory as the value factory for orders_quantity_spinner.
    This means that the spinner's value range (0 to 20) and starting value (0) are now applied to
    orders_quantity_spinner.
1204
1205
1206            3. Placement in Initialization or Button Click:
1207
1208            ordersSpinner() is called in the initialize() method (often used in JavaFX controllers to
    initialize UI elements when the scene is loaded) or on a button click.
1209            When called in initialize(), it sets up the spinner with the specified range and default
    value as soon as the interface is loaded, making it ready for user interaction.
1210
1211             */
1212        }
1213
1214        private int qty;
1215
1216        public void ordersShowSpinnerValue() {
1217            qty = orders_quantity_spinner.getValue();
1218        }
1219
1220        public ObservableList<CustomerData> ordersListData() {
1221            customerId(); // pay button eke explain ekata meka yodaganna
1222            ObservableList<CustomerData> listData = FXCollections.observableArrayList();
1223            String sql = "SELECT * FROM customer WHERE customer_id = '" + customer_Id + "' ";
1224
```

```
1228
1229                CustomerData customerData;
1230                prepare = conn.prepareStatement(sql);
1231                resultSet = prepare.executeQuery();
1232
1233                while (resultSet.next()) {
1234                    customerData = new CustomerData(
1235                            resultSet.getInt("customer_id"),
1236                            resultSet.getString("type"),
1237                            resultSet.getString("brand"),
1238                            resultSet.getString("productName"),
1239                            resultSet.getInt("quantity"),
1240                            resultSet.getDouble("price"),
1241                            resultSet.getDate("date"));
1242
1243                    listData.add(customerData);
1244                }
1245
1246            } catch (Exception e) {
1247                e.printStackTrace();
1248            }
1249
1250            return listData;
1251        }
1252
1253        private ObservableList<CustomerData> orderList;
1254
1255        public void ordersShowListData() {
1256
1257            orderList = ordersListData();
1258
1259            orders_col_type.setCellValueFactory(new PropertyValueFactory<>("type"));
1260            orders_col_brand.setCellValueFactory(new PropertyValueFactory<>("brand"));
1261            orders_col_productName.setCellValueFactory(new PropertyValueFactory<>("productName"));
1262            orders_col_quantity.setCellValueFactory(new PropertyValueFactory<>("quantity"));
1263            orders_col_price.setCellValueFactory(new PropertyValueFactory<>("price"));
1264
1265            orders_tableView.setItems(orderList);
1266
1267            ordersDisplayTotal();
1268        }
1269
1270        private int customer_Id;
1271
1272        public void customerId() {
1273
1274            String customerId = "SELECT * FROM customer";
1275            conn = databaseHandler.connectDb();
1276
1277            try {
1278
1279                prepare = conn.prepareStatement(customerId);
1280                resultSet = prepare.executeQuery();
1281
1282                int checkId = 0;
1283
1284                while (resultSet.next()) {
1285
1286                    // GET LAST CUSTOMER ID
1287                    customer_Id = resultSet.getInt("customer_id");
```

```
1291                String checkData = "SELECT * FROM customer_receipt";
1292
1293                statement = conn.createStatement();
1294                resultSet = statement.executeQuery(checkData);
1295
1296                while (resultSet.next()) {
1297                    checkId = resultSet.getInt("customer_id");
1298                }
1299
1300                if (customer_Id == 0) {
1301                    customer_Id += 1;
1302                } else if (checkId == customer_Id) {
1303                    customer_Id += 1;
1304                }
1305
1306                /*
1307
1308                above block checks if a unique customer_Id should be assigned:
1309
1310                if(customer_Id == 0) checks if there were no customer_id records in the customer table
    and increments customer_Id by 1.
1311                else if(checkId == customer_Id) checks if the last customer_id from customer_receipt
    matches customer_Id. If they are the same, it increments customer_Id by 1 to avoid duplication.
1312
1313                 */
1314            } catch (Exception e) {
1315                e.printStackTrace();
1316            }
1317        }
1318
1319        public void switchForm(ActionEvent event) {
1320
1321            if (event.getSource() == home_btn) {
1322
1323                home_form.setVisible(true);
1324                addProducts_form.setVisible(false);
1325                orders_form.setVisible(false);
1326
1327                home_btn.setStyle("-fx-background-color:linear-gradient(to bottom right, #269e70,
    #969635);");
1328                addProducts_btn.setStyle("-fx-background-color:transparent;");
1329                orders_btn.setStyle("-fx-background-color:transparent;");
1330
1331                homeDisplayTotalOrders();
1332                homeTotalIncome();
1333                homeAvailableProducts();
1334                homeIncomeChart();
1335                homeOrdersChart();
1336
1337            } else if (event.getSource() == addProducts_btn) {
1338
1339                home_form.setVisible(false);
1340                addProducts_form.setVisible(true);
1341                orders_form.setVisible(false);
1342
1343                home_btn.setStyle("-fx-background-color:transparent;");
1344                addProducts_btn.setStyle("-fx-background-color:linear-gradient(to bottom right,
    #269e70, #969635);");
1345                orders_btn.setStyle("-fx-background-color:transparent;");
1346
```

