FPT ACADEMY INTERNATIONAL

FPT – APTECH COMPUTER EDUCATION

Centre Name: ACE-THUDUC-1-FPT

Address: 62 Street 36, Van Phuc Residential Area, Hiep Binh Phuoc Ward, Thu Duc City

Pharmacy

Supervisor	**Mr. Pham Cong Danh**	
Batch	**T5.2308.M0**	
Group	**Group 02**	
No	**Student code**	**Full name**
1	Student1501057	Hoang Gia Huy
2	Student1501059	Nguyen Anh Quan
3	Student1501053	Nguyen Anh Minh
4	Student1501060	Tran Nhat Linh
5	Student1491746	Doan Duc Do

Month: June Year: 2024

This is to certify that

Mr. **Hoang Gia Huy**

Mr. **Nguyen Anh Quan**

Mr. **Nguyen Anh Minh**

Mr. **Tran Nhat Linh**

Mr. **Doan Duc Do**

Have successfully Designed & Developed

Pharmacy

Submitted by:

Mr. PHAM CONG DANH

Date Of Issue:

10/7/2024

Authorized Signature:

CONTENT

ACKNOWLEDGE

We extend our deepest gratitude and appreciation to all those who have contributed to the successful completion of our Semester 2 project as part of the Advanced Diploma of Software Engineering program. This collaborative effort involved the dedicated work of our team of five students, who have tirelessly strived to deliver a robust and innovative pharmacy management application.

First and foremost, we would like to express our heartfelt thanks to our esteemed teacher, Mr. Danh, whose guidance and mentorship have been instrumental throughout the development of our project. His wealth of knowledge, unwavering support, and constructive feedback have been invaluable, shaping not only our technical skills but also fostering a deep understanding of software engineering principles.

The foundation of our project lies in the integration of various technologies, and we want to acknowledge the pivotal role played by JavaFX and Scene Builder in NetBeans. These tools have been the building blocks of our project, enabling us to create a dynamic and feature-rich application. The hands-on experience gained in utilizing these technologies has not only enriched our technical prowess but also provided practical insight into real-world software development.

The collaborative nature of this project has allowed us to cultivate effective teamwork and communication skills. Working as a cohesive unit, we learned the importance of coordination, task delegation, and problem-solving. These invaluable skills extend beyond the realm of software engineering and are applicable to various aspects of our academic and professional lives.

Furthermore, we want to express our gratitude to our fellow team members for their commitment and diligence. Each member has brought a unique set of skills and perspectives, contributing to the overall success of the project.

In conclusion, this project has been a journey of growth, learning, and camaraderie. We are sincerely thankful to everyone who has been part of this endeavor, shaping it into a rewarding experience that will undoubtedly influence our future endeavors in the field of software engineering.

INTRODUCTION

Greetings! We are excited to present our project in the realm of software engineering. As part of our Advanced Diploma studies, our team of five diligent students has developed a comprehensive pharmacy management application. The primary objective is to provide pharmacies with a user-friendly platform to efficiently manage their business operations.

We aim to deliver an application that is not only visually appealing but also technologically robust. Our focus is on creating a seamless user experience, enabling pharmacies to easily handle tasks such as inventory management, prescription tracking, and customer care. With intuitive management features, our goal is to simplify the management process, providing users with a convenient and efficient means to run their pharmacies.

We invite you to join us on this technological journey as we unveil a pharmacy management application that not only showcases our technical expertise but also enhances the operational efficiency of every pharmacy.

PROBLEM DEFINITION

Our project focuses on addressing common struggles that pharmacies face when managing and operating their businesses. Currently, tasks like inventory management, order tracking, and customer interaction are not always straightforward and can be time-consuming. Additionally, existing solutions often do not leverage the latest technologies, resulting in a less smooth user experience. Our goal is to tackle these issues by creating a simple and effective pharmacy management application.

We utilize JavaFX and Scene Builder to build the user interface, along with SQL for data storage and management. In doing so, we aim to create an application that users can easily utilize to manage inventory, record orders, and interact with customers efficiently. We believe that this project will help improve the productivity and working experience of pharmacies, enabling them to save time and enhance their business management.

CUSTOMER REQUIREMENT SPECIFICATIONS

1.USER/INTERFACE

Input:

1. **Forms and Text Fields:** Users can input information through forms and text fields. This could include filling out personal details, selecting preferences, or entering search queries.
2. **Buttons and Links** Users interact with buttons and links to navigate through different sections of the pharmacy management app, submit forms, or trigger specific actions. Thanks to the user-friendly and intuitive interface, users can easily check inventory status, manage prescriptions, and track sales revenue. The app also provides real-time notifications, allowing users to quickly stay updated on important changes or the latest information.
3. **Dropdowns and Selections:** Users can make selections from dropdown menus or choose options from lists to customize their experience.
4. **Checkboxes and Radio Buttons:** Users can provide input by selecting checkboxes or radio buttons to indicate preferences or choices.
5. **Uploads:** If applicable, users may upload files, images, or documents using file input fields.

Process:

1. **Form Submission:** When users fill out forms and submit them, the application processes the input, which may involve data validation and verification.
2. **Dynamic Updates:** Based on user input or interactions, certain sections of the application may dynamically update without requiring a full app reload.
3. **Client-Side Scripting:** JavaFX can be used for client-side scripting to enhance user interaction, validate input, and dynamically modify the content without reloading the entire page.

Output:

1. **Displaying Results:** The application outputs information based on user input, such as displaying search results, updated content, or personalized recommendations.
2. **Error Messages:** If there are issues with the input, the application may output error messages to guide users in correcting their input.
3. **Confirmation Messages:** Users may receive confirmation messages for successful actions, such as submitting a form or completing a transaction.
4. **Visual Changes:** The application may undergo visual changes to reflect the processed information, such as updating images, text, or layout dynamically.

2. ADMIN

Input

1. **Login Credentials:** The system requires input in the form of username and password for authentication during the login process.
2. **Configuration Settings:** Administrators may input or modify configuration settings, such as system preferences, security parameters, or feature toggles.

Process:

1. **Authentication:** The system processes login credentials to authenticate administrators and grant access to the admin panel.
2. **Authorization:** Once authenticated, the system checks for appropriate authorization levels to determine the scope of actions an administrator can perform.
3. **Configuration Processing:** Changes made to configuration settings are processed by the system to implement the updated preferences or rules.

Output:

1. **Admin Dashboard:** After successful login, the system outputs an admin dashboard, providing an overview of system status, analytics, or key metrics.
2. **Error Messages:** In case of incorrect login credentials or invalid input during configuration, the system outputs error messages to guide administrators.
3. **Activity Logs:** The system generates logs to record administrator activities, helping to track changes and monitor system interactions.
4. **System Status Updates:** If configuration settings are modified, the system outputs updates to reflect changes in system behavior or functionality.
5. **Confirmation Messages:** After successful actions, such as data uploads or configuration updates, the system outputs confirmation messages to inform administrators.

Additional Aspects:

1. **Role Management:** The system allows administrators to manage user roles and permissions, defining who can access certain features or perform specific actions.
2. **Security Measures:** Input validation, secure authentication protocols, and encryption are crucial components to ensure the security of admin interactions.
3. **Notification System:** The system may include a notification system to alert administrators about critical events, updates, or potential issues.
4. **Backup and Recovery:** Administrators may have the capability to initiate backup processes and recovery procedures in case of data loss or system failures.

Hardware / Software requirement

Hardware:

For Setting Up the Application

- Processor Intel Core I3 or higher.
- Memory 6 GB RAM or greater.
- Monitor: Super VGA (1024x768) or higher resolution
- Internet Access: Modem/ADSL Internet access is required

Software:

Server:

- Operation System: Window 7 or later.
- Databases: SQL.
- Development Framework: JavaFX
- Browser: Google Chrome version 35.

Client:

- Operation System Window 7 or higher.
- Browser Google Chrome , MS-Edge, Firefox

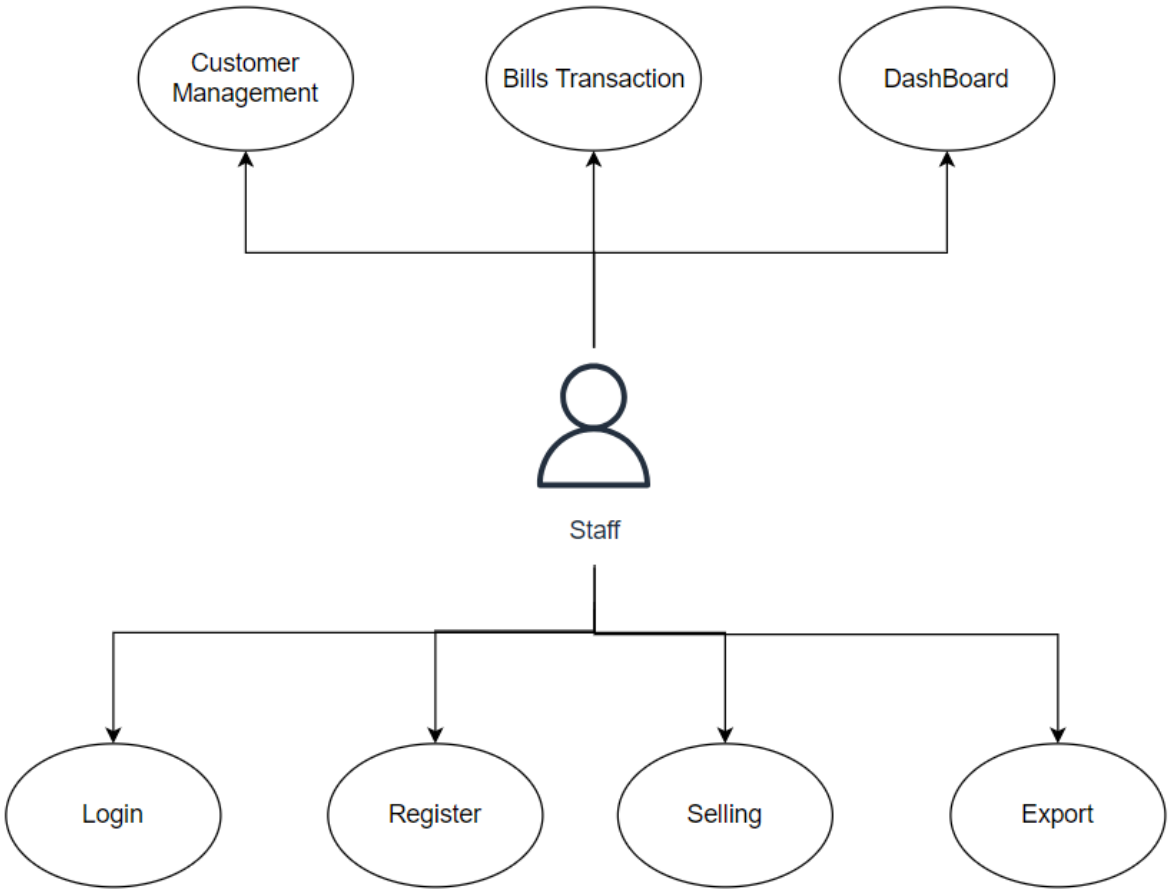
TASK SHEET REVIEW 1

Project: Pharmacy			**Date of preparation of activity plan:**			
#	**Task**	**Prepared by**	**Start date**	**Actual Date**	**Team member name**	**status**
1	Acknowledgement	Huy	27/05/2024	31/05/2024	Huy	completed
2	Introduction	Huy	27/05/2024	31/05/2024	Huy	completed
3	Problem definition	Huy	27/05/2024	31/05/2024	Huy	completed
4	Customer requirement specification	Huy	27/05/2024	31/05/2024	Huy	completed
5	Scope of work	Huy	27/05/2024	31/05/2024	Huy	completed
6	Hardware/software requirement	Huy	27/05/2024	31/05/2024	Huy	completed
7	Task Sheet	Huy	27/05/2024	31/05/2024	Huy	completed
Review			Signature of instructor			
			Mr. Pham Cong Danh			

REVIEW 2

1 .USECASE

1.1 Use case for Staff



Login

Use case name	Login	
Actors	User	
Description	Staff who has registered an account can login	
Requirements	Staff provides username and password	
Pre-conditions	N/A	
Post-conditions	Success: Staff is logged in to application	
	Fail: Alert and refill the information	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click 'Login' button on Home Page 3. Actors input Email and Password, then click the 'Login' button	2. System redirects to Login Page with the following controls: - "Email" text field - "Password" text field - "Login" button 4. System checks the information 5. System redirects to Home page
Exceptions	**Actor's actions:**	**System's responses:**
	1. Actors input invalid email or password.	2.System redirects to Login page with the following controls: - "Email" text field - "Password" text field - "Login" button 3.System shows message: "Invalid email and password"

Register

Use case name	Export	
Actors	User	
Description	Create new staff account	
Requirements	Have a phone email to receive OTP when forgot password	
Pre-conditions	N/A	
Post-conditions	Success: Create a new account and return to login page	
	Fail: Alert and refill information	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click 'Sign Up" Button under the "Login" button	2.The systems show the form which force actors fill in like ID, Full name, Password, Confirm Password, Phone, Emai, Address, Role (set Staff default)
	3. Actors fill in the form and click the "Sign Up"	4. Systems check whether ID, Phone, Email exist. 5. System inserts the status into database 6. System shows success message and return to Login page
Exceptions	**Actor's actions:**	**System's responses:**

Selling

Use case name	Selling	
Actors	Staff	
Description	Staff can sell product form storage	
Requirements	Input name of medicine	
Pre-conditions	N/A	
Post-conditions	Success: Show information that matches search phrase	
	Fail: No information is shown	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors input search phrase 3.Actors click information in list or view everything that matches search phrase 5. Actors click the products and click sell	2.System shows list of information that match the search phrase 4. Systems show bill in right table 6. System update products to SQL
Exceptions	**Actor's actions:**	**System's responses:**
	1.Actors enter is not wrong information	2.The system does not show the search information

Export:

Use case name	Export	
Actors	Staff	
Description	Export the data file to be printed to PDF	
Requirements	Data is not null	
Pre-conditions	N/A	
Post-conditions	Success: Export PDF to your computer	
	Fail: N/A	
Basic flow	**Actor's actions:**	**System's responses:**
	1.Actors click the "Export" button	2.System export to your computer
Exceptions	**Actor's actions:**	**System's responses:**
	1. N/A	2. N/A

Customer Management

Use case name	Customer Management	
Actors	Employee	
Description	Employee can manage all customer	
Requirements	N/A	
Pre-conditions	N/A	
Post-conditions	Success: Show up all customer's information	
	Fail: Actor can't do anything in the page	
Basic flow	**Actor's actions:**	**System's responses:**
	1.Actors click Customer Management button sidebar	2.System display the management page
Exceptions	**Actor's actions:**	**System's responses:**

	N/A	N/A

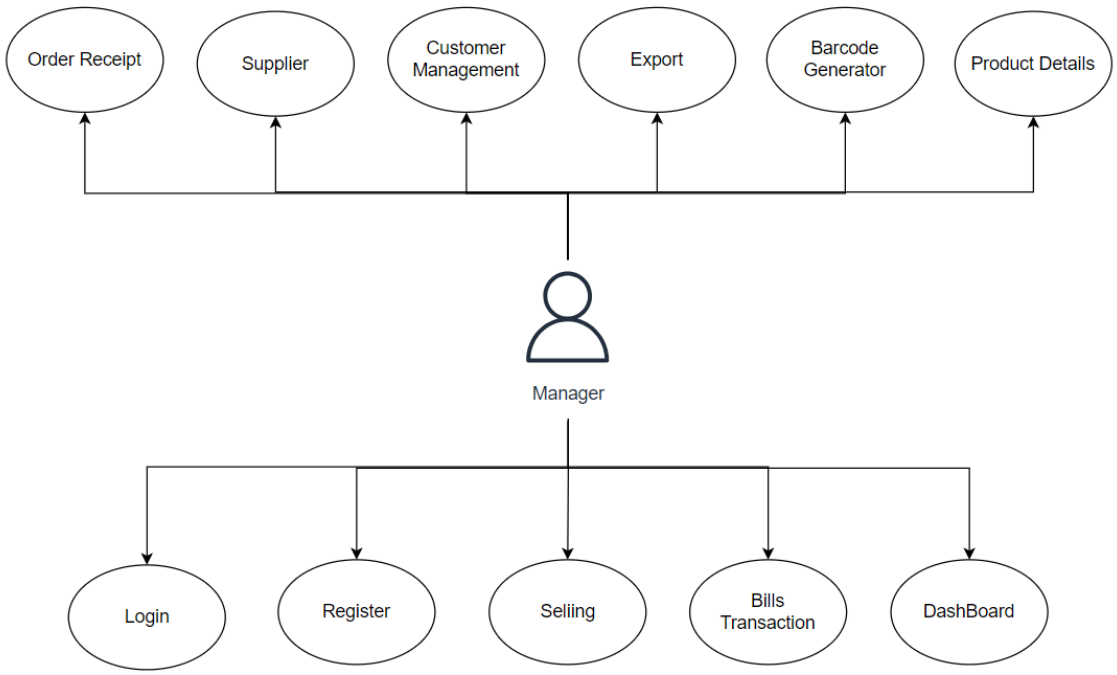
Bills Transaction

Use case name	Bills Transaction	
Actors	Staff	
Description	Manage bills	
Requirements	N/A	
Pre-conditions	N/A	
Post-conditions	Success: System shows the data	
	Fail: No information shown	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click on "Bills Transaction" button.	2.System shows Bills ID, Customer Name, Phone, Price, Method, Date
Exceptions	**Actor's actions:**	**System's responses:**
	1. Actors double click the bill	2. System shows clearly about that bill

Dashboard

Use case name	Dashboard	
Actors	Staff	
Description	See all information show in Home Page	
Requirements	N/A	
Pre-conditions		
Post-conditions	Success: N/A	
	Fail: N/A	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors log in successfully	1.System redirects to Home page
Alternative flow	**Actor's actions:**	**System's responses:**
	N/A	N/A
Exceptions	**Actor's actions:**	**System's responses:**
	N/A	N/A

1.2 Use case for Managers



Order Receipt

Use case name	Order Receipt	
Actors	Admin	
Description	Admin order product	
Requirements	N/A	
Pre-conditions	N/A	
Post-conditions	Success: System show up bills(Receipt ID, Order Date, Receive Date, Supplier, Total, Status) and can delete	
	Fail: N/A	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click 'Order Receipt" button 3. Admin click "Add New" button and fill the form in right table and click "+ " button	2. System show up data 4. System will update to database
Exceptions	**Actor's actions:**	**System's responses:**

Supplier

Use case name	Supplier	
Actors	Admin	
Description	Show data and add new supplier	
Requirements	N/A	
Pre-conditions	N/A	
Post-conditions	Success: N/A	
	Fail: N/A	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors choose "Supplier" button 3. Actors choose "Add New Supplier" 5. Actors click "Add" button	2. Systems show up the information products 4. System show the form with Name Address Phone Email 6. System checks the information and add to database
Exceptions	**Actor's actions:**	**System's responses:**
	N/A	N/A

Barcode Generator

Use case name	Barcode Generator	
Actors	Admin	
Description	Create barcode image and save to file	
Requirements	N/A	
Pre-conditions	N/A	
Post-conditions	Success: N/A	
	Fail: N/A	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click "Barcode Generator" button on Product Details 3. Actors click "Save"	2. System show the folder to save image 4. System will save to your computer
Exceptions	**Actor's actions:**	**System's responses:**
	N/A	N/A

Product Details

Use case name	Product Details	
Actors	Admin	
Description	See all full information of medicine	
Requirements	N/A	
Pre-conditions	N/A	
Post-conditions	Success: All information will show up	
	Fail: N/A	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click "Product Details" button	2. System show up
Exceptions	**Actor's actions:**	**System's responses:**
	N/A	N/A