```
1350                addProductsSearch();
1351
1352            } else if (event.getSource() == orders_btn) {
1353
1354                home_form.setVisible(false);
1355                addProducts_form.setVisible(false);
1356                orders_form.setVisible(true);
1357
1358                home_btn.setStyle("-fx-background-color:transparent;");
1359                addProducts_btn.setStyle("-fx-background-color:transparent;");
1360                orders_btn.setStyle("-fx-background-color:linear-gradient(to bottom right, #269e70,
    #969635);");
1361
1362                orders_productType_combo.getSelectionModel().clearSelection();
1363                orders_brandName_combo.getSelectionModel().clearSelection();
1364                orders_productName_combo.getSelectionModel().clearSelection();
1365
1366                ordersShowListData();
1367                OrdersListTypes();
1368                ordersListBrands();
1369                ordersListProductName();
1370                ordersSpinner();
1371
1372            }
1373        }
1374
1375        public void defaultNav(){
1376            home_btn.setStyle("-fx-background-color:linear-gradient(to bottom right, #269e70,
    #969635);"); // software eka run karala home page eka penne, ema nisa home_btn eka click unama watena
    pata watila tyena one muladima
1377        }
1378
1379        private double x = 0;
1380        private double y = 0;
1381
1382        public void logout() {
1383            try {
1384                Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
1385                alert.setTitle("Confirmation Message");
1386                alert.setHeaderText(null);
1387                alert.setContentText("Are you sure want to logout?");
1388                Optional<ButtonType> option = alert.showAndWait();
1389
1390                if (option.get().equals(ButtonType.OK)) {
1391
1392                    main_form.getScene().getWindow().hide();
1393
1394                    // LINK YOUR LOGIN FORM
1395                    Parent root = FXMLLoader.load(getClass().getResource("/fxml/Login.fxml"));
1396                    Stage stage = new Stage();
1397                    Scene scene = new Scene(root);
1398
1399                    root.setOnMousePressed((MouseEvent event) -> {
1400                        x = event.getSceneX();
1401                        y = event.getSceneY();
1402                    });
1403
1404                    root.setOnMouseDragged((MouseEvent event) -> {
1405                        stage.setX(event.getScreenX() - x);
1406                        stage.setY(event.getScreenY() - y);
```

```
1410
1411                    root.setOnMouseReleased((MouseEvent event) -> {
1412                        stage.setOpacity(1);
1413                    });
1414
1415                    stage.initStyle(StageStyle.TRANSPARENT);
1416
1417                    stage.setScene(scene);
1418                    stage.show();
1419                } else {
1420                    return;
1421                }
1422
1423            } catch (Exception e) {
1424                e.printStackTrace();
1425            }
1426        }
1427
1428        public void displayUsername(){
1429            username_lbl.setText(GetData.username);
1430        }
1431
1432        public void minimize() {
1433            Stage stage = (Stage) main_form.getScene().getWindow();
1434            stage.setIconified(true);
1435        }
1436
1437        public void close() {
1438            System.exit(0);
1439        }
1440
1441        @Override
1442        public void initialize(URL url, ResourceBundle rb) {
1443
1444            displayUsername();
1445            defaultNav();
1446
1447            homeDisplayTotalOrders();
1448            homeTotalIncome();
1449            homeAvailableProducts();
1450            homeIncomeChart();
1451            homeOrdersChart();
1452
1453            // TO SHOW THE DATA ON TABLEVIEW
1454            addProductsShowListData();
1455            addProductsListTypes();
1456            addProductsListStatus();
1457
1458            ordersShowListData();
1459            OrdersListTypes();
1460            ordersListBrands();
1461            ordersListProductName();
1462            ordersSpinner();
1463        }
1464
1465    }
```

```
import java.io.IOException;
import javafx.scene.input.MouseEvent;
import javafx.stage.StageStyle;

/**
 * JavaFX App
 */
public class App extends Application {

    private static Scene scene;
    private static Parent root;
    private double x = 0;
    private double y = 0;