Staff

Use case name	Staff	
Actors	Admin	
Description	See all information about employees	
Requirements	N/A	
Pre-conditions	N/A	
Post-conditions	Success: Display all Staff information	
	Fail: N/A	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click "Staff" button 3. Actors fill in form under the table information 5. Actors click Add button	2. System show up 3. System show the database add new like: ID, Name, Gender, Phone, Address, Gmail, Birthday, Role and Status 6. System save and update to database
Exceptions	**Actor's actions:**	**System's responses:**
	1. N/A	2. N/A

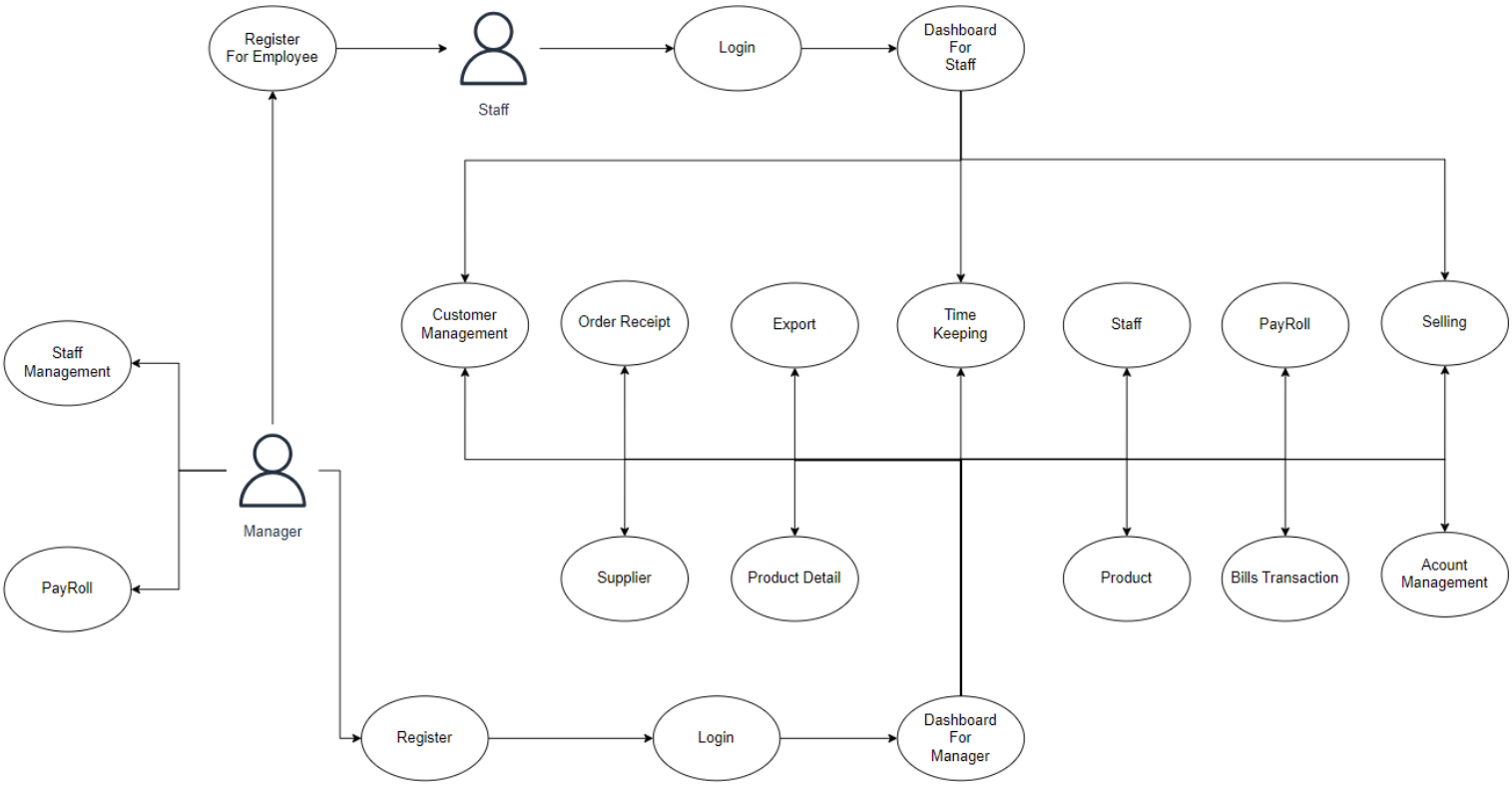
Payroll Management

Use case name	Payroll Management	
Actors	Admin	
Description	Payroll Management	
Requirements	admin	
Pre-conditions	Admin can see all information and can change it	
Post-conditions	Success: the systems show up the table view	
	Fail: Recheck	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click Fund Management button on tab 3. Actors select name of fund to see	2. System redirects to search fund section 4. System checks the information 5. System returns the result
Exceptions	**Actor's actions:**	**System's responses:**
	N/A	N/A

Time Keeping

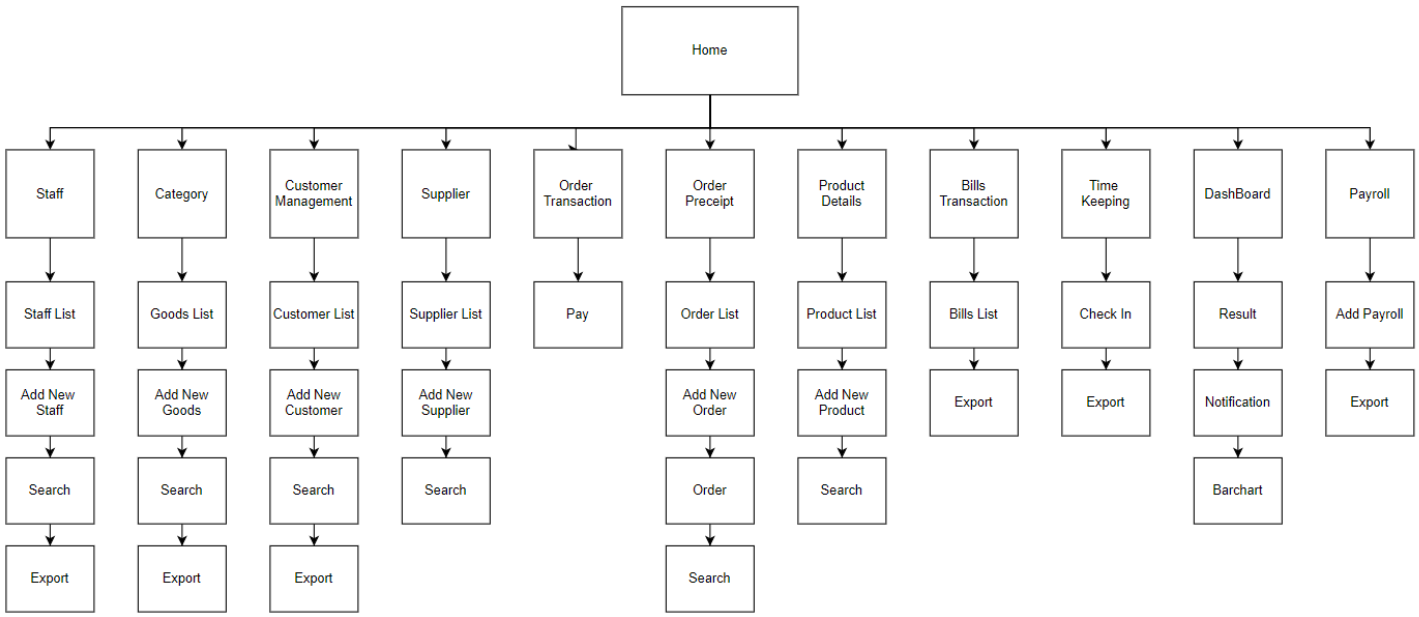
Use case name	Time Keeping	
Actors	Admin	
Description	See check in check out	
Requirements	Admin	
Pre-conditions	Admin can see all process checked of staff	
Post-conditions	Success: N/A	
	Fail: N/A	
Basic flow	**Actor's actions:**	**System's responses:**
	1. Actors click Time Keeping button 3. Actors click Check In	2. System show the table with ID, Staff ID, Name, Day, Check in, Check out 4. System get date time and update to database
Exceptions	**Actor's actions:**	**System's responses:**
	N/A	N/A

2.Data Flow Diagram (DFD)

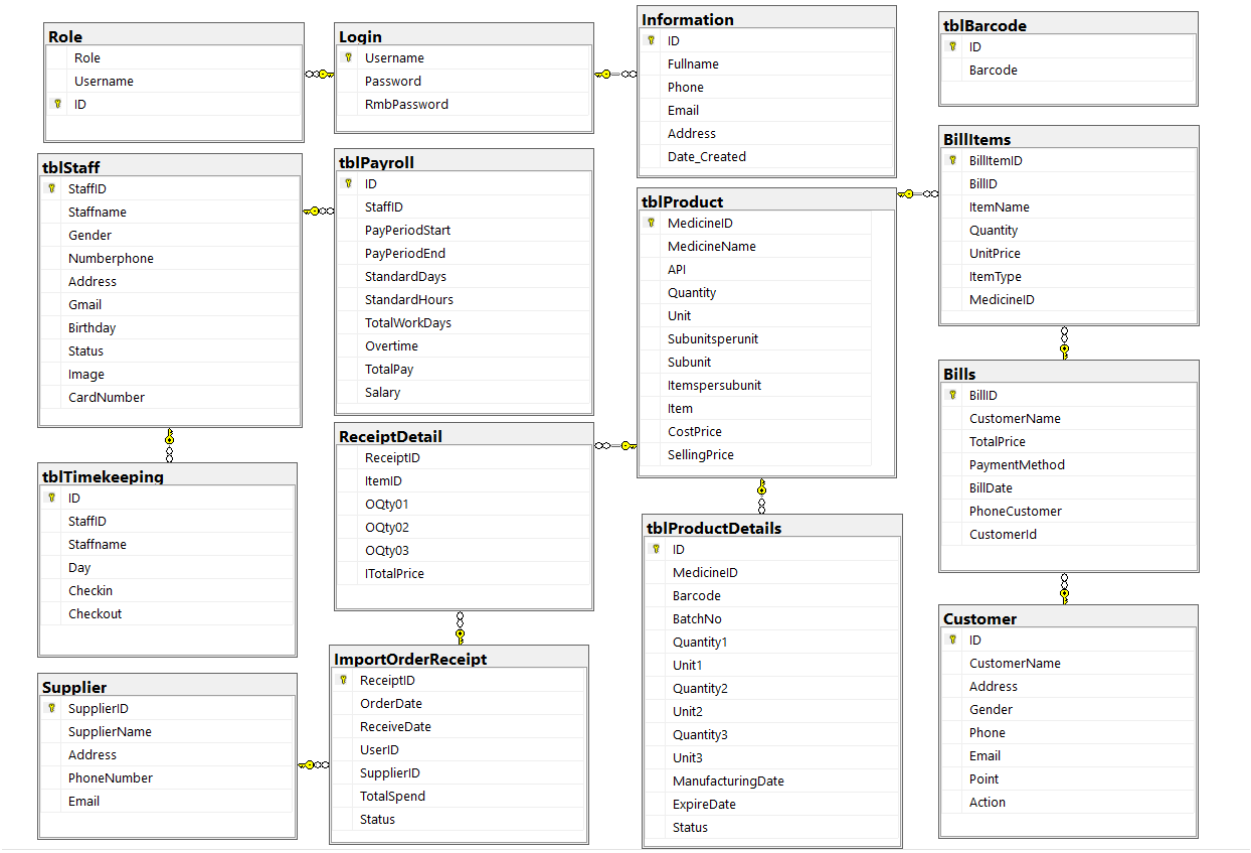


SITE MAP

1. User



ENTITY RELATIONSHIP DIAGRAM (ERD)



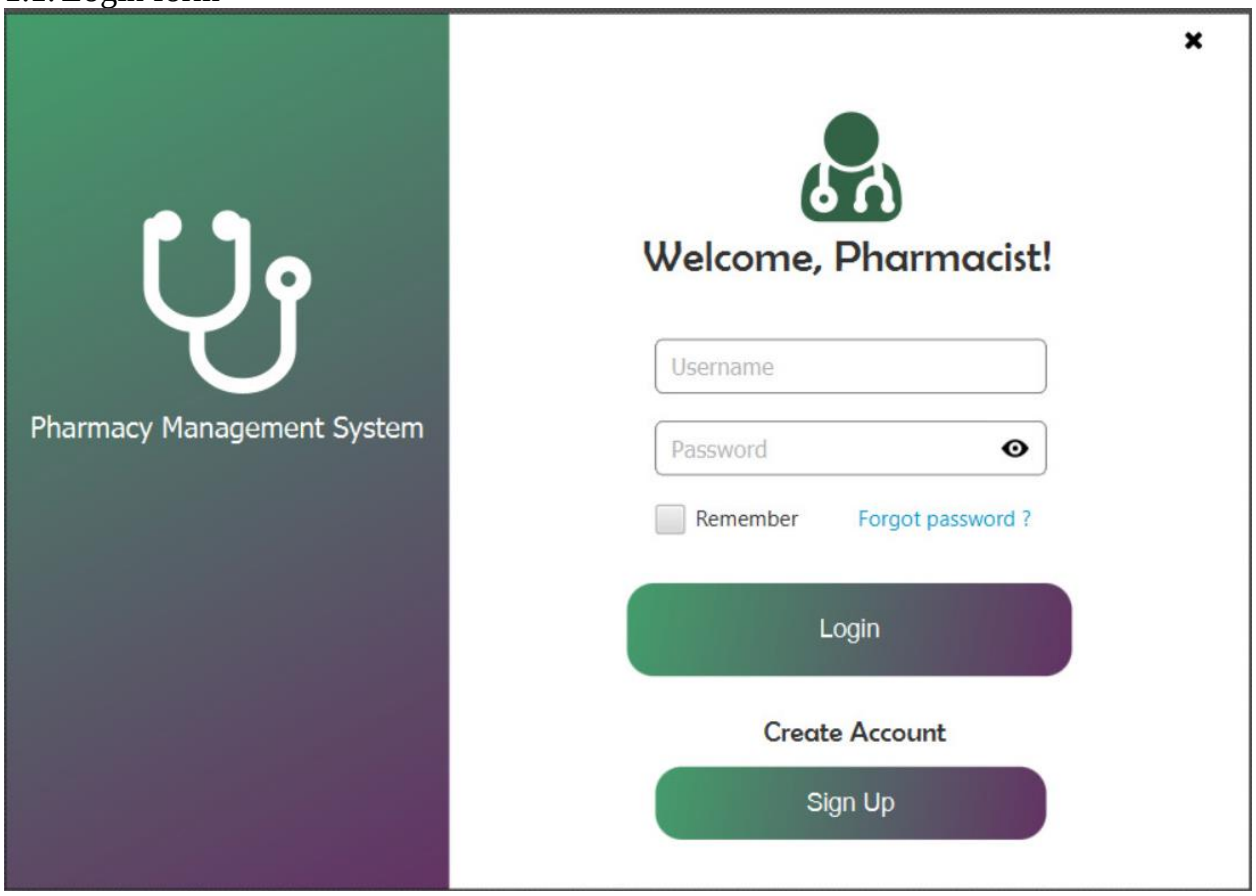
TASK SHEET REVIEW 2

<table>
<tr><th colspan="3">Project: Pharmacy</th><th colspan="4">Date of preparation of activity plan:</th></tr>
<tr><th>#</th><th>Task</th><th>Prepared by</th><th>Start date</th><th>Actuals Date</th><th>Team member name</th><th>status</th></tr>
<tr><td>1</td><td>Use case</td><td rowspan="4">Hoang Gia Huy
&
Nguyen Anh Minh
&
Nguyen Anh Quan
&
Tran Nhat Linh
&
Doan Duc Do</td><td>31/05/2024</td><td>07/06/2024</td><td rowspan="4">Hoang Gia Huy
&
Nguyen Anh Minh
&
Nguyen Anh Quan
&
Tran Nhat Linh
&
Doan Duc Do</td><td>completed</td></tr>
<tr><td>2</td><td>Data Flow Diagram (DFD)</td><td>31/05/2024</td><td>07/06/2024</td><td>completed</td></tr>
<tr><td>3</td><td>Site Map</td><td>31/05/2024</td><td>07/06/2024</td><td>completed</td></tr>
<tr><td>4</td><td>Entity Relationship Diagram (ERD)</td><td>31/05/2024</td><td>07/06/2024</td><td>completed</td></tr>
<tr><td colspan="3">Review</td><td colspan="4">Signature of instructor</td></tr>
<tr><td colspan="3"></td><td colspan="4"></td></tr>
<tr><td colspan="3"></td><td colspan="4">Mr. Pham Cong Danh</td></tr>
</table>

REVIEW 3
GUI DESIGN

1. User side

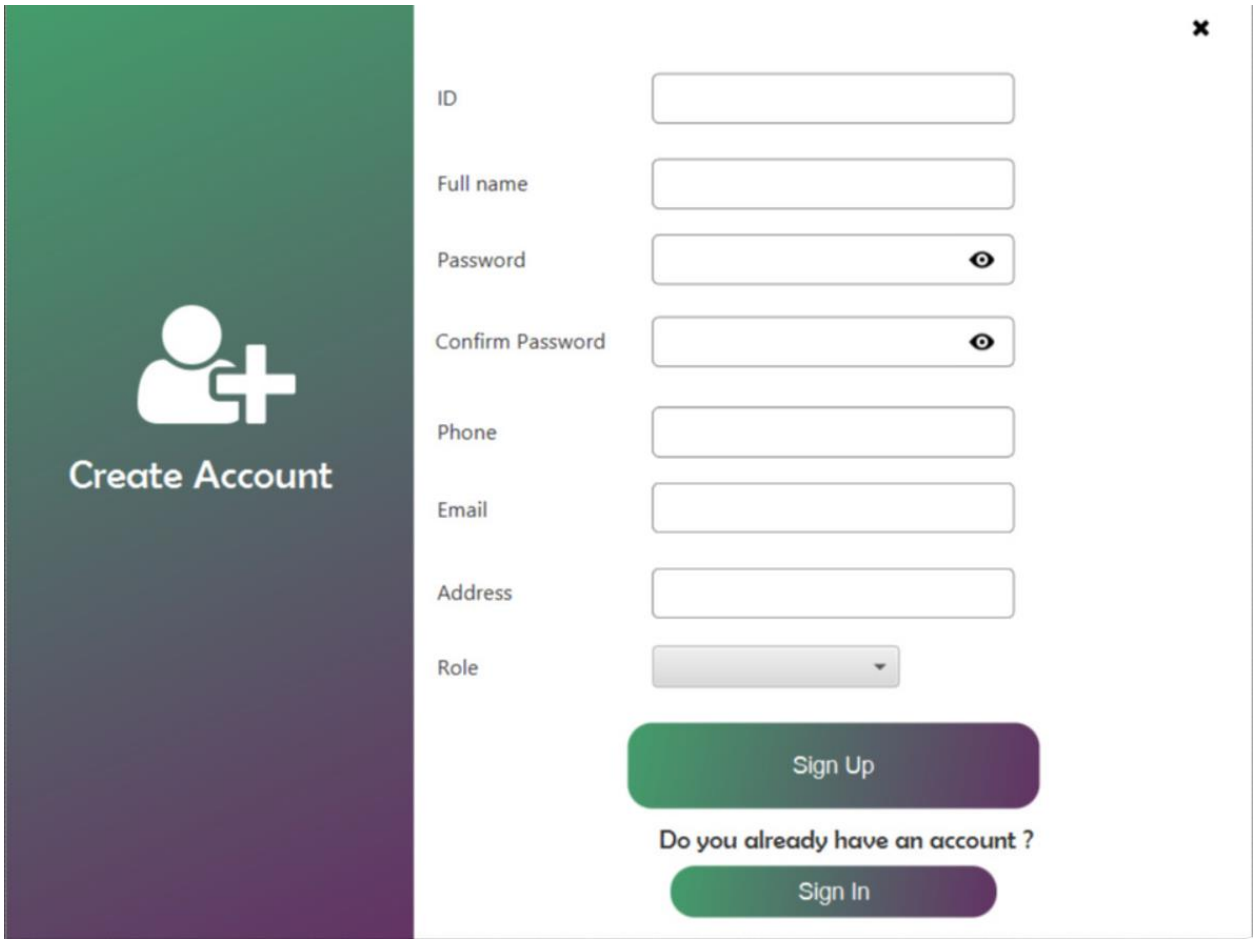
1.1. Login form



Users will fill in information log in:

+ Text field Username
  Validate: "Name is required!", "User does not exist!"
+ Text field Password
  Validate: "Password is required"
  "Wrong password"
+ Icon Eye/Eye Close: Show/Hide Password
+ Checkbox Remember: Remember password for the next sign in
+ Link "Forgot password": Click to go Reset password page
+ Login Button: Complete fill in and start validation to sign in
+ Sign Up Button: Click to go Register Staff Account page

1.2. Register page

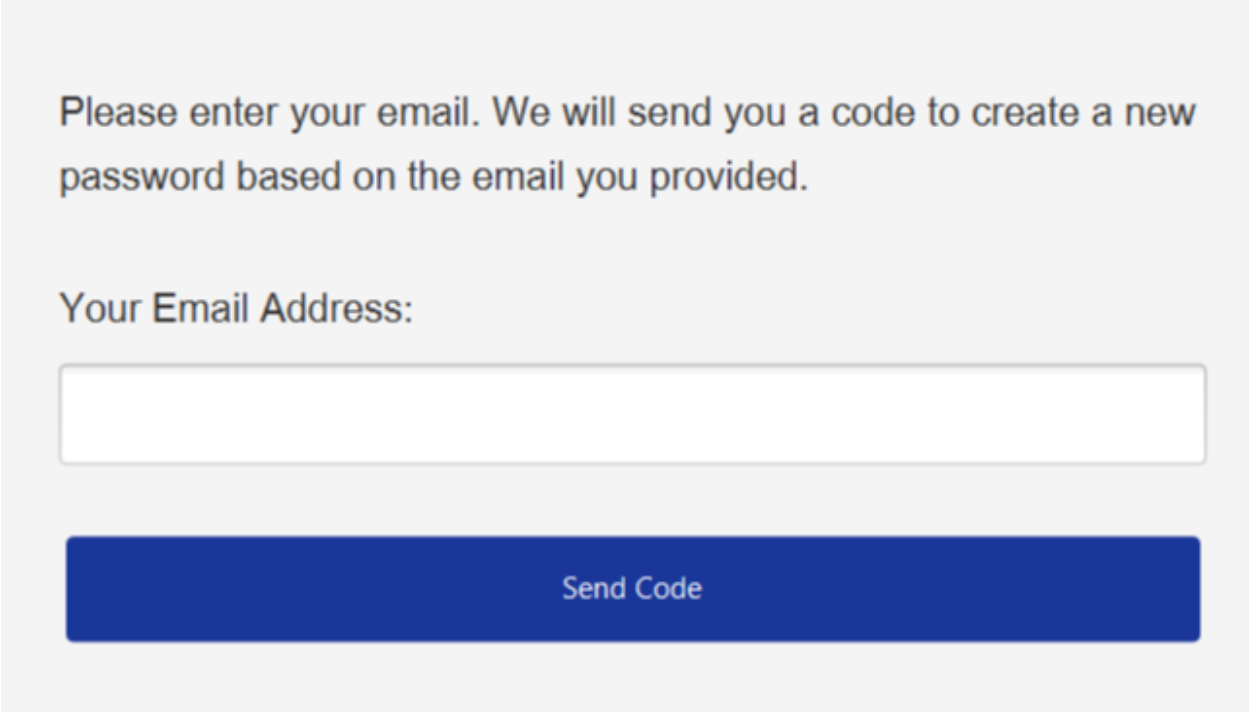


Users fill full information:

+ Text field String Name
    Validate: “Name is required”, “Name must not be longer than 30 characters”
+ Text field String Email (Gmail is registered with the company)
    Validate: “Email is required”, validate with letter “@”
+ Text field String Address
+ Text field Integer Phone
    Validate: “Phone is required”, “Format error, enter number”
+ Text field String Password
    Validate: “Password is required”, “Wrong password!”
+ Role: Admin-Staff (Default Admin).
+ Then click the “Sign Up" button to complete the login process.

If you have an account, you click the “Sign In" button to comeback sign up page.

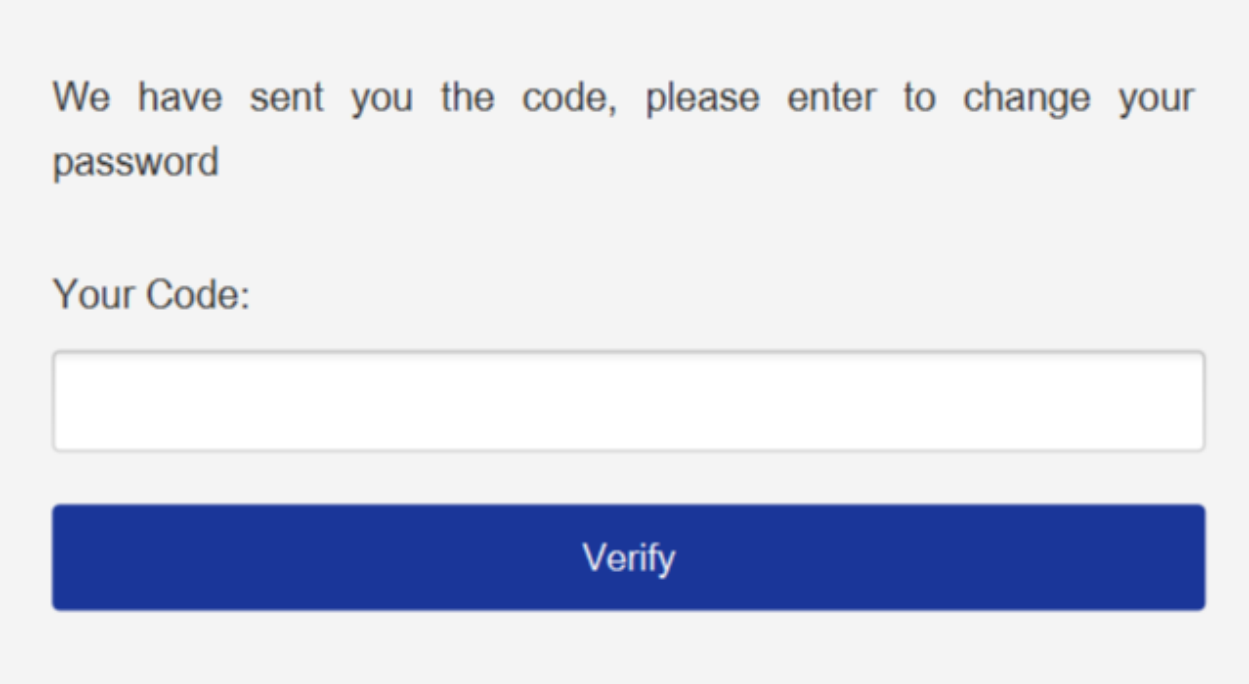
1.3. Forgot Password



+ User input email (which register before) to text field and click “Send Code” button to validate and receive OTP by email

Validate: “Email does not exist”, “Wrong email”, “Wrong format”

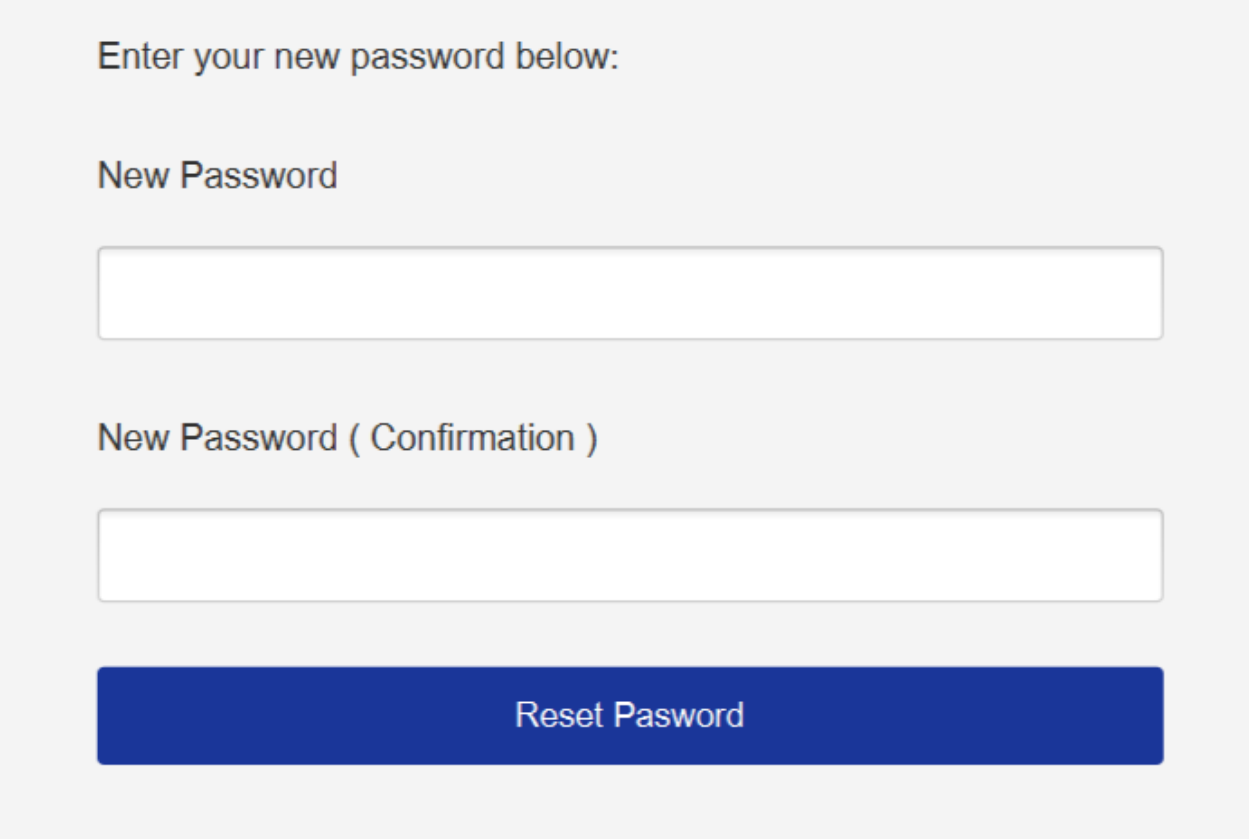
1.4. Verify OTP



+ After receive OTP email, user enter it to text field and click “Verify” button to validate

Validate: “Wrong OTP”

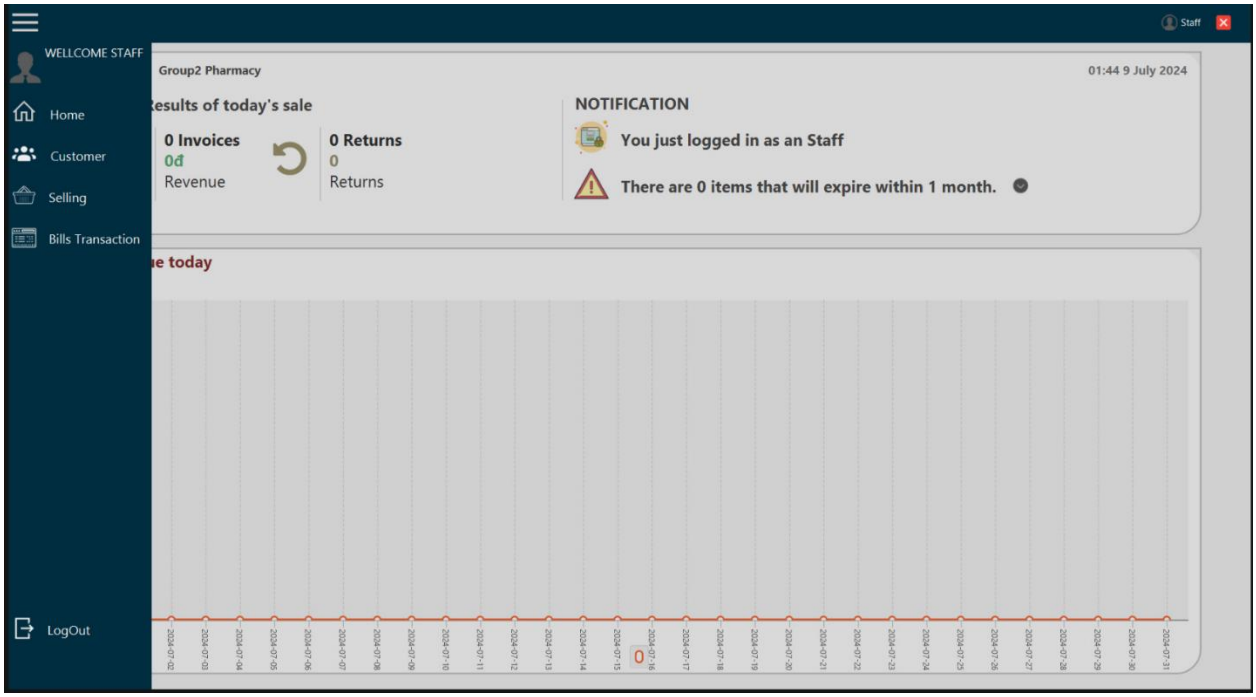
1.5. Reset Password



+ User input new password and confirm password and click “Reset Password” button to update new password into database and return to login page
Validate: “Password is not blank”, “Confirm Password is not blank”

1.6. Home

1.6.1. Dashboard for Staff:



The dashboard page displays:

-A tab bar in left side:

+Profile User with the title “Welcome Staff *Name of user*”

+ Button: Home to go Home page, Customer to go Customer Management Page, Selling to go Sell Page

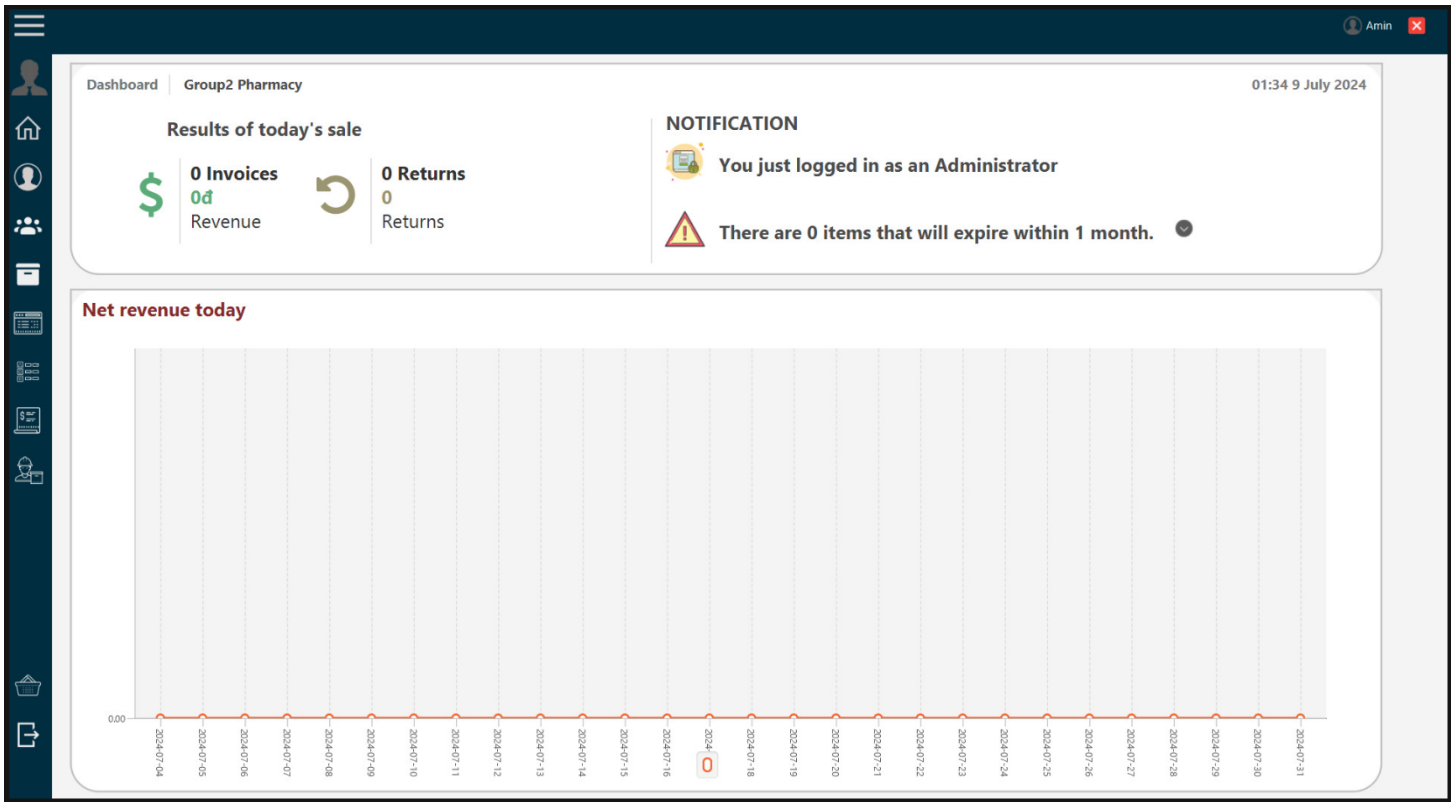
+ Log Out to sign out the application

- Dashboard in right side:

+ Above: show the report of the day, warning if sign up in unknow device, warning goods out of date…

+Below: displays a statistical data table

1.6.2. Dashboard for Admin:



The dashboard page displays:

-A tab bar in left side:

+Profile User with the title “Welcome Staff *Name of user*”

+ Button: Home to go Home page,
Customer to go Customer Management Page
Product go to Product Page
Transaction to go Transaction Page