    @Override
    public void start(Stage stage) throws IOException {

        Parent root = FXMLLoader.load(getClass().getResource("/fxml/Login.fxml"));
        scene = new Scene(root, 623, 439); // Specify the fxml folder

        root.setOnMousePressed((MouseEvent event) -> {
            x = event.getSceneX();
            y = event.getSceneY();
        });

        root.setOnMouseDragged((MouseEvent event) -> {
            stage.setX(event.getScreenX() - x);
            stage.setY(event.getScreenY() - y);

            stage.setOpacity(0.8);
        });

        root.setOnMouseReleased((MouseEvent event) -> {
            stage.setOpacity(1);
        });

        stage.initStyle(StageStyle.TRANSPARENT);

        stage.setScene(scene);
        stage.show();
    }

    static void setRoot(String fxml) throws IOException {
        scene.setRoot(loadFXML("fxml/" + fxml)); // Specify the fxml folder
    }

    private static Parent loadFXML(String fxml) throws IOException {
        FXMLLoader fxmlLoader = new FXMLLoader(App.class.getResource("/" + fxml + ".fxml")); // Include leading slash for
resources
        return fxmlLoader.load();
    }

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

4.4 Customer data

```
package com.mycompany.yt_inventorymanagementsystem_javafx;

import java.util.Date;
```

```
 *
 * @author USER
 */
public class CustomerData {

    private Integer customerId;
    private String type;
    private String brand;
    private String productName;
    private Integer quantity;
    private Double price;
    private Date date;

    public CustomerData(Integer customerId, String type, String brand, String productName, Integer quantity, Double
price, Date date) {
        this.customerId = customerId;
        this.type = type;
        this.brand = brand;
        this.productName = productName;
        this.quantity = quantity;
        this.price = price;
        this.date = date;
    }

    public Integer getCustomerId() {
        return customerId;
    }

    public String getType() {
        return type;
    }

    public String getBrand() {
        return brand;
    }

    public String getProductName() {
        return productName;
    }

    public Integer getQuantity() {
        return quantity;
    }

    public Double getPrice() {
        return price;
    }

    public Date getDate() {
        return date;
    }


}
```

4.5 Primary

```
5  package com.mycompany.yt_inventorymanagementsystem_javafx;
6
7  import java.io.IOException;
8  import javafx.fxml.FXML;
9
10 public class PrimaryController {
11
12     @FXML
13     private void switchToSecondary() throws IOException {
14         App.setRoot("secondary");
15     }
16 }
```

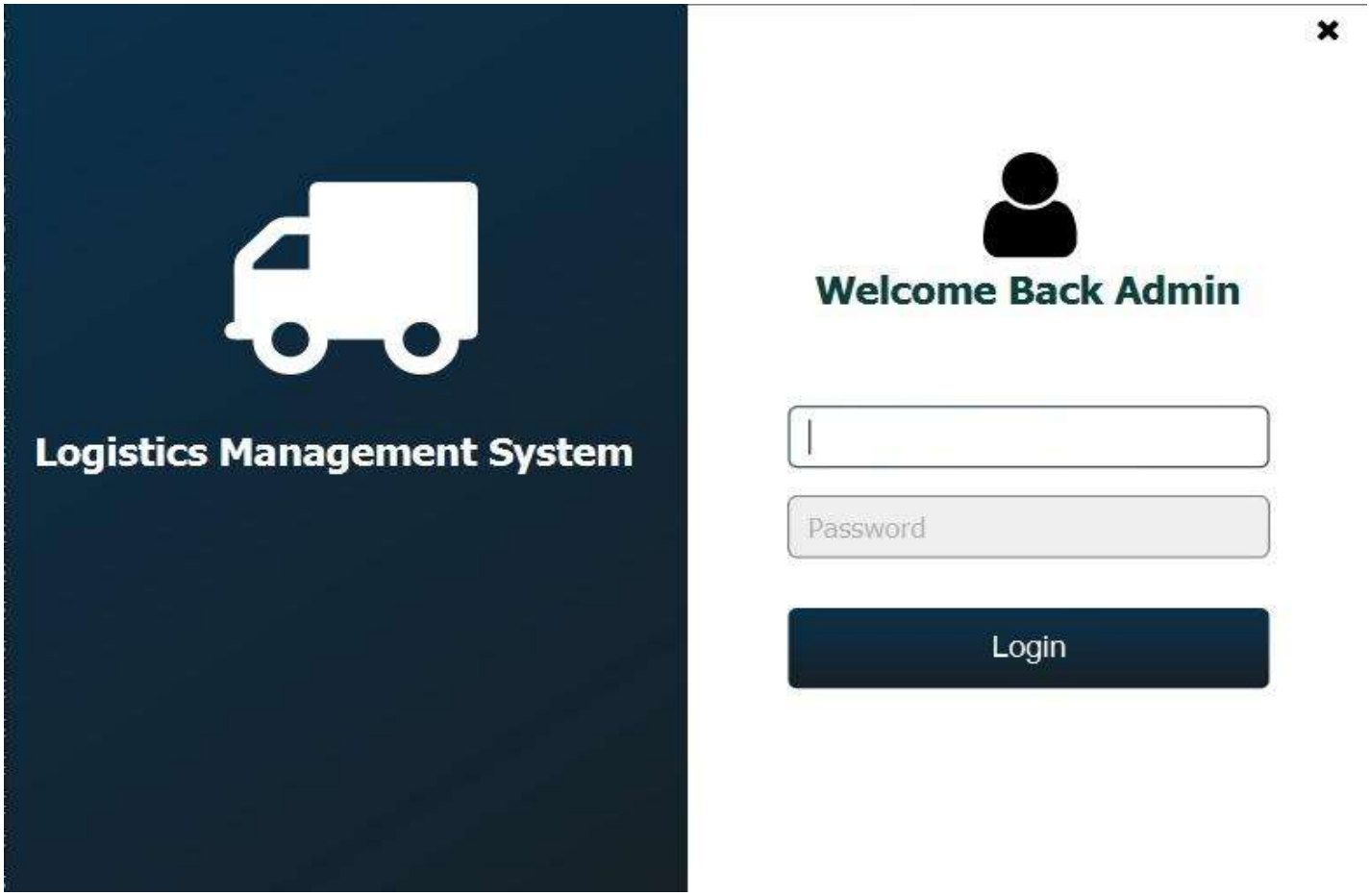
4.6 Products model

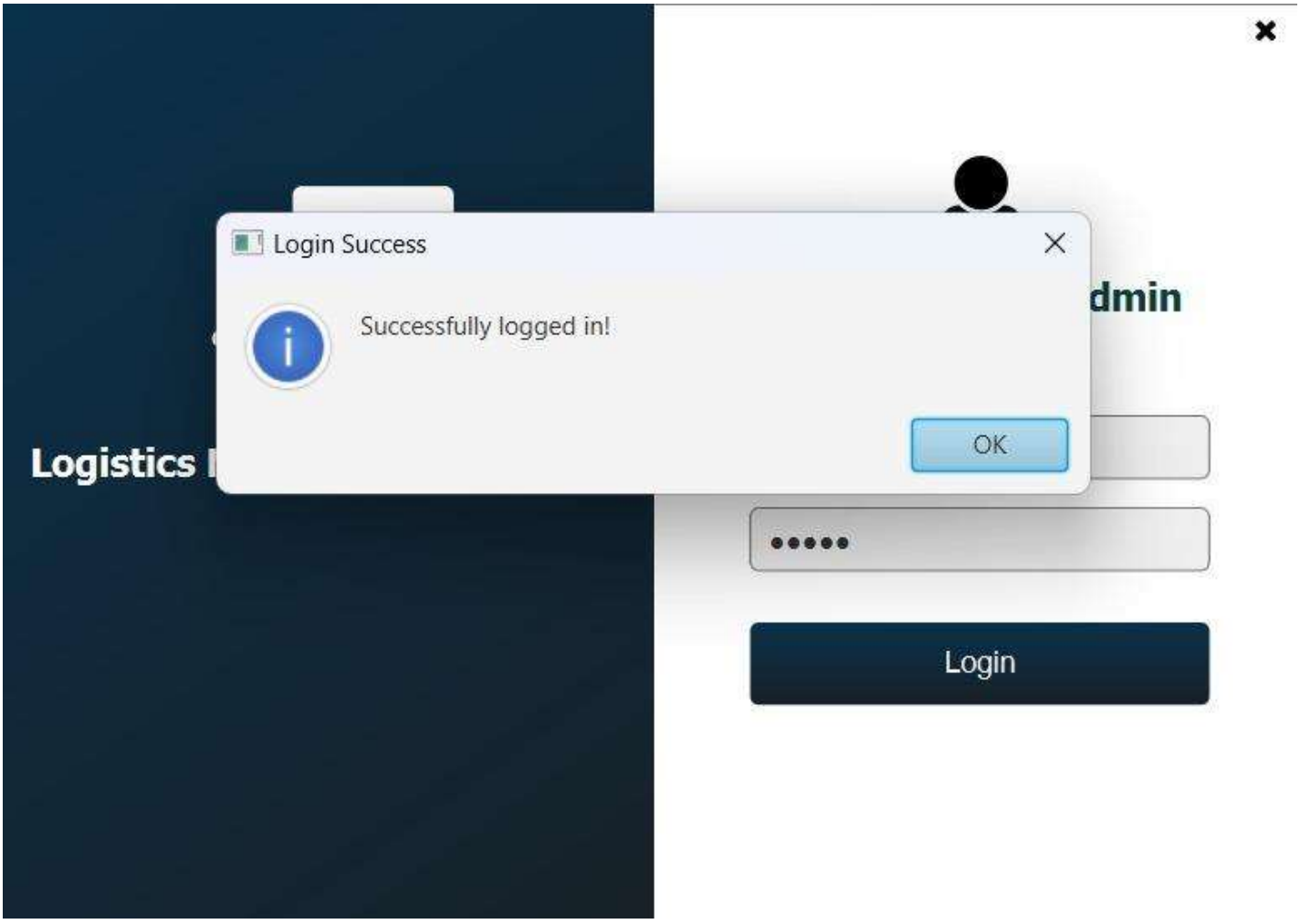
```
package com.mycompany.yt_inventorymanagementsystem_javafx;

import java.sql.Date;

/**
 *
 * @author USER
 */
public class ProductsModel {

    private Integer productId;
    private String type;
    private String brand;
    private String productName;
    private Double price;
    private String status;
    private String image;
    private Date date; //  It represents only the date (year, month, and day) without any time information (hours, minutes, seconds, or milliseconds).

    public ProductsModel(Integer productId, String type, String brand, String productName, Double price, String status, String image, Date date) {
        this.productId = productId;
        this.type = type;
        this.brand = brand;
        this.productName = productName;
        this.price = price;
        this.status = status;
        this.image = image;
        this.date = date;
    }

    public Integer getProductId() {
        return productId;
    }

    public String getType() {
        return type;
    }

    public String getBrand() {
        return brand;
    }

    public String getProductName() {
```