+ Setting to custom your panel, color theme, account…

+ Log Out to sign out the application

+ Transaction Button: Click go to Transaction page

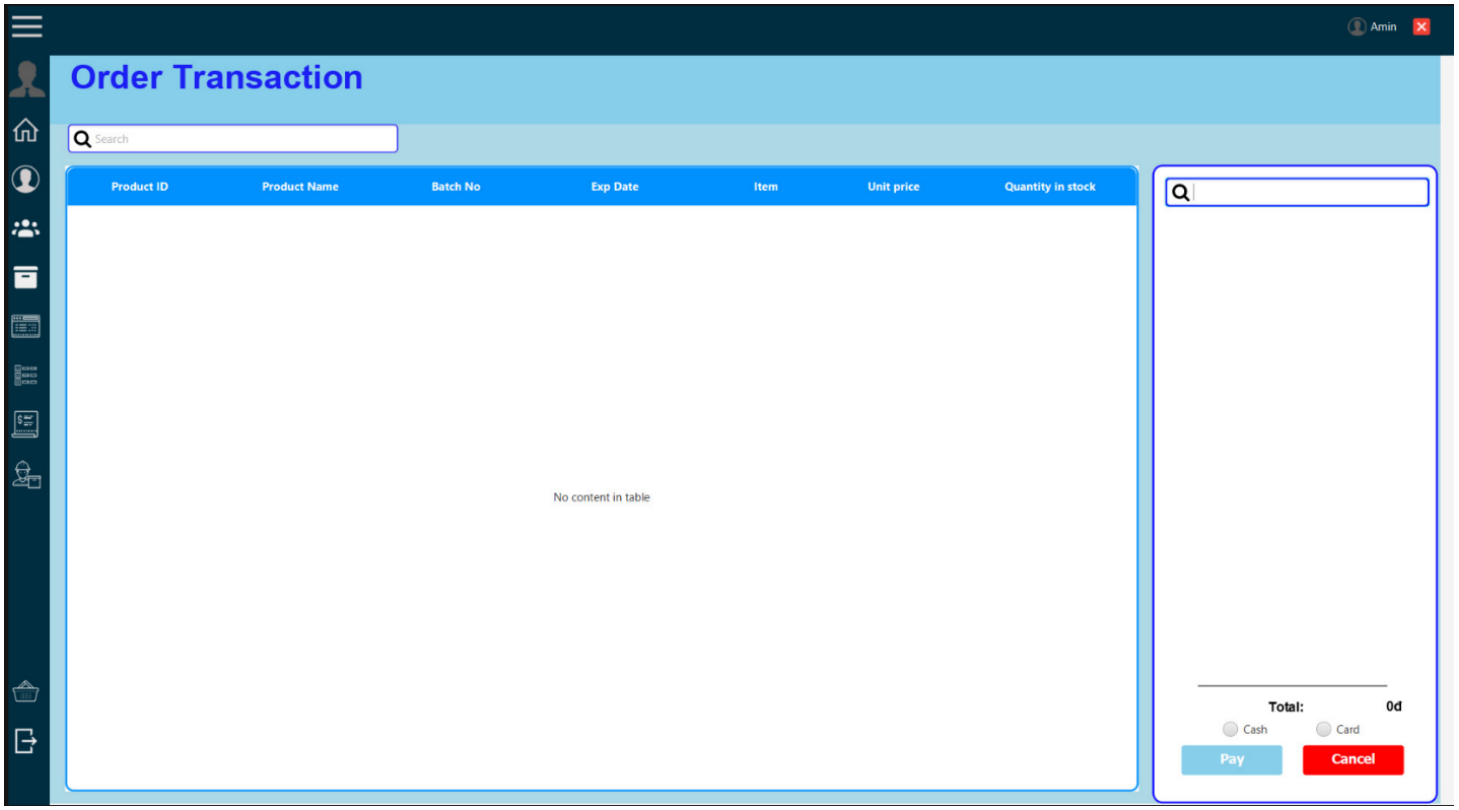
+ Account Management Button: Click to go to Account Management page

- Dashboard in right side:

+ Above: show the report of the day, warning if sign up in unknow device, warning goods out of date…

+Below: displays a statistical data table

1.6.3. Order Transaction



Search: + Search order by Name shown in table below in left side
+ Search customer by name or number in table below in right side

Users fill full information:

Integer Quantity

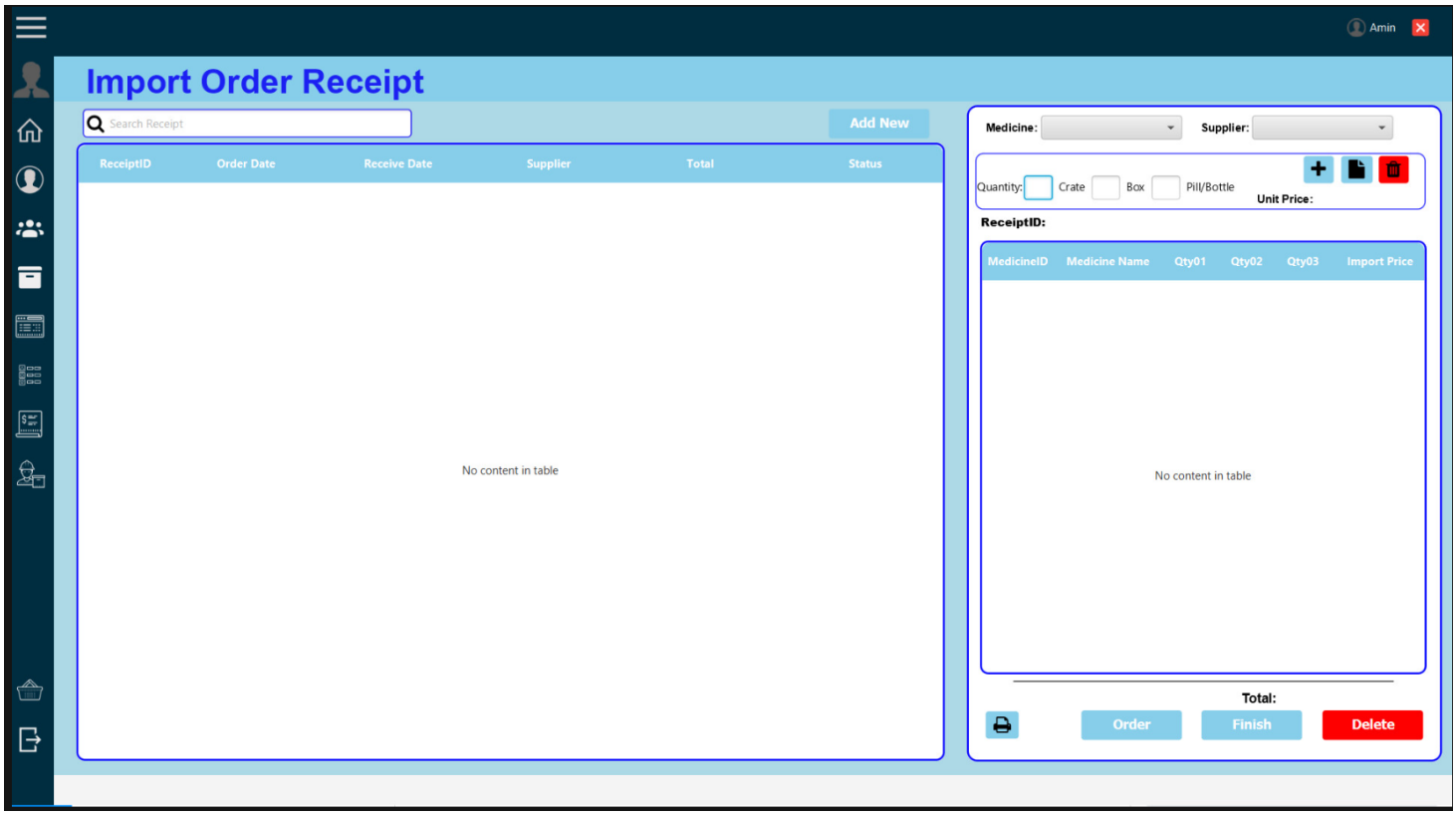
Integer Pill

Radio Choice Cash/Card to choose type of payment

Pay Button: to pay the order

Cancel Button: cancel process

1.7. Receipt Management



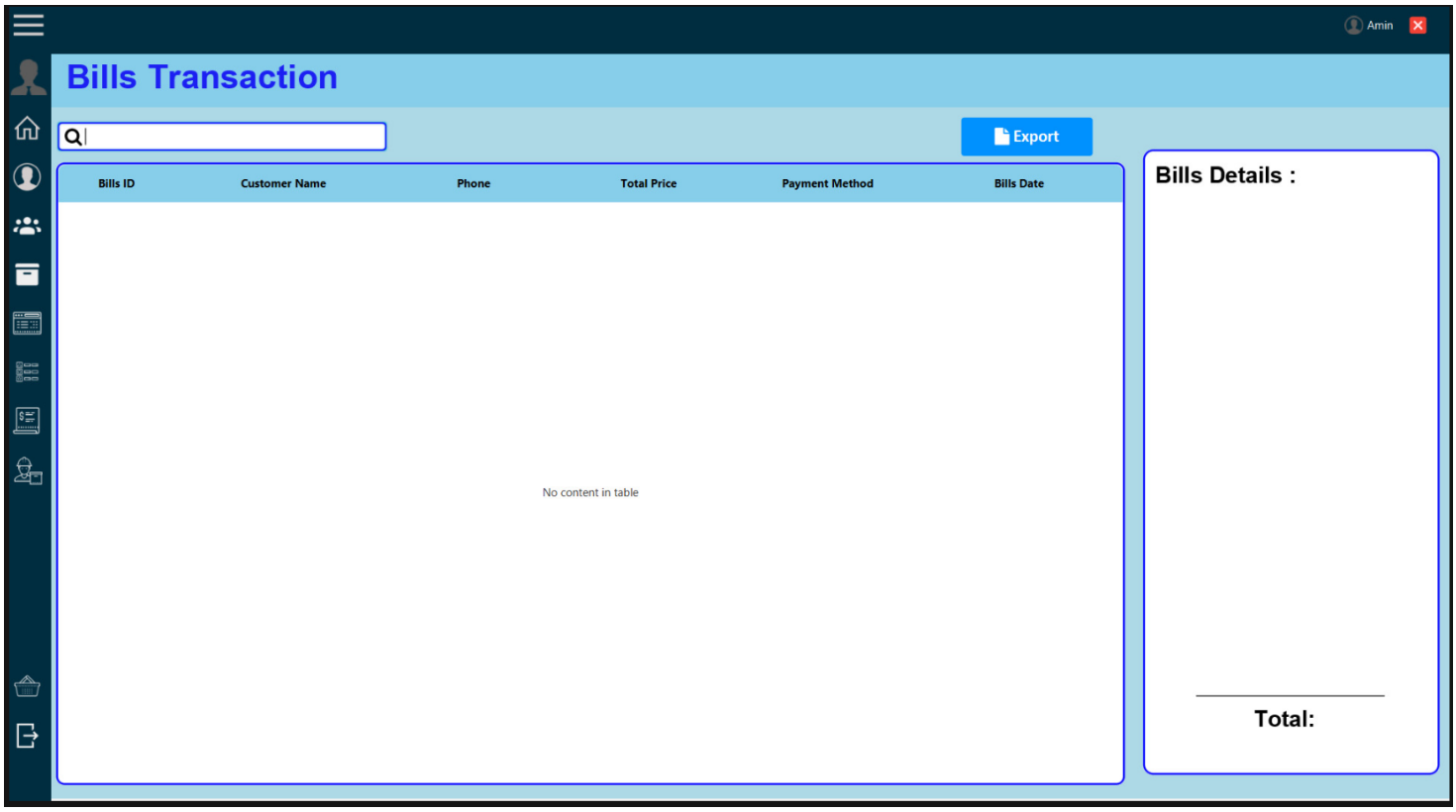
Search: + Search receipt by Name shown in table below in left side
+ Search bill by name or number in table below in right side

Users fill full information:

Integer Quantity
Integer Crate
Integer Box
Integer Pill/Bottle

Delete Button: Delete all product show in table
Print Button: Print a doc, excel… file to report
Cancel Button: Cancel process
Trash Icon: Delete the product
Order Button: Order receipt
Finish Button: Finish order process
Medicine Selection: Choose name of medicine
Supplier Selection: Choose name of supplier

2.1. Bills Transaction



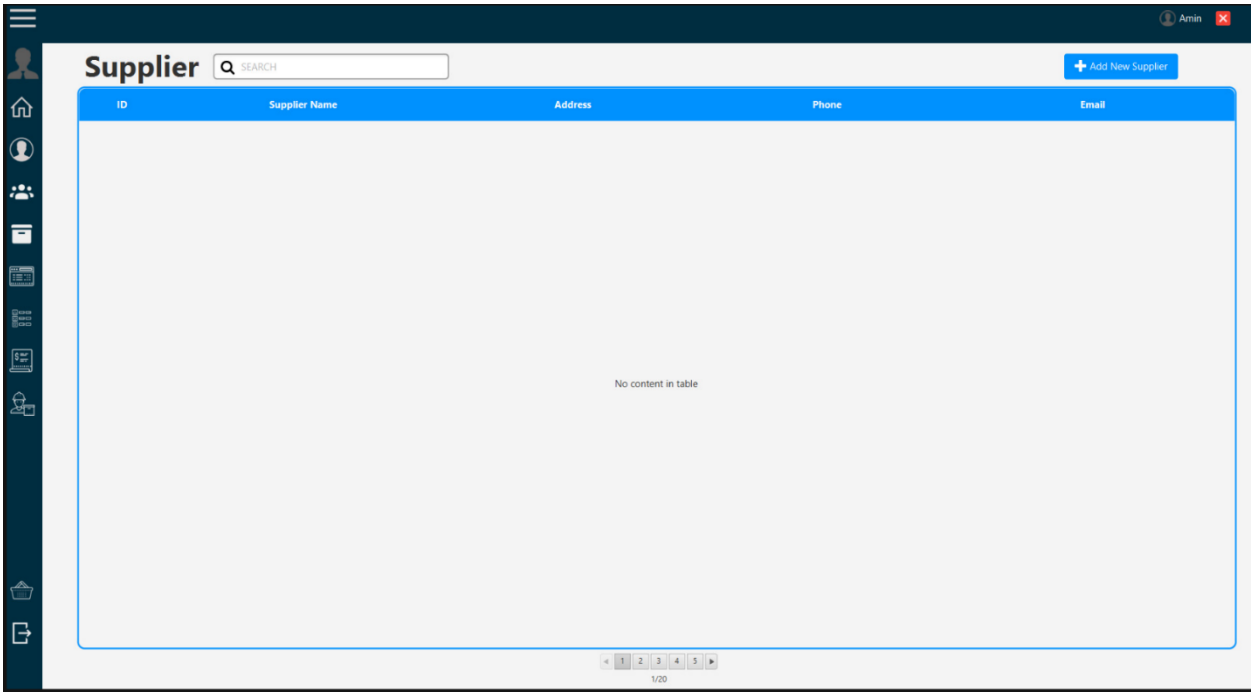
Search: + Search bills by Name shown in table below in left side
+ Search bill by Name or number in table belowin right side

Users fill full information:

Integer Quantity
Integer Crate
Integer Box
Integer Pill
Integer Batch No with can have a device support to scan

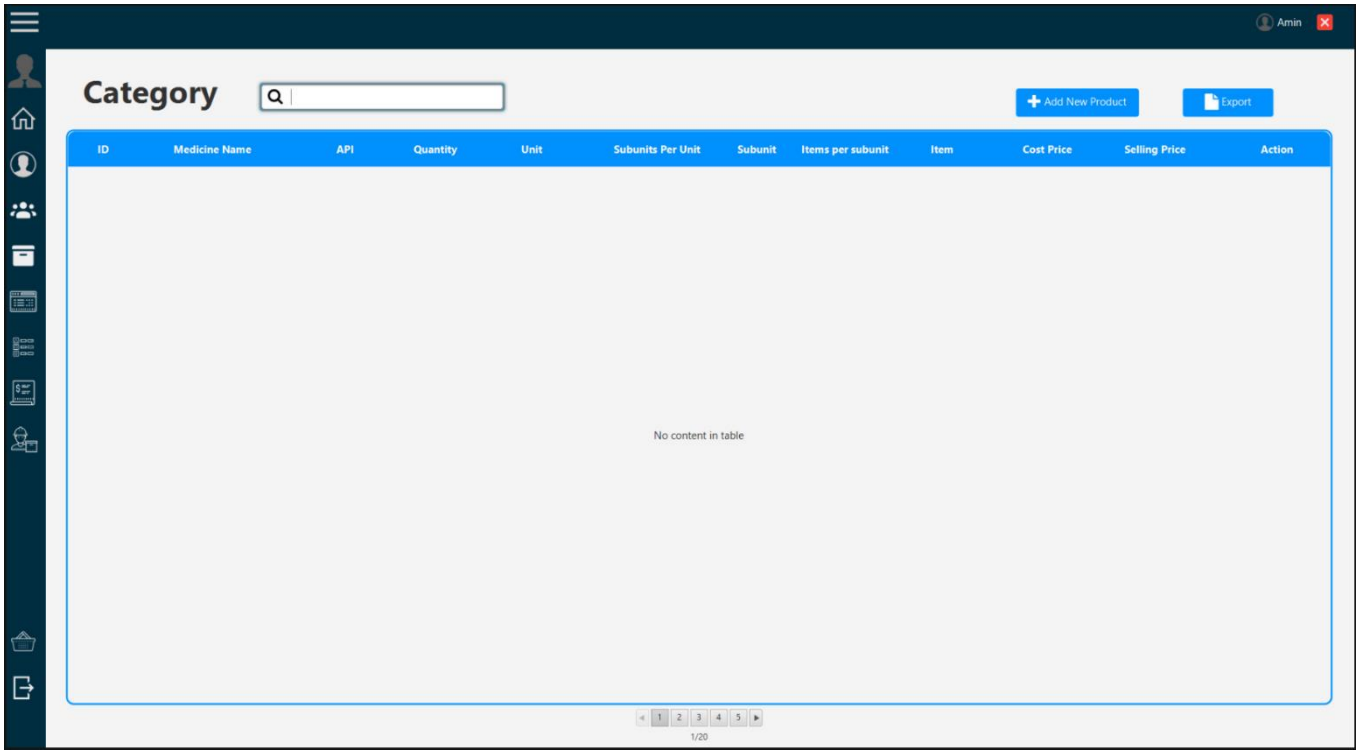
Print Button: Print a doc, excel… file to report

3.1. Supplier

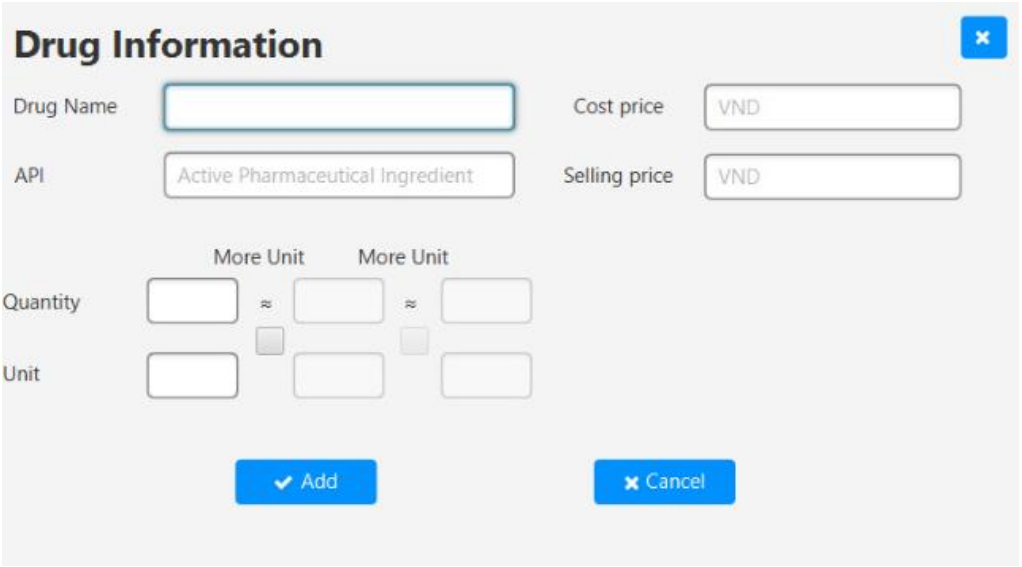


Search:	Search product by Name shown in table below
Table shows up full information:	ID Supplier Name Address Phone Email

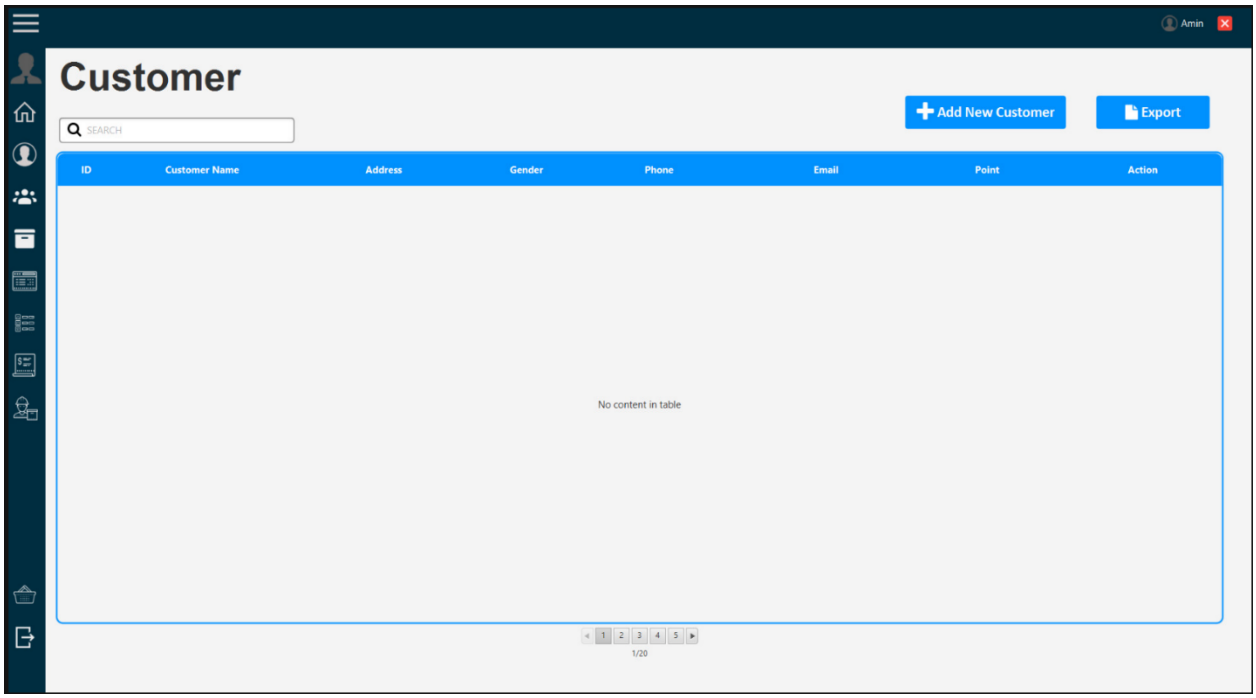
4.1. Category



Search:	Search product by Name shown in table below
Export Button:	Click to go to Export page
Add New Button:	Click to go to Add new form

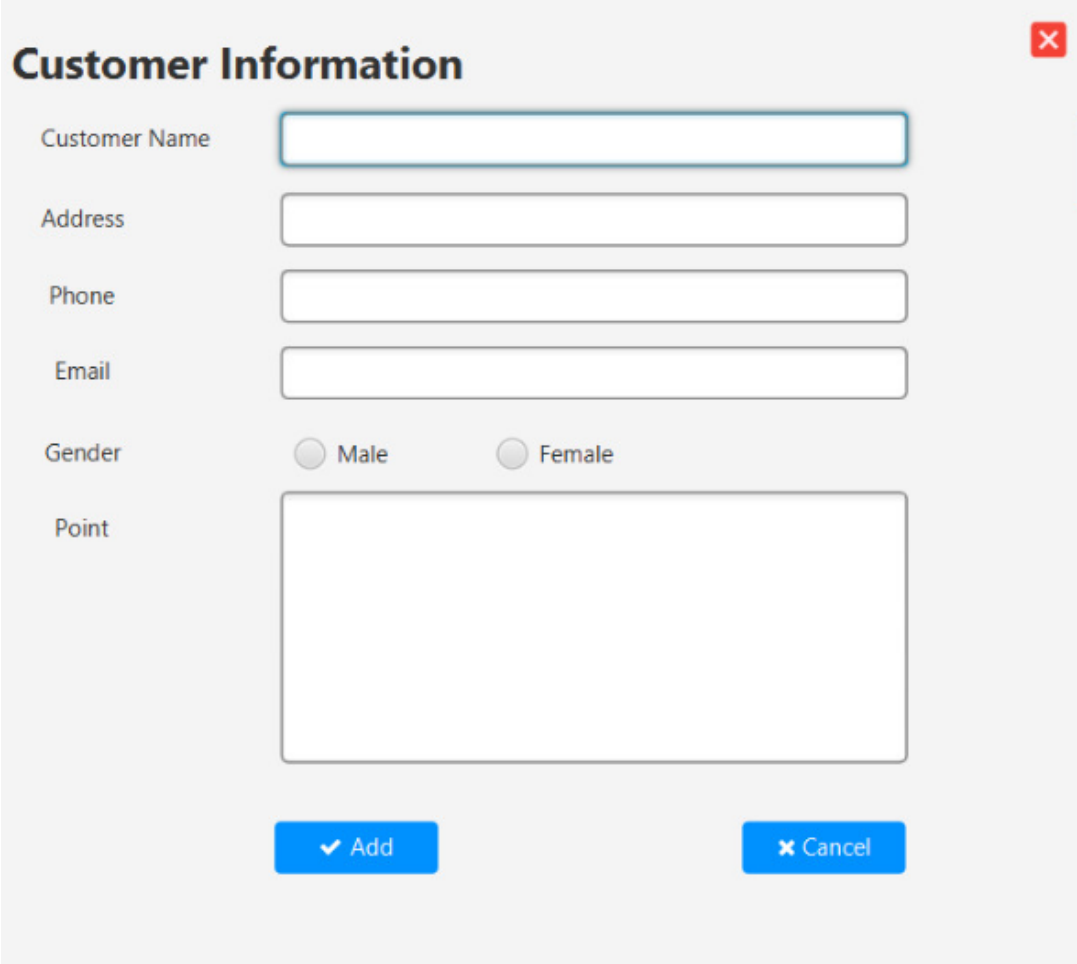


4.2. Customer Management



Search:	Search product by Name shown in table below
Add New Customer Button:	Click to go to Add new form

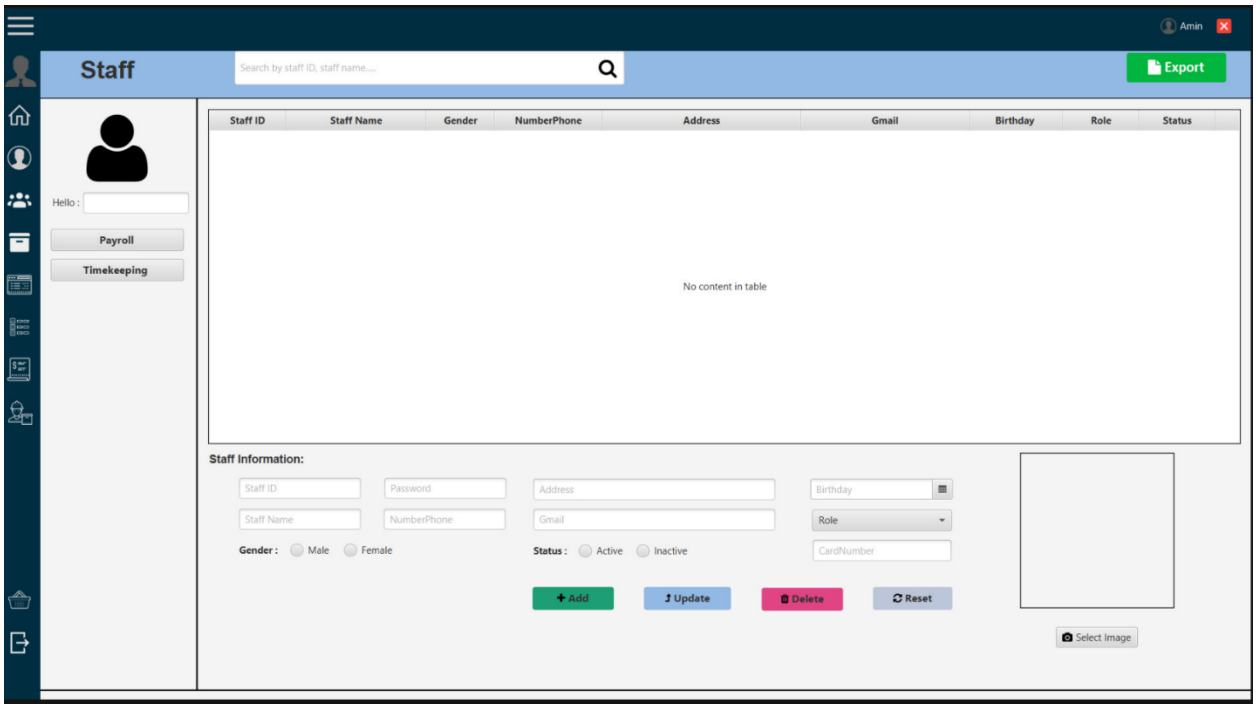
2.7.1.And new customer



Users fill full information below:

+ Text field Customer Name
  Validate: “Name must not be longer than 30 characters”
+ Text field Address
+ Text field integer Phone
  Validate: “Phone number must be entered in numbers”
+ Text field string Email (validate with @)
+ Choice Gender
  Validate: “Gender is not blank”
+ Text field string Note

4.3. Staff Management

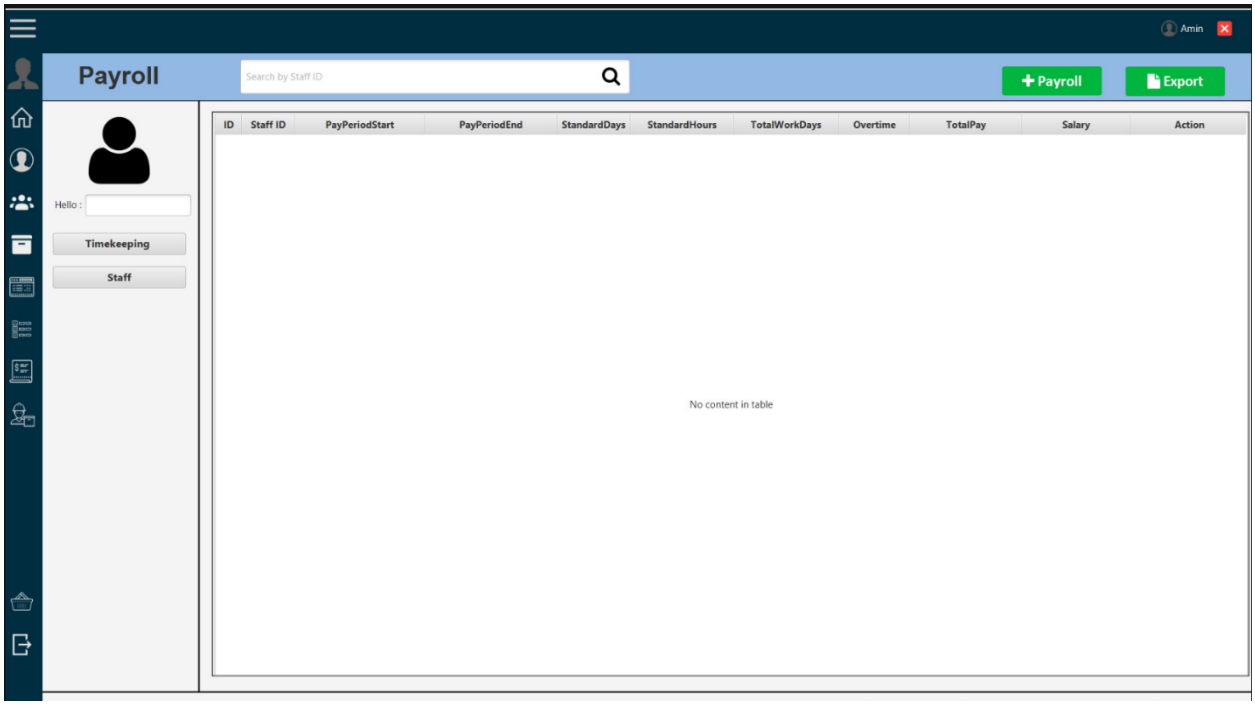


Search: + Search product by Name shown in table below
Payroll Button: Click to go Payroll page
Timekeeping Button: Click to go Timekeeping page
Export Button: Click to go Export page
User will fill full information:

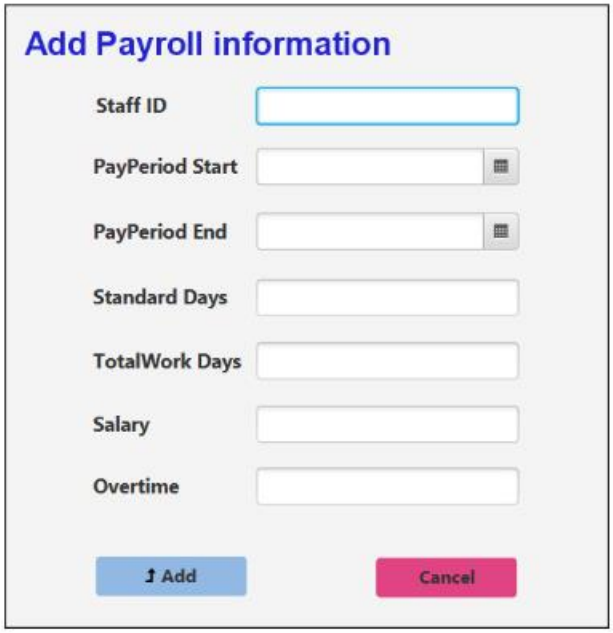
- Integer Staff ID: Validate: “NV*number*”
- String Password Validate: “Password is not blank”
- String Address
- Date picker Birthday
- String Staff Name Validate: “Name shorter than 30 characters”
- Integer Phone Validate: “Enter by number”
- String Gmail Validate with @
- Choice Role: Manager-Staff
- Choice Gender: Male-Female
- Choice Status: Active- Inactive
- Import Picture

Add Button: Complete addition
Update Button: Choose the staff’s information and change it
Delete Button: Delete info is chosen
Reset Button: Reset all text field

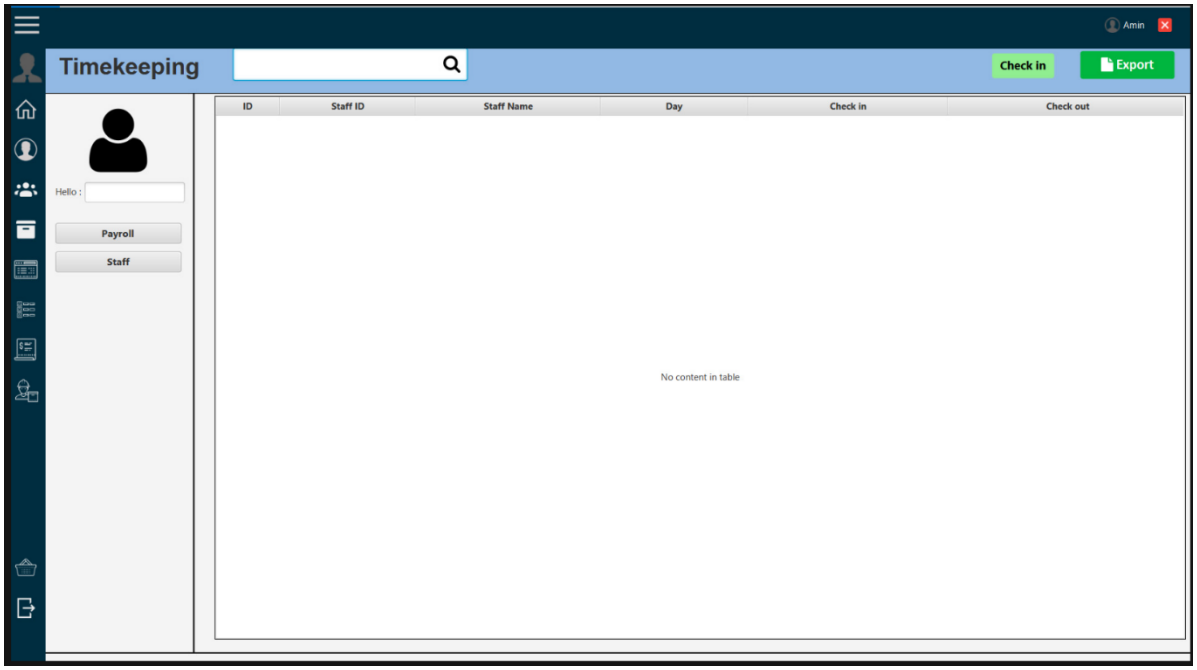
4.4. Payroll page



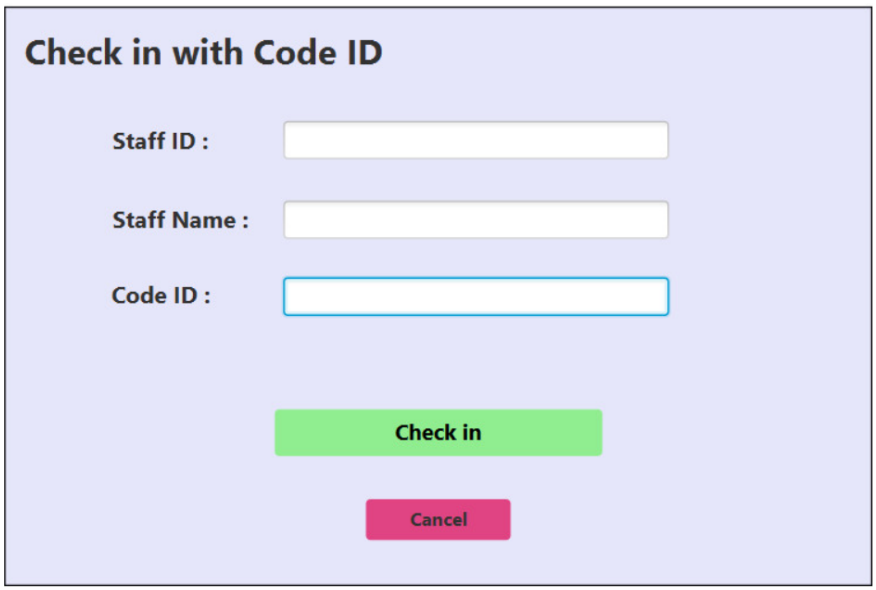
Search: Search product by Name shown in table below
Payroll Button:
Export Button: Click to go Export page
Timekeeping Button: Click to go Timekeeping page
Staff Button: Click to go Staff page