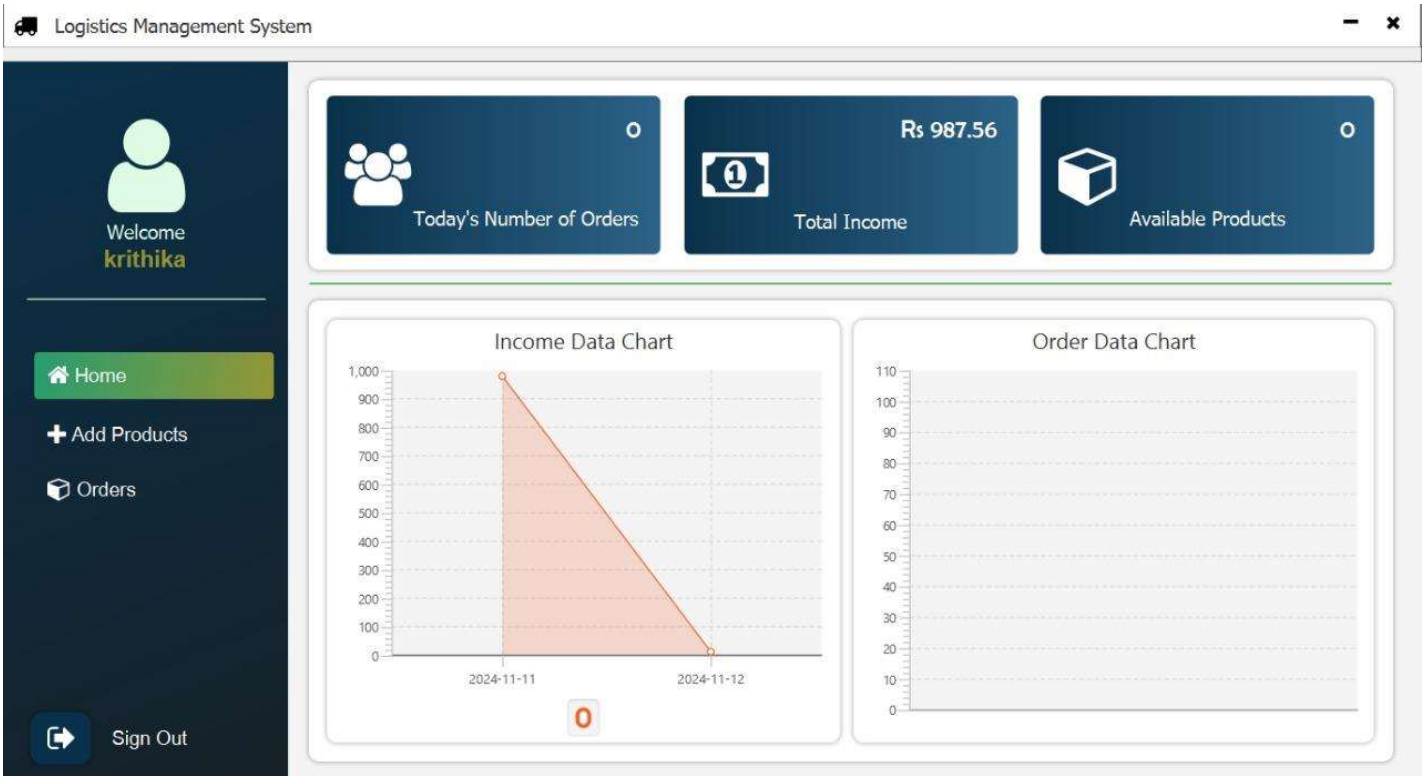
```
    public Double getPrice() {
        return price;
    }

    public String getStatus() {
        return status;
    }

    public String getImage() {
        return image;
    }

    public Date getDate() {
        return date;
    }


}
```