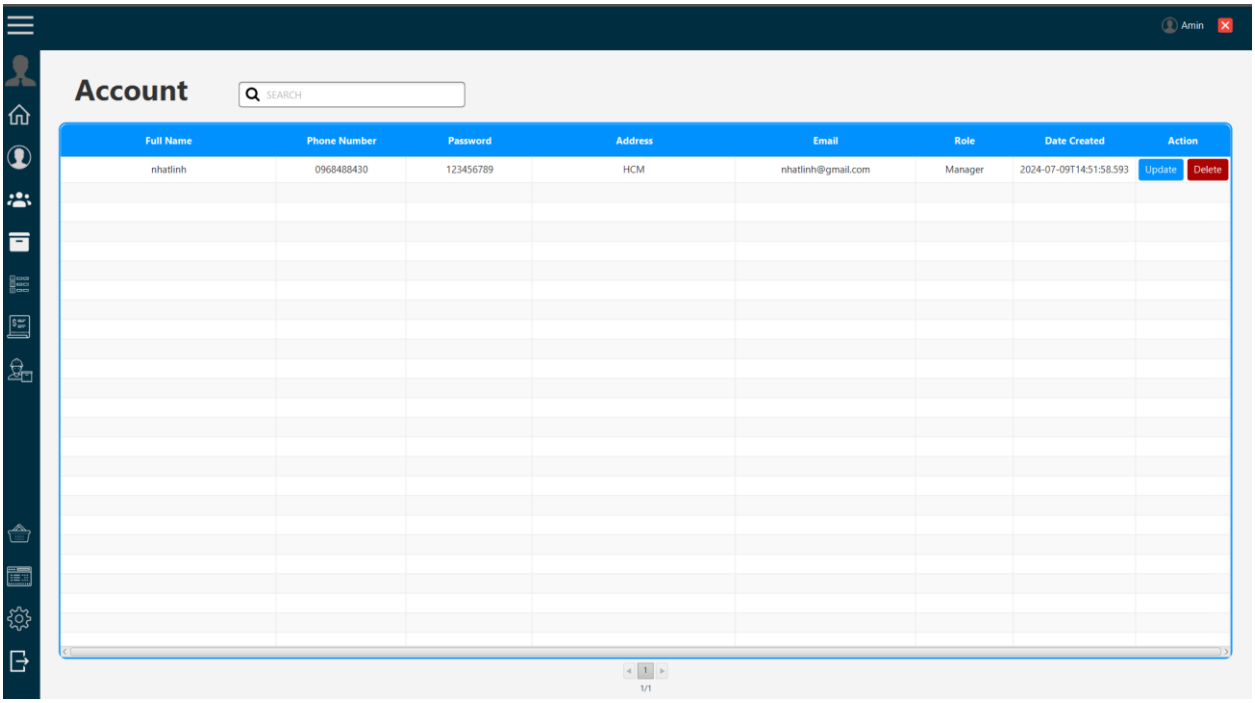
4.5. Time Keeping Page



Search:	Search product by Name shown in table below
Date picker Icon:	Choose the day to show
Export Button:	Click to go Export page
Staff Button:	Click to go Staff page



4.6. Account Management



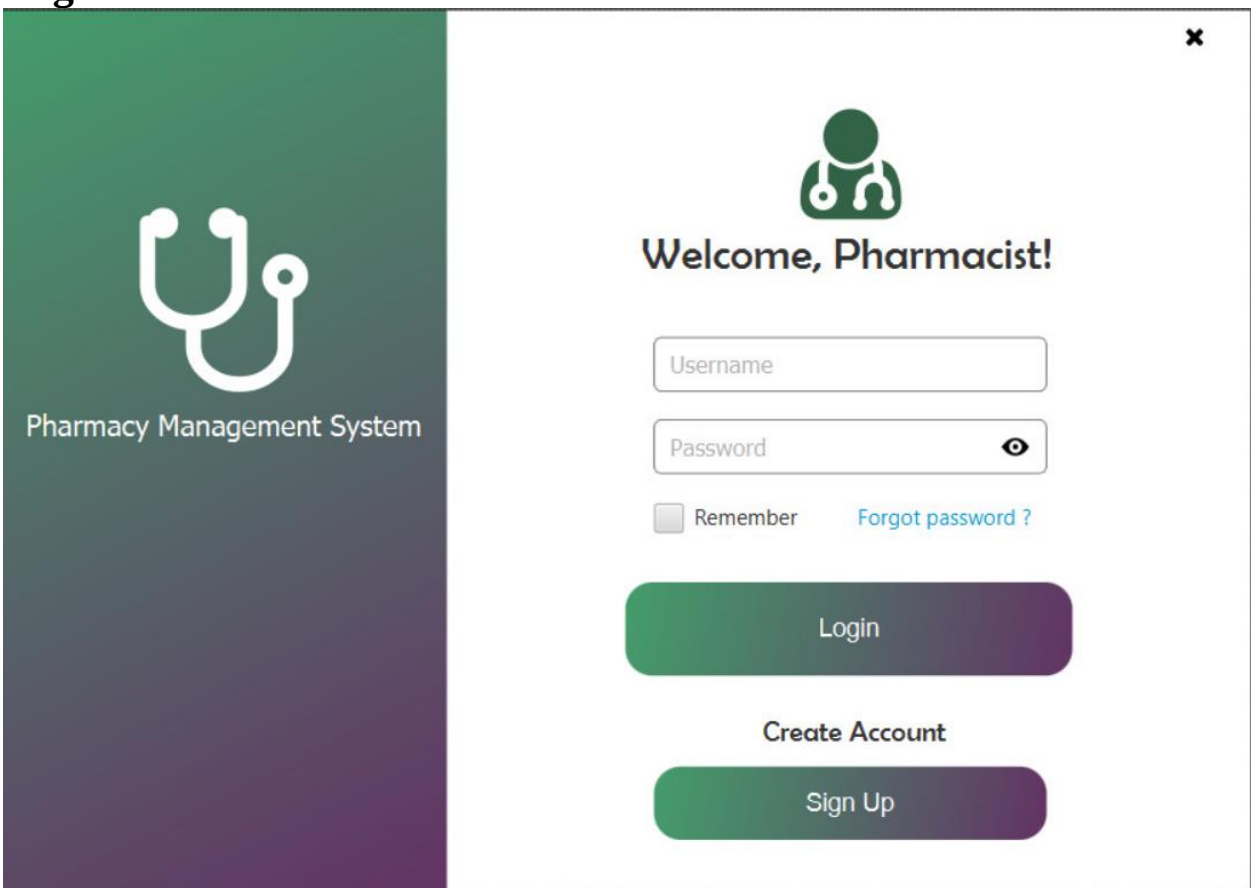
Search:	Search product by Name/Phone/Email shown in table below
Update Button:	Click to Update data selected
Delete Button:	Click to delete data selected

TASK SHEET REVIEW 3

Project: Pharmacy			**Date of preparation of activity plan:**			
#	**Task**	**Prepared by**	**Start date**	**Actuals Date**	**Team member name**	**status**
1	GUI DESIGN	Hoang Gia Huy & Nguyen Anh Minh & Nguyen Anh Quan & Tran Nhat Linh & Doan Duc Do	07/06/2024	17/6/2024	Hoang Gia Huy & Nguyen Anh Minh & Nguyen Anh Quan & Tran Nhat Linh & Doan Duc Do	completed
Review			Signature of instructor			
			Mr. Pham Cong Danh			

Coding Review 1

1.1 Login Form



1.1 Import

```
1  import demoproject1.FXMLNhanVienController;
2  import java.io.IOException;
3  import java.net.URL;
4  import java.util.ResourceBundle;
5  import javafx.beans.value.ObservableValue;
6  import javafx.fxml.FXML;
7  import javafx.fxml.FXMLLoader;
8  import javafx.fxml.Initializable;
9  import java.sql.Timestamp;
10 import java.time.Instant;
11 import javafx.scene.Scene;
12 import javafx.scene.control.Button;
13 import javafx.scene.control.CheckBox;
14 import javafx.scene.control.Hyperlink;
15 import javafx.scene.control.PasswordField;
16 import javafx.scene.control.TextField;
17 import javafx.scene.image.ImageView;
18 import javafx.stage.Stage;
19 import java.sql.*;
20 import javafx.scene.Parent;
21 import javafx.scene.input.MouseEvent;
22 import javafx.stage.Modality;
23 import javafx.stage.StageStyle;
24 import sam.FXMLMainController;
25 import sam.FXMLStaffController;
26 import sam.Sam;
```

1.2 Code

```java
public class LoginController implements Initializable {
    private FXMLNhanVienController nhanvienController;
    private FXMLMainController mainController;
    private FXMLStaffController staffController;

    public void setMainController(FXMLMainController mainController) {
        this.mainController = mainController;
    }
    @FXML
    private TextField username;

    @FXML
    private Button btnlogin;
    @FXML
    private Button btnsignup;

    @FXML
    private Button btnclose;
    @FXML
    private CheckBox ckremember;
    @FXML
    private ImageView EYE;

    @FXML
    private ImageView EYE_SLASH;
    @FXML
    private Button btnEYE;

    @FXML
    private Button btnEYE_SLASH;
    @FXML
    private PasswordField passwordField;

    @FXML
    private TextField showpasswordField;
    @FXML
    private Hyperlink forgotpassword;
    private boolean isPasswordVisible = false;
//=============================================
    // Connect Database
    /**
     *
     */

    private Connection conn;
    private PreparedStatement prepare;
    private ResultSet rs;

//=============================================
    private static final String UPDATE_RMBPASSWORD_SQL = "UPDATE Login SET RmbPassword = ? WHERE Username = ?";
    private static final String SELECT_RMBPASSWORD_SQL = "SELECT RmbPassword FROM Login WHERE Username = ?";

```

```java
public LoginController() {
     this.conn = DB.ConnectDB.getConnectDB();
}

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

    btnclose.setOnAction(event -> btnclose());
    //Show/Hide Password
    btnEYE.setOnAction(event -> togglePasswordVisibility());
    btnEYE_SLASH.setOnAction(event -> togglePasswordVisibility());
    EYE.setOnMouseClicked(event -> togglePasswordVisibility());
    EYE_SLASH.setOnMouseClicked(event -> togglePasswordVisibility());
    updatePasswordVisibility();
    forgotpassword.setOnAction(event -> ForgotPassword());

    btnsignup.setOnAction(event -> CreateAdminAccount());
    btnlogin.setOnAction(event -> {
        try {
            login();
        } catch (IOException | SQLException ex) { // Print the exception for debugging purposes
            // Print the exception for debugging purposes
            AlertHelper.errorMessage("Error during login: " + ex.getMessage());
        }
    });

    passwordField.addEventFilter(MouseEvent.MOUSE_CLICKED, event -> {
        loadPassword();

    });
    passwordField.focusedProperty().addListener((observable, oldValue, newValue) -> {
        if (newValue) { // Khi focus được nhận
            loadPassword();
            resetRemember();
        }
    });

    showpasswordField.textProperty().addListener((ObservableValue<? extends String> observable, String oldValue, String newValue) -> {
        if (passwordField.getText() == null  || !passwordField.getText().equals(newValue)) {
            passwordField.setText(newValue);
        }
    });
}
private String fullName = "";
```

```java
public void login() throws IOException, SQLException {
        try {
            if (username.getText().isEmpty()) {
                AlertHelper.errorMessage("Username cannot be empty");
                return;
            }

            if (passwordField.getText().isEmpty()) {
                AlertHelper.errorMessage("Password cannot be empty");
                return;
            }

            // Verify username and password
            String selectLogin = "SELECT Username FROM Login WHERE Username = ? AND Password = ?";
            conn = DB.ConnectDB.getConnectDB();
            prepare = conn.prepareStatement(selectLogin);
            prepare.setString(1, username.getText());
            prepare.setString(2, passwordField.getText());
            rs = prepare.executeQuery();

            if (rs.next()) {
                // Username and password match, now fetch the role
                String selectRole = "SELECT Role FROM Role WHERE Username = ?";
                prepare = conn.prepareStatement(selectRole);
                prepare.setString(1, username.getText());
                rs = prepare.executeQuery();

                if (rs.next()) {
                    String role = rs.getString("Role");
                    String fxmlFile = "";

                    if (!"Manager".equals(role)) {
                        if ("Staff".equals(role)) {
                            fxmlFile = "/sam/FXMLStaff.fxml";
                        } else {
                            AlertHelper.errorMessage("Unknown role: " + role);
                            return;
                        }
                    } else {
                        fxmlFile = "/sam/FXMLMain.fxml";
                    }
                    // lấy thông tin username
                    String selectUserInfo = "SELECT Fullname FROM Information WHERE Phone = ?";
                    prepare = conn.prepareStatement(selectUserInfo);
                    prepare.setString(1, username.getText());
                    rs = prepare.executeQuery();

                    if (rs.next()) {
                        fullName = rs.getString("Fullname");
                    }

                    if (ckremember.isSelected()) {
                        savePassword(username.getText(), passwordField.getText());
                    } else {
                        clearRMPassword(username.getText());
                    }
                    FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));
                    Parent root = loader.load();
                    Scene scene = new Scene(root);
                    String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
                    scene.getStylesheets().add(cssPath);

                    Stage stage = (Stage) username.getScene().getWindow(); // Get the current stage

                    stage.setResizable(true);

                    if("Staff".equals(role)){
                        FXMLStaffController staffController = loader.getController();
                    staffController.setFullName(fullName);

                    }else{
                        FXMLMainController mainController = loader.getController();
                    mainController.setFullName(fullName);
                    }

                    Sam appInstance = new Sam();
                    appInstance.makeWindowDraggable(stage, root);

                    stage.setScene(scene);
                    stage.centerOnScreen();
                    stage.show();
                } else {
                    AlertHelper.errorMessage("Role not found for the username");
                }
            } else {
                AlertHelper.errorMessage("Incorrect username or password");
            }
        } catch (SQLException e) {
            AlertHelper.errorMessage("Database error: " + e.getMessage());
        } finally {
            closeResources(rs, prepare, conn);
        }
    }
```

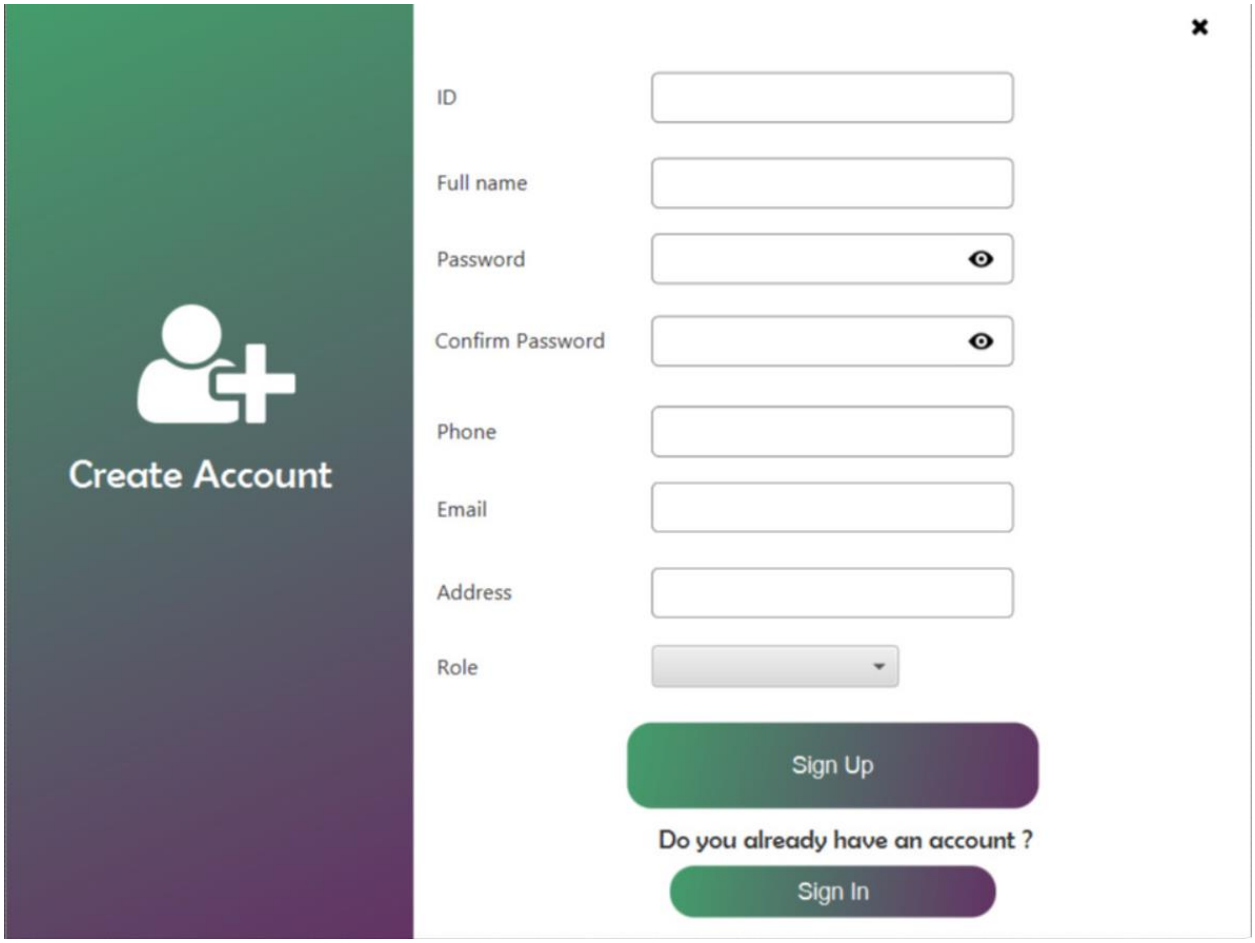
```java
private void closeResources(ResultSet rs, PreparedStatement prepare, Connection conn) {
        try {
            if (rs != null) {
                rs.close();
            }
            if (prepare != null) {
                prepare.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
        }
    }

    // RMP Function
    //
    private void resetRemember() {
        if (isPasswordRemembered()) {
            ckremember.setSelected(true);
        } else {
            ckremember.setSelected(false);
        }
    }

    private boolean isPasswordRemembered() {
        try (Connection conn = DB.ConnectDB.getConnectDB()) {
            if (conn == null) {
                System.out.println("Connection is null. Cannot check password.");
                return false;
            }

            String usernameText = username.getText();
            if (usernameText == null || usernameText.isEmpty()) {
                System.out.println("Username field is empty.");
                return false;
            }

            try (PreparedStatement selectStmt = conn.prepareStatement(SELECT_RMBPASSWORD_SQL)) {
                selectStmt.setString(1, usernameText);
                try (ResultSet rs = selectStmt.executeQuery()) {
                    if (rs.next()) {
                        String rmbPasswordText = rs.getString("RmbPassword");
                        return rmbPasswordText != null && !rmbPasswordText.isEmpty();
                    }
                }
            }
        } catch (SQLException e) {
            System.err.println("SQL error: " + e.getMessage());
        }
        return false;
    }

    private void savePassword(String username, String password) {
        try (Connection conn = DB.ConnectDB.getConnectDB()) {
            if (conn == null) {
                System.out.println("Connection is null. Cannot save password.");
                return;
            }

            String query = UPDATE_RMBPASSWORD_SQL;
            try (PreparedStatement stmt = conn.prepareStatement(query)) {
                stmt.setString(1, password);
                stmt.setString(2, username);
                int rowsUpdated = stmt.executeUpdate();
                if (rowsUpdated > 0) {
                    System.out.println("Password remembered successfully.");
                } else {
                    System.out.println("No user found with the specified username.");
                }
            }
        } catch (SQLException e) {
            System.out.println("SQL Exception: " + e.getMessage());
        }
    }
```

```java
private void clearRMPassword(String username) {
        try (Connection conn = DB.ConnectDB.getConnectDB()) {
            if (conn == null) {
                System.out.println("Connection is null. Cannot clear password.");
                return;
            }

            String query = "UPDATE Login SET RmbPassword = NULL WHERE Username = ?";
            try (PreparedStatement stmt = conn.prepareStatement(query)) {
                stmt.setString(1, username);
                int rowsUpdated = stmt.executeUpdate();
                if (rowsUpdated > 0) {
                    System.out.println("Password cleared successfully.");
                } else {
                    System.out.println("No user found with the specified username.");
                }
            }
        } catch (SQLException e) {
            System.out.println("SQL Exception: " + e.getMessage());
        }
    }

    private void loadPassword() {
        try (Connection conn = DB.ConnectDB.getConnectDB()) {
            if (conn == null) {
                System.out.println("Connection is null. Cannot load password.");
                return;
            }

            String usernameText = username.getText();
            if (usernameText == null || usernameText.isEmpty()) {
                System.out.println("Username field is empty.");
                return;
            }

            try (PreparedStatement selectStmt = conn.prepareStatement(SELECT_RMBPASSWORD_SQL)) {
                selectStmt.setString(1, usernameText);
                try (ResultSet rs = selectStmt.executeQuery()) {
                    if (rs.next()) {
                        String rmbPasswordText = rs.getString("rmbpassword");
                        passwordField.setText(rmbPasswordText);
                        showpasswordField.setText(rmbPasswordText);
                        ckremember.setSelected(true);
                    }
                }
            }
        } catch (SQLException e) {
            System.err.println("SQL error: " + e.getMessage());
        }
    }

    // End RMP Fucntion
    // Show Password
    @FXML
    public void togglePasswordVisibility() {
        isPasswordVisible = !isPasswordVisible;
        updatePasswordVisibility();
    }

    public void updatePasswordVisibility() {
        if (isPasswordVisible) {
            // Show password
            passwordField.setVisible(false);
            showpasswordField.setVisible(true);
            showpasswordField.setText(passwordField.getText());
            btnEYE.setVisible(false);
            btnEYE_SLASH.setVisible(true);
        } else {
            // Hide password
            passwordField.setVisible(true);
            showpasswordField.setVisible(false);
            btnEYE.setVisible(true);
            btnEYE_SLASH.setVisible(false);
        }
    }

```

```java
//End Show Password
    @FXML
    public void ForgotPassword() {
        try {
            // Tạo một FXMLLoader để tải tệp FXML
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/ForgotPassword/ForgotPassword.fxml"));

            // Tải tệp FXML và tạo một đối tượng Parent
            Parent root = loader.load();

            // Tạo một Scene mới từ đối tượng Parent
            Scene scene = new Scene(root);


            // Tạo một Stage mới
            Stage Stageforgot = new Stage();
            Stageforgot.setTitle("Forgot Password"); // Tiêu đề của cửa sổ mới
            Stageforgot.setScene(scene);
            Stageforgot.initStyle(StageStyle.UNDECORATED);

            // Đặt kiểu Modality cho Stage mới nếu bạn muốn khóa giao diện cũ cho đến khi giao diện mới được đóng
            Stageforgot.initModality(Modality.APPLICATION_MODAL);

            // Hiển thị Stage mới
            Stageforgot.show();
        } catch (IOException e) {

        }
    }

    @FXML
    public void CreateAdminAccount() {
        try {
            FXMLLoader loader;
            loader = new FXMLLoader(getClass().getResource("/Register/Register.fxml"));
            Parent root = loader.load();
            Scene scene = new Scene(root);
            Stage stage = (Stage) username.getScene().getWindow();
            stage.setScene(scene);
            stage.show();
        } catch (IOException e) {

        }
    }

    public void btnclose() {
        System.exit(0);
    }

}
```

```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package Login;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *
 * @author huypg
 */
public class Validate {
     // Phương thức để kiểm tra xem tài khoản có tồn tại không
    public boolean checkAccountExists(String username) {
        String sql = "SELECT COUNT(*) FROM users WHERE username = ?";
        try (
            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement stmt = conn.prepareStatement(sql)
        ) {
            stmt.setString(1, username);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                int count = rs.getInt(1);
                return count > 0;
            }
        } catch (SQLException e) {
        }
        return false;
    }

    // Phương thức để kiểm tra tính chính xác của mật khẩu
    public boolean checkPassword(String username, String password) {
        String sql = "SELECT password FROM users WHERE username = ?";
        try (
            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement stmt = conn.prepareStatement(sql)
        ) {
            stmt.setString(1, username);
            ResultSet rs = stmt.executeQuery();
            if (rs.next()) {
                String storedPassword = rs.getString("password");
                return storedPassword.equals(password);
            }
        } catch (SQLException e) {
         }
        return false;
    }
}
```

2 Register


Create Account
ID
Full name
Password
Confirm Password
Phone
Email
Address
Role
Sign Up
Do you already have an account ?
Sign In

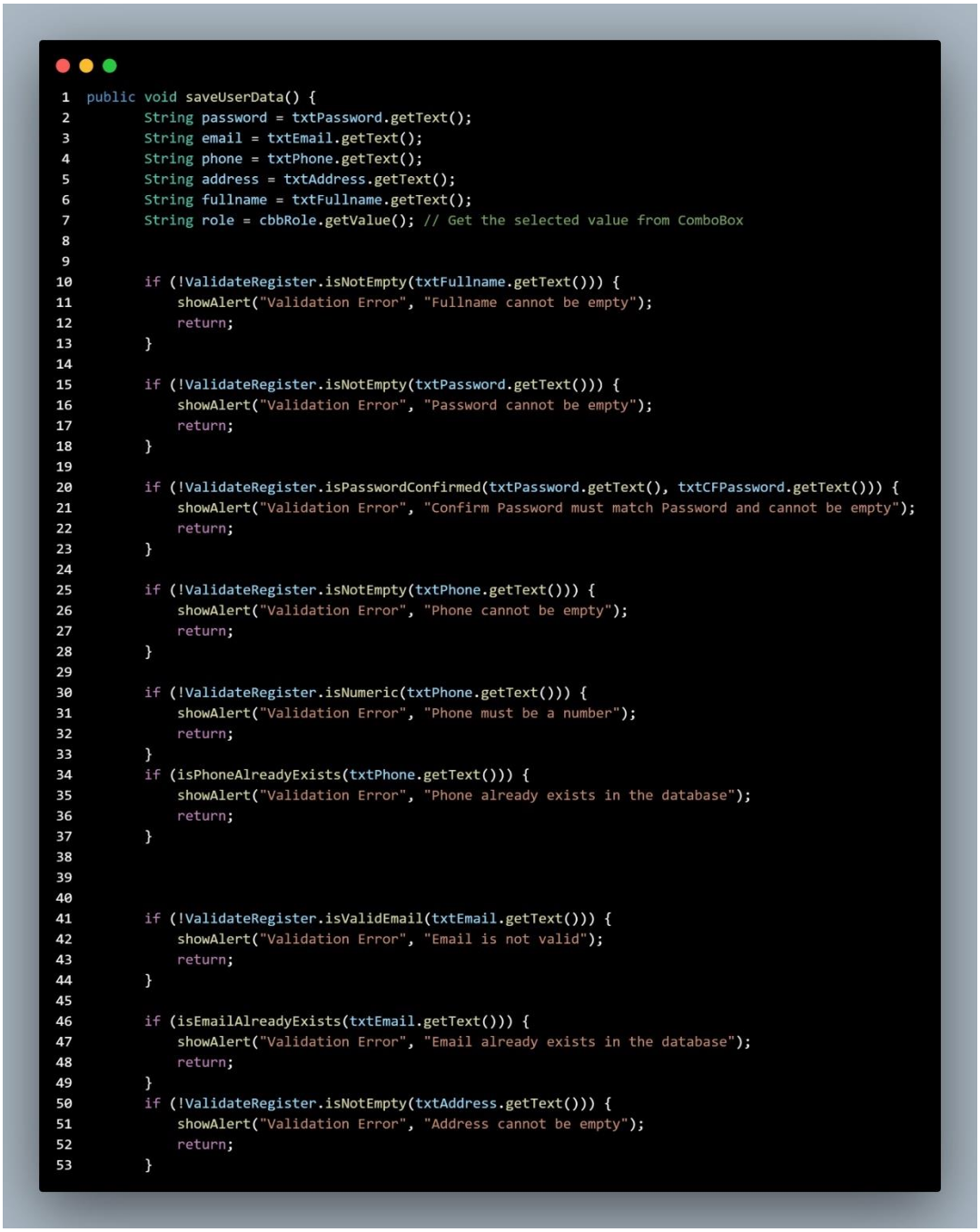
2.1 Import

```java
@FXML
    private ImageView EYE;

    @FXML
    private ImageView EYE_SLASH;

    @FXML
    private Button btnEYE;

    @FXML
    private Button btnEYE_SLASH;

    @FXML
    private ImageView EYECF;

    @FXML
    private ImageView EYE_SLASHCF;

    @FXML
    private Button btnEYECF;

    @FXML
    private Button btnEYE_SLASHCF;

    @FXML
    private Button btnSignin;

    @FXML
    private Button btnSignup;

    @FXML
    private ComboBox<String> cbbRole;

    @FXML
    private Button close;
```

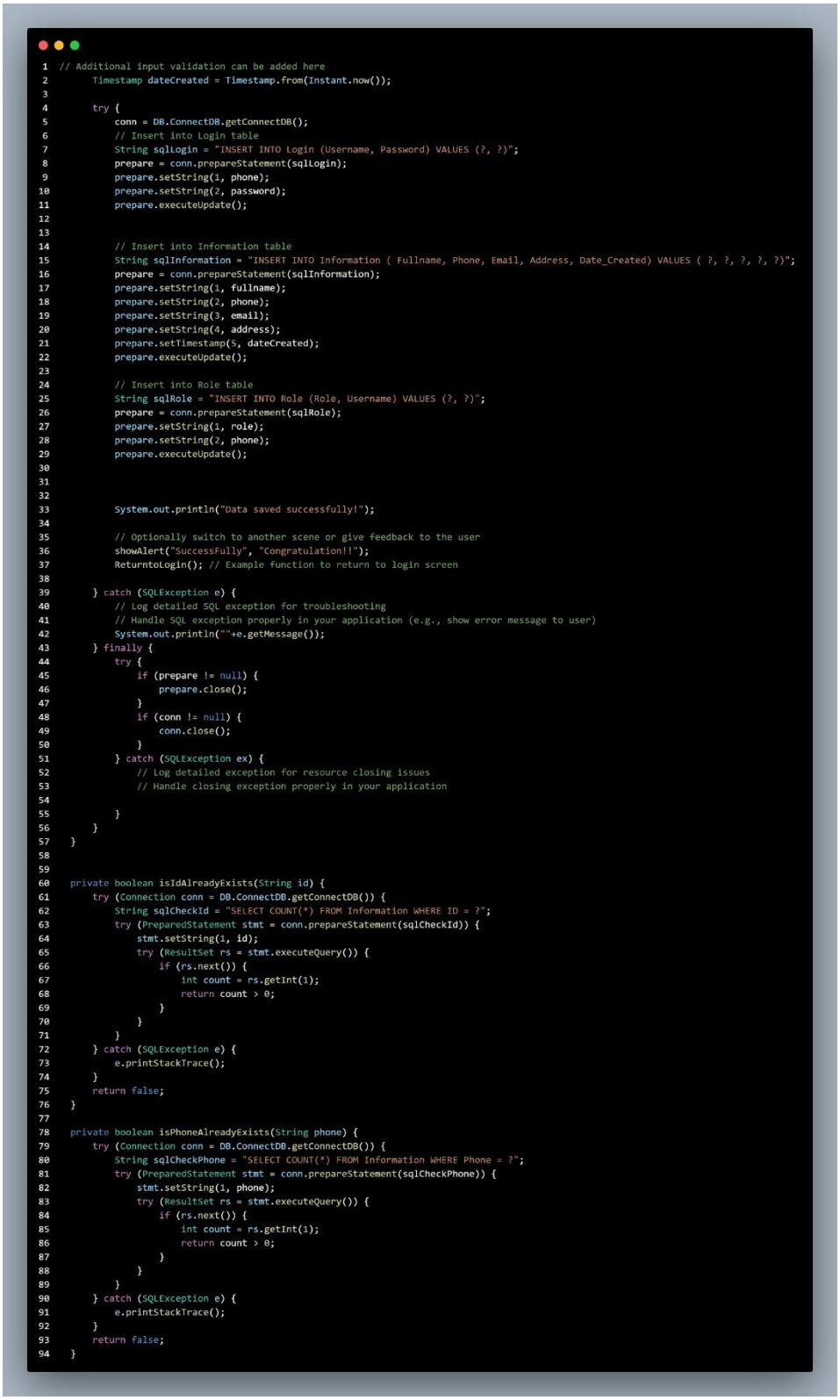
```java
@FXML
    private TextField txtFullname;

    @FXML
    private TextField txtAddress;

    @FXML
    private TextField txtEmail;

    @FXML
    private PasswordField txtPassword;

    @FXML
    private PasswordField txtCFPassword;

    @FXML
    private TextField txtShowPassword;

    @FXML
    private TextField txtshowCFPassword;

    @FXML
    private TextField txtPhone;

    @FXML
    private TextField txtID;

    @FXML

    private boolean isPasswordVisible = false;
    private boolean isCFPasswordVisible = false;

    private Connection conn;
    private PreparedStatement prepare;
```

2.2 Code

```java
@Override
    public void initialize(URL url, ResourceBundle rb) {
        conn = DB.ConnectDB.getConnectDB();
        cbbRole.getItems().addAll("Staff", "Manager");
        cbbRole.setValue("Staff");
        btnSignup.setOnAction(event -> handleRegisterButton());
        btnSignin.setOnAction(event -> ReturntoLogin());
        close.setOnAction(event -> handleCloseButton());
        //Password
        btnEYE.setOnAction(event -> togglePasswordVisibility());
        btnEYE_SLASH.setOnAction(event -> togglePasswordVisibility());
        EYE.setOnMouseClicked(event -> togglePasswordVisibility());
        EYE_SLASH.setOnMouseClicked(event -> togglePasswordVisibility());
        updatePasswordVisibility();
        //Confirm Password
        btnEYECF.setOnAction(event -> toggleCFPVisibility());
        btnEYE_SLASHCF.setOnAction(event -> toggleCFPVisibility());
        EYECF.setOnMouseClicked(event -> toggleCFPVisibility());
        EYE_SLASHCF.setOnMouseClicked(event -> toggleCFPVisibility());
        updateCFPVisibility();
    }

    private void showAlert(String title, String message) {
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

    public class ValidateRegister {

        public static boolean isNotEmpty(String input) {
            return input != null && !input.trim().isEmpty();
        }

        public static boolean isPasswordConfirmed(String password, String confirmPassword) {
            return isNotEmpty(password) && password.equals(confirmPassword);
        }

        public static boolean isNumeric(String input) {
            return input != null && input.matches("\\d+");
        }

        public static boolean isValidEmail(String email) {
            String emailRegex = "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$";
            Pattern pat = Pattern.compile(emailRegex);
            return email != null && pat.matcher(email).matches();
        }
    }

    @FXML
    private void handleRegisterButton() {
        saveUserData();
    }

    @FXML
    private void handleCloseButton() {
        System.exit(0);
    }

    public void ReturntoLogin() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/Login/Login.fxml"));
            Parent root = loader.load();
            Scene scene = new Scene(root);
            Stage stage = (Stage) close.getScene().getWindow();
            stage.setScene(scene);
            stage.show();
        } catch (IOException e) {

        }
    }
```



```java
public void saveUserData() {
        String password = txtPassword.getText();
        String email = txtEmail.getText();
        String phone = txtPhone.getText();
        String address = txtAddress.getText();
        String fullname = txtFullname.getText();
        String role = cbbRole.getValue(); // Get the selected value from ComboBox


        if (!ValidateRegister.isNotEmpty(txtFullname.getText())) {
            showAlert("Validation Error", "Fullname cannot be empty");
            return;
        }

        if (!ValidateRegister.isNotEmpty(txtPassword.getText())) {
            showAlert("Validation Error", "Password cannot be empty");
            return;
        }

        if (!ValidateRegister.isPasswordConfirmed(txtPassword.getText(), txtCFPassword.getText())) {
            showAlert("Validation Error", "Confirm Password must match Password and cannot be empty");
            return;
        }

        if (!ValidateRegister.isNotEmpty(txtPhone.getText())) {
            showAlert("Validation Error", "Phone cannot be empty");
            return;
        }

        if (!ValidateRegister.isNumeric(txtPhone.getText())) {
            showAlert("Validation Error", "Phone must be a number");
            return;
        }
        if (isPhoneAlreadyExists(txtPhone.getText())) {
            showAlert("Validation Error", "Phone already exists in the database");
            return;
        }



        if (!ValidateRegister.isValidEmail(txtEmail.getText())) {
            showAlert("Validation Error", "Email is not valid");
            return;
        }

        if (isEmailAlreadyExists(txtEmail.getText())) {
            showAlert("Validation Error", "Email already exists in the database");
            return;
        }
        if (!ValidateRegister.isNotEmpty(txtAddress.getText())) {
            showAlert("Validation Error", "Address cannot be empty");
            return;
        }
```



```
// Additional input validation can be added here
        Timestamp dateCreated = Timestamp.from(Instant.now());

        try {
            conn = DB.ConnectDB.getConnectDB();
            // Insert into Login table
            String sqlLogin = "INSERT INTO Login (Username, Password) VALUES (?, ?)";
            prepare = conn.prepareStatement(sqlLogin);
            prepare.setString(1, phone);
            prepare.setString(2, password);
            prepare.executeUpdate();


            // Insert into Information table
            String sqlInformation = "INSERT INTO Information ( Fullname, Phone, Email, Address, Date_Created) VALUES ( ?, ?, ?, ?, ?)";
            prepare = conn.prepareStatement(sqlInformation);
            prepare.setString(1, fullname);
            prepare.setString(2, phone);
            prepare.setString(3, email);
            prepare.setString(4, address);
            prepare.setTimestamp(5, dateCreated);
            prepare.executeUpdate();

            // Insert into Role table
            String sqlRole = "INSERT INTO Role (Role, Username) VALUES (?, ?)";
            prepare = conn.prepareStatement(sqlRole);
            prepare.setString(1, role);
            prepare.setString(2, phone);
            prepare.executeUpdate();



            System.out.println("Data saved successfully!");

            // Optionally switch to another scene or give feedback to the user
            showAlert("SuccessFully", "Congratulation!!");
            ReturntoLogin(); // Example function to return to login screen

        } catch (SQLException e) {
            // Log detailed SQL exception for troubleshooting
            // Handle SQL exception properly in your application (e.g., show error message to user)
            System.out.println(""+e.getMessage());
        } finally {
            try {
                if (prepare != null) {
                    prepare.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException ex) {
                // Log detailed exception for resource closing issues
                // Handle closing exception properly in your application

            }
        }
    }


    private boolean isIdAlreadyExists(String id) {
        try (Connection conn = DB.ConnectDB.getConnectDB()) {
            String sqlCheckId = "SELECT COUNT(*) FROM Information WHERE ID = ?";
            try (PreparedStatement stmt = conn.prepareStatement(sqlCheckId)) {
                stmt.setString(1, id);
                try (ResultSet rs = stmt.executeQuery()) {
                    if (rs.next()) {
                        int count = rs.getInt(1);
                        return count > 0;
                    }
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }

    private boolean isPhoneAlreadyExists(String phone) {
        try (Connection conn = DB.ConnectDB.getConnectDB()) {
            String sqlCheckPhone = "SELECT COUNT(*) FROM Information WHERE Phone = ?";
            try (PreparedStatement stmt = conn.prepareStatement(sqlCheckPhone)) {
                stmt.setString(1, phone);
                try (ResultSet rs = stmt.executeQuery()) {
                    if (rs.next()) {
                        int count = rs.getInt(1);
                        return count > 0;
                    }
                }
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return false;
    }
```

```
private boolean isEmailAlreadyExists(String email) {
        try (Connection conn = DB.ConnectDB.getConnectDB()) {
            String sqlCheckEmail = "SELECT COUNT(*) FROM Information WHERE BINARY  Email = ?";
            try (PreparedStatement stmt = conn.prepareStatement(sqlCheckEmail)) {
                stmt.setString(1, email);
                try (ResultSet rs = stmt.executeQuery()) {
                    if (rs.next()) {
                        int count = rs.getInt(1);
                        return count > 0;
                    }
                }
            }
        } catch (SQLException e) {
        }
        return false;
    }

    // Show Password
    @FXML
    public void togglePasswordVisibility() {
        isPasswordVisible = !isPasswordVisible;
        updatePasswordVisibility();
    }

    public void updatePasswordVisibility() {
        if (isPasswordVisible) {
            // Show password
            txtPassword.setVisible(false);
            txtShowPassword.setVisible(true);
            txtShowPassword.setText(txtPassword.getText());
            btnEYE.setVisible(false);
            btnEYE_SLASH.setVisible(true);
        } else {
            // Hide password
            txtPassword.setVisible(true);
            txtShowPassword.setVisible(false);
            btnEYE.setVisible(true);
            btnEYE_SLASH.setVisible(false);
        }
    }

    //End Show Password
    // Show Confirm Password
    @FXML
    public void toggleCFPVisibility() {
        isCFPasswordVisible = !isCFPasswordVisible;
        updateCFPVisibility();
    }

    public void updateCFPVisibility() {
        if (isCFPasswordVisible) {
            // Show password
            txtCFPassword.setVisible(false);
            txtshowCFPassword.setVisible(true);
            txtshowCFPassword.setText(txtCFPassword.getText());
            btnEYECF.setVisible(false);
            btnEYE_SLASHCF.setVisible(true);
        } else {
            // Hide password
            txtCFPassword.setVisible(true);
            txtshowCFPassword.setVisible(false);
            btnEYECF.setVisible(true);
            btnEYE_SLASHCF.setVisible(false);
        }
    }

    //End Show Confirm Password
}
```