4.7 Module info :

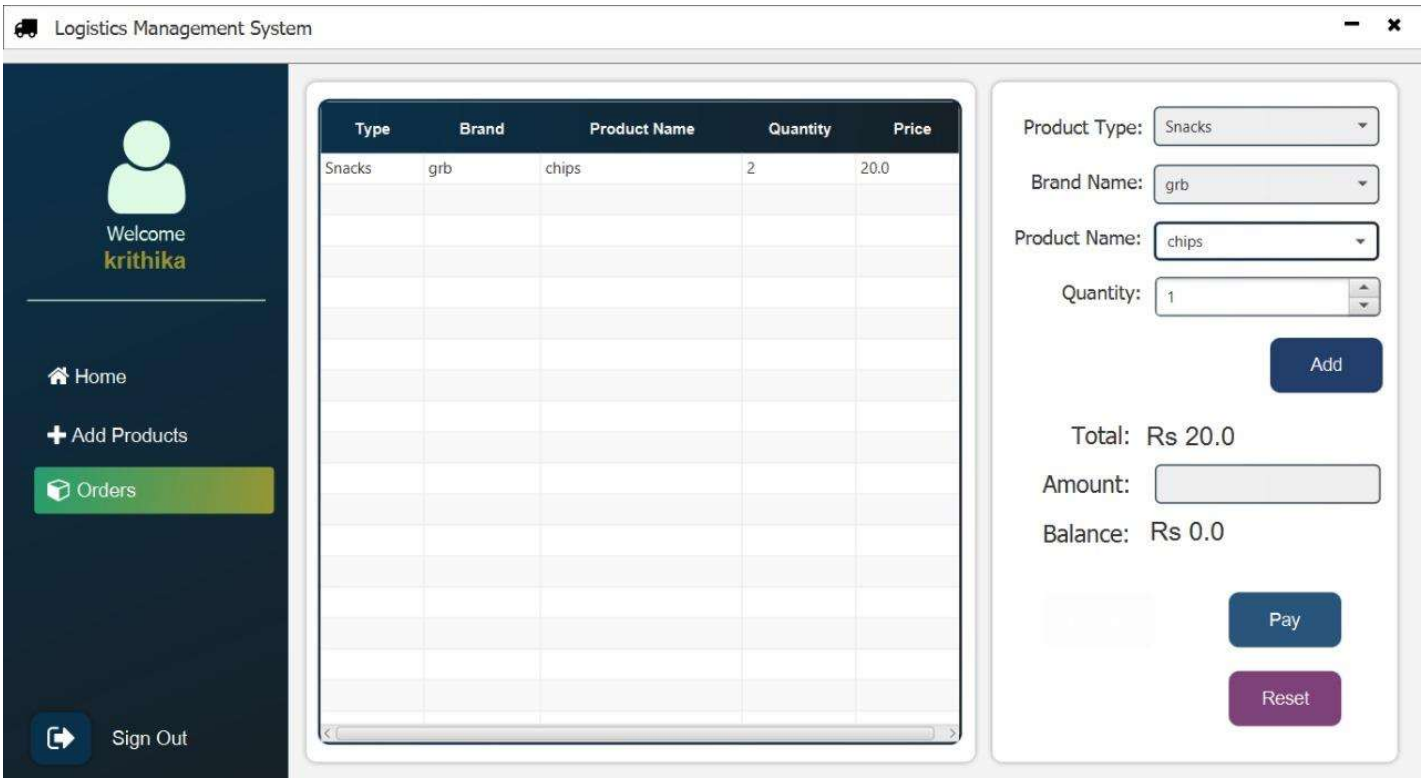
```
module com.mycompany.yt_inventorymanagementsystem_javafx {
    requires javafx.controls;
    requires javafx.fxml;
    requires java.sql;
    requires de.jensd.fx.glyphs.fontawesome;
    requires java.base;
    requires jasperreports;

    opens com.mycompany.yt_inventorymanagementsystem_javafx to javafx.fxml;
    exports com.mycompany.yt_inventorymanagementsystem_javafx;
}
```


Logistics Management System
Welcome Back Admin
Password
Login

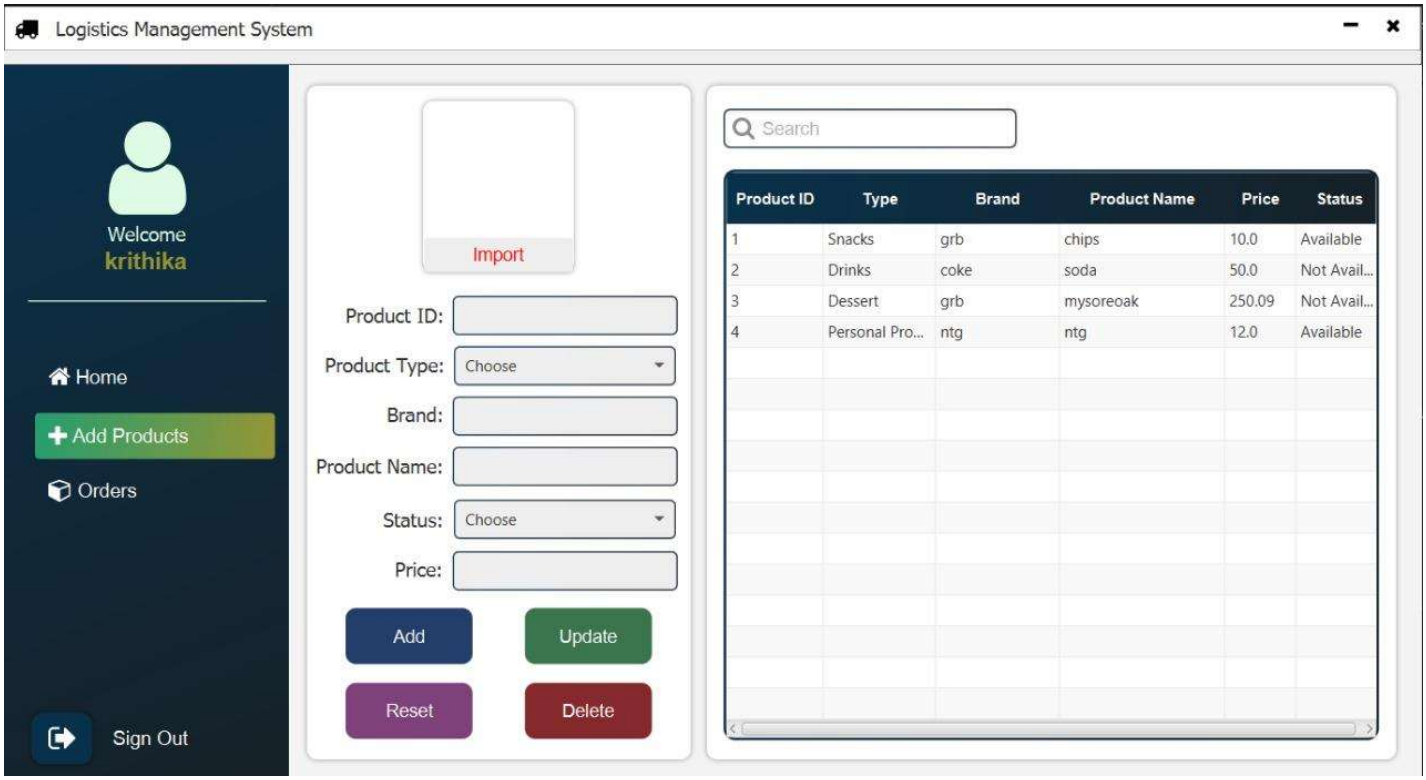

Login Success
Successfully logged in!
OK
dmin
Logistics
Login


Logistics Management System
Welcome
krithika
Home
Add Products
Orders
Sign Out
0
Today's Number of Orders
Rs 987.56
Total Income
0
Available Products
Income Data Chart
1,000
900
800
700
600
500
400
300
200
100
0
2024-11-11
2024-11-12
0
Order Data Chart
110
100
90
80
70
60
50
40
30
20
10
0

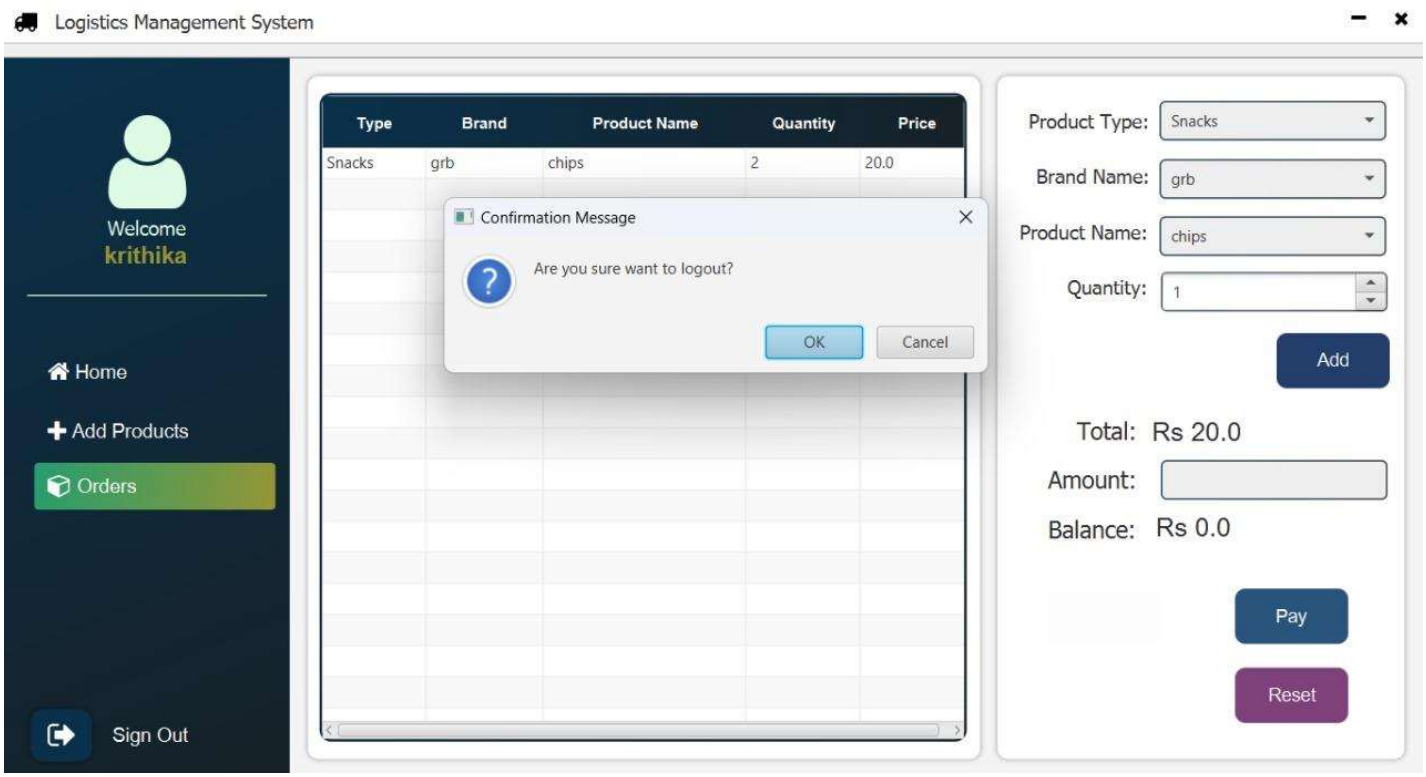
5.4 entering orders



5.5 adding products



5.6 logout successful



Conclusion

With the help of our project, businesses can efficiently manage their logistics operations by tracking inventory, coordinating with suppliers, and processing orders seamlessly. The system organizes all critical data in one centralized platform, accessible through their accounts, saving time and ensuring a more streamlined, professional approach to logistics management.

Reference

1. https://www.javatpoint.com/java-tutorial
2. https://www.wikipedia.org/
3. https://www.w3schools.com/sql/
4. https://www.codecademy.com/learn/learn-sql