3 Forgot Password

```java
public class ForgotPasswordController implements Initializable {

    @FXML
    private StackPane Fogot;

    @FXML
    private Button exit;
    @FXML
    private Button btnSendCode;

    @FXML
    private Button btnVerify;

    @FXML
    private AnchorPane forgotpane;

    @FXML
    private TextField hide;

    @FXML
    private TextField txtCode;

    @FXML
    private TextField txtemail;

    @FXML
    private AnchorPane verifypane;

    @FXML
    private AnchorPane resetpane;

    @FXML
    private ImageView EYE;

    @FXML
    private ImageView EYECF;

    @FXML
    private ImageView EYE_SLASH;

    @FXML
    private ImageView EYE_SLASHCF;

    @FXML
    private PasswordField NewPassword;

    @FXML
    private PasswordField ConfirmNewPassword;

    @FXML
    private TextField ShowConfirmNewPassword;

    @FXML
    private TextField Shownewpassword;

    @FXML
    private Button btnEYE;

    @FXML
    private Button btnEYECF;

    @FXML
    private Button btnEYE_SLASH;

    @FXML
    private Button btnEYE_SLASHCF;

    @FXML
    private Button btnResetPassword;

    private boolean isPasswordVisible = false;
    private boolean isCFPasswordVisible = false;

```

3.1 Code

```java
@Override
    public void initialize(URL url, ResourceBundle rb) {
        btnResetPassword.setOnAction(event -> resetPassword());
        btnEYE.setOnAction(event -> togglePasswordVisibility());
        btnEYE_SLASH.setOnAction(event -> togglePasswordVisibility());
        EYE.setOnMouseClicked(event -> togglePasswordVisibility());
        EYE_SLASH.setOnMouseClicked(event -> togglePasswordVisibility());
        updatePasswordVisibility();
        btnEYECF.setOnAction(event -> toggleCFPasswordVisibility());
        btnEYE_SLASHCF.setOnAction(event -> toggleCFPasswordVisibility());
        EYECF.setOnMouseClicked(event -> toggleCFPasswordVisibility());
        EYE_SLASHCF.setOnMouseClicked(event -> toggleCFPasswordVisibility());
        updateCFPasswordVisibility();

        exit.setOnMouseClicked(event -> {
            Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
            stage.close();
        });
    }

    @FXML
    private void Sendcode(ActionEvent event) throws Exception {
        if (event.getSource() == btnSendCode) {

            forgotpane.setVisible(false);
            verifypane.setVisible(true);
            resetpane.setVisible(false);
            sendOTP();

        }
    }

    @FXML
    private void VerifyCode(ActionEvent event) throws Exception {
        if (event.getSource() == btnVerify) {

            String otp = hide.getText();
            String enteredOtp = txtCode.getText().trim();

            if (otp.equals(enteredOtp)) {
                // OTP is correct, load the Staff Dashboard
                forgotpane.setVisible(false);
                verifypane.setVisible(false);
                resetpane.setVisible(true);
            } else {
                JOptionPane.showMessageDialog(null, "Invalid OTP. Please try again.");
            }
        }
    }

    private void showAlert(String title, String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

```

```java
@FXML
    private void resetPassword() {
        String newPassword = NewPassword.getText().trim();
        String confirmNewPassword = ConfirmNewPassword.getText().trim();

        if (newPassword.isEmpty() || confirmNewPassword.isEmpty()) {
            // Hiển thị thông báo lỗi: Cả hai trường phải được điền
            showAlert("Error", "Both fields must be filled.");
            return;
        }

        if (!newPassword.equals(confirmNewPassword)) {
            // Hiển thị thông báo lỗi: Mật khẩu không khớp
            showAlert("Error", "Passwords do not match.");
            return;
        }

        try (Connection connection = DB.ConnectDB.getConnectDB()) {
            // Bước 1: Lấy Phone từ bảng Information dựa trên email
            String phoneQuery = "SELECT Phone FROM Information WHERE Email = ?";
            try (PreparedStatement phoneStatement = connection.prepareStatement(phoneQuery)) {
                phoneStatement.setString(1, txtemail.getText());
                ResultSet resultSet = phoneStatement.executeQuery();
                if (resultSet.next()) {
                    String phone = resultSet.getString("Phone");

                    // Đặt username bằng giá trị phone
                    String username = phone;

                    // Bước 3: Cập nhật mật khẩu mới trong bảng Login
                    String updatePasswordQuery = "UPDATE Login SET Password = ? WHERE Username = ?";
                    try (PreparedStatement updatePasswordStatement = connection.prepareStatement(updatePasswordQuery)) {
                        updatePasswordStatement.setString(1, newPassword);
                        updatePasswordStatement.setString(2, username);
                        int rowsUpdated = updatePasswordStatement.executeUpdate();
                        if (rowsUpdated > 0) {
                            System.out.println("Password updated successfully for user: " + username);
                            try {
                                FXMLLoader loader = new FXMLLoader(getClass().getResource("/Login/Login.fxml"));
                                Parent root = loader.load();

                                Stage stage = new Stage();
                                stage.initStyle(StageStyle.UNDECORATED);
                                stage.setScene(new Scene(root));
                                stage.show();
                                clearRMPassword(username);

                            } catch (IOException e) {
                                JOptionPane.showMessageDialog(null, "Error loading verification screen: " + e.getMessage());
                            }
                        } else {
                            System.out.println("Failed to update password for user: " + username);
                        }
                    }
                } else {
                    System.out.println("No user found with phone: ");
                    // Xử lý khi không tìm thấy Username
                }
            }
        } catch (SQLException e) {
            // Xử lý ngoại lệ SQL
            e.printStackTrace();
        }
    }
```

```java
@FXML
    private void clearRMPassword(String username) {
        try (Connection conn = DB.ConnectDB.getConnectDB()) {
            if (conn == null) {
                System.out.println("Connection is null. Cannot clear password.");
                return;
            }

            String query = "UPDATE Login SET RmbPassword = NULL WHERE Username = ?";
            try (PreparedStatement stmt = conn.prepareStatement(query)) {
                stmt.setString(1, username);
                int rowsUpdated = stmt.executeUpdate();
                if (rowsUpdated > 0) {
                    System.out.println("Password cleared successfully.");
                } else {
                    System.out.println("No user found with the specified username.");
                }
            }
        } catch (SQLException e) {
            System.out.println("SQL Exception: " + e.getMessage());
        }
    }

    // Toggle visibility of NewPassword field
    @FXML
    private void togglePasswordVisibility() {
        isPasswordVisible = !isPasswordVisible;
        updatePasswordVisibility();
    }

    private void updatePasswordVisibility() {
        if (isPasswordVisible) {
            NewPassword.setVisible(false);
            Shownewpassword.setVisible(true);
            Shownewpassword.setText(NewPassword.getText());
            btnEYE.setVisible(false);
            btnEYE_SLASH.setVisible(true);
        } else {
            NewPassword.setVisible(true);
            Shownewpassword.setVisible(false);
            btnEYE.setVisible(true);
            btnEYE_SLASH.setVisible(false);
        }
    }

    // Toggle visibility of ConfirmNewPassword field
    @FXML
    private void toggleCFPasswordVisibility() {
        isCFPasswordVisible = !isCFPasswordVisible;
        updateCFPasswordVisibility();
    }

    private void updateCFPasswordVisibility() {
        if (isCFPasswordVisible) {
            ConfirmNewPassword.setVisible(false);
            ShowConfirmNewPassword.setVisible(true);
            ShowConfirmNewPassword.setText(ConfirmNewPassword.getText());
            btnEYECF.setVisible(false);
            btnEYE_SLASHCF.setVisible(true);
        } else {
            ConfirmNewPassword.setVisible(true);
            ShowConfirmNewPassword.setVisible(false);
            btnEYECF.setVisible(true);
            btnEYE_SLASHCF.setVisible(false);
        }
    }
```

```
1
2       private String generateOTP(int len) {
3           // All possible characters of my OTP
4           String str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" + "0123456789";
5           int n = str.length();
6
7           // String to hold my OTP
8           StringBuilder otp = new StringBuilder();
9
10          for (int i = 1; i <= len; i++) {
11              otp.append(str.charAt((int) (Math.random() * str.length())));
12          }
13          return otp.toString();
14      }
15
16      public void sendOTP() {
17          int len = 6;
18          String otp = generateOTP(len);
19          hide.setText(otp);
20          String host = "smtp.gmail.com";
21          final String username = "mingken036@gmail.com";
22          final String password = "phch ducw memx hopt";
23          Properties props = new Properties();
24          props.put("mail.smtp.host", host);
25          props.put("mail.smtp.port", 465);
26          props.put("mail.smtp.auth", "true");
27          props.put("mail.smtp.starttls.enable", "true");
28          props.put("mail.smtp.debug", true);
29          props.put("mail.smtp.socketFactory.port", 465);
30          props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
31          props.put("mail.smtp.socketFactory.fallback", false);
32
33          try {
34              Session session = Session.getDefaultInstance(props, null);
35              session.setDebug(true);
36
37              MimeMessage message = new MimeMessage(session);
38              message.setText("Your OTP is " + hide.getText());
39              message.setSubject("OTP For your Reset Account");
40              message.setFrom(new InternetAddress(username));
41              message.addRecipient(RecipientType.TO, new InternetAddress(txtemail.getText().trim()));
42              message.saveChanges();
43              try {
44                  Transport transport = session.getTransport("smtp");
45                  transport.connect(host, username, password);
46                  transport.sendMessage(message, message.getAllRecipients());
47                  transport.close();
48                  JOptionPane.showMessageDialog(null, "OTP has send to your Email id");
49              } catch (Exception e) {
50                  JOptionPane.showMessageDialog(null, "Please check your internet connection");
51              }
52
53          } catch (Exception e) {
54
55              e.printStackTrace();
56              JOptionPane.showMessageDialog(null, e);
57          }
58
59      }
60
61  }
62
```

4.Dashboard Admin

```java
public class FXMLMainController implements Initializable {

    private LoginController loginController;
    private BillsController billsController;

    public void setBillsController(BillsController billsController) {
        this.billsController = billsController;
    }

    public FXMLMainController() {

    }
    private CustomerController customerController;
    private OrderTransactionController orderTransactionController;

    private static FXMLMainController instance;
    private boolean drawerImageOpen = false;
    @FXML
    private ImageView Bills;

    @FXML
    private ImageView Customer;

    @FXML
    private ImageView Home;

    @FXML
    private ImageView Logout;

    @FXML
    private ImageView OrderReceipt;

    @FXML
    private ImageView Product;

    @FXML
    private ImageView ProductDetails;
    @FXML
    private ImageView Setting;
    @FXML
    private ImageView Selling;

    @FXML
    private ImageView Staff;

    @FXML
    private ImageView Supplier;

    @FXML
    private Button BtnSetting;
    @FXML
    private Label Time;
    private Timeline timeline;
    @FXML
    private Label expiryCountLabel;
    @FXML
    private Button btnLogOut;
    @FXML
    private Button Export;
    @FXML
    private Button User;
    @FXML
    private TextField textField;
    @FXML
    public AnchorPane opacityPane, drawerPane, main;
    @FXML
    private Button drawerImage;
    @FXML
    private Label UsernameTF;
    @FXML
    private ImageView exit;
    @FXML
    private TextArea expiringProductsDetails;
    @FXML
    private Label orderCountLabelInvoide;
    @FXML
    private Label orderCountLabel;
```

4.1 Code

```java
public static FXMLMainController getInstance() {
        return instance;
    }

    public void setFullName(String fullName) {
        UsernameTF.setText(fullName);
        UsernameTF.setFont(Font.font("System", 20.0));
    }

    public void loadFXML(String fxml) {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource(fxml));
            Parent root = loader.load();

            if (fxml.equals("/demoproject1/FXMLChamluong.fxml")) {
                FXMLChamluongController controller = loader.getController();
                controller.setMainController(this); // Thiết lập mainController cho FXMLChamluongController
            } else if (fxml.equals("/demoproject1/FXMLNhanVien.fxml")) {
                FXMLNhanVienController controller = loader.getController();
                controller.setMainController(this); // Thiết lập mainController cho FXMLNhanvienController
            } else if (fxml.equals("/demoproject1/FXMLBangluong.fxml")) {
                FXMLBangluongController controller = loader.getController();
                controller.setMainController(this); // Thiết lập mainController cho FXMLBangluongController
            }

            main.getChildren().clear();
            main.getChildren().add(root);

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

    }

    @FXML
    private void handleSupplier() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/eproject/Supplier.fxml"));
            Parent productPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(productPane);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }

```

```java
@FXML
    private void handleOrderReceipt() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/eproject/ImportOrderReceipt.fxml"));
            Parent productPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(productPane);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }

    @FXML
    private void handleProduct() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/Druglist/Druglist.fxml"));
            Parent productPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(productPane);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }

    @FXML
    private void handleSetting() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/AccountInfo/AccountInfo.fxml"));
            Parent productDetails = loader.load();

            main.getChildren().clear();
            main.getChildren().add(productDetails);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }
```

```java
@FXML
    private void handleProductDetails() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/ImportMedicineDetails/ImportMedicineDetails.fxml"));
            Parent productDetails = loader.load();

            main.getChildren().clear();
            main.getChildren().add(productDetails);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }

    @FXML
    private void handleSellingButtonClick() {
        try {
            // Load the OrderTransaction.fxml file
            FXMLLoader loader = new FXMLLoader(getClass().getResource("OrderTransaction.fxml"));
            Parent orderTransactionPane = loader.load();

            // Access the controller instance from the loader
            OrderTransactionController orderTransactionController = loader.getController();

            // Set the CustomerController instance in OrderTransactionController
            orderTransactionController.setCustomerController(new CustomerController());

            // Clear the main pane and add OrderTransaction.fxml to it
            main.getChildren().clear();
            main.getChildren().add(orderTransactionPane);

            // Show the opacity pane
            opacityPane.setVisible(true);

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

        // Animation to fade out opacity pane
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        // Animation to slide drawer pane out of view
        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }
```

```java
@FXML
    private void handleCustomerClick() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("Customer.fxml"));
            Parent customerPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(customerPane);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }

    @FXML
    private void handleBillsClick() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("Bills.fxml"));
            Parent customerPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(customerPane);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }

    @FXML
    private void handleDashboardButtonClick() {
        try {

            FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLMain.fxml"));
            Parent mainPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(mainPane);

            opacityPane.setVisible(true);

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

        }

        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600); // Move drawerPane to the left
        translateTransition.play();

        drawerImageOpen = false;
    }
```

```java
@FXML
private void handleUserButtonClick() {

    loadFXML("/demoproject1/FXMLNhanVien.fxml");
    FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
    fadeTransition.setFromValue(0.15);
    fadeTransition.setToValue(0);
    fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
    fadeTransition.play();

    TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
    translateTransition.setByX(-600);
    translateTransition.play();

    drawerImageOpen = false;
}

public void updateLabels() {
    // Call updateTotalBillsAndRevenue from BillsController
    billsController.updateTotalBillsAndRevenue();
    // Retrieve updated values
    int totalBills = billsController.getTotalBills();
    int totalRevenue = billsController.getTotalRevenue();

    // Update labels in FXMLStaff
    orderCountLabel.setText(String.format("%d Invoices", totalBills));
    orderCountLabelInvoide.setText(String.format("%dd", totalRevenue));
}

@FXML
private LineChart<String, Number> revenueChart;
private final CategoryAxis xAxis = new CategoryAxis();
private final NumberAxis yAxis = new NumberAxis();

public void updateRevenueChart() {
    revenueChart.getData().clear(); // Xóa dữ liệu hiện tại trên biểu đồ

    // Lấy dữ liệu tổng doanh thu theo từng ngày trong tháng hiện tại
    XYChart.Series<String, Number> series = new XYChart.Series<>();
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
    Calendar calendar = Calendar.getInstance();
    calendar.set(Calendar.DAY_OF_MONTH, 4); // Đặt ngày là ngày đầu tiên của tháng

    try {

        Connection conn = DB.ConnectDB.getConnectDB();
        // Lặp qua từng ngày trong tháng và lấy tổng doanh thu
        while (calendar.get(Calendar.YEAR) == Calendar.getInstance().get(Calendar.YEAR)
                && calendar.get(Calendar.MONTH) <= Calendar.getInstance().get(Calendar.MONTH)) {
            String currentDate = sdf.format(calendar.getTime());
            String sql = "SELECT SUM(TotalPrice) AS TotalRevenue FROM Bills WHERE BillDate = ?";

            PreparedStatement pst = conn.prepareStatement(sql);
            pst.setString(1, currentDate);
            ResultSet rs = pst.executeQuery();

            if (rs.next()) {
                int totalRevenue = rs.getInt("TotalRevenue");
                int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
                // Thêm dữ liệu vào series, chuyển totalRevenue thành Integer
                series.getData().add(new XYChart.Data<>(currentDate, totalRevenue));
            }

            pst.close();
            calendar.add(Calendar.DAY_OF_MONTH, 1); // Chuyển sang ngày tiếp theo

        }

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

    // Thêm series vào biểu đồ
    revenueChart.getData().add(series);
}
```

```java
@FXML
private void checkAllProductsExpiryInfo() {
    List<ProductExpiryTable> expiringProductsList = fetchExpiringProductsFromDatabase();

    // Update UI on the JavaFX Application Thread
    Platform.runLater(() -> {
        expiryCountLabel.setText("There are " + expiringProductsList.size() + " items that will expire within 1 month.");
    });
}

@FXML
private void showProductExpiryInfoForm() {
    List<ProductExpiryTable> expiringProductsList = fetchExpiringProductsFromDatabase();

    try {
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/sam/ProductExpiryInfo.fxml"));
        Parent root = loader.load();
        ProductExpiryInfoController productExpiryInfoController = loader.getController();
        productExpiryInfoController.setExpiringProducts(expiringProductsList);

        Stage stage = new Stage();
        stage.setScene(new Scene(root));
        stage.initStyle(StageStyle.UNDECORATED);
        stage.show();

    } catch (IOException ex) {
        System.out.println("Error loading ProductExpiryInfo.fxml: " + ex.getMessage());
    }
}

private List<ProductExpiryTable> fetchExpiringProductsFromDatabase() {
    List<ProductExpiryTable> expiringProductsList = new ArrayList<>();
    String sql = "SELECT p.MedicineID, p.MedicineName, pd.ManufacturingDate, pd.ExpireDate "
            + "FROM tblProductDetails pd "
            + "JOIN tblProduct p ON pd.MedicineID = p.MedicineID";

    try (Connection conn =  DB.ConnectDB.getConnectDB(); PreparedStatement pst = conn.prepareStatement(sql); ResultSet rs = pst.executeQuery()) {

        while (rs.next()) {
            String medicineID = rs.getString("MedicineID");
            String medicineName = rs.getString("MedicineName");
            LocalDate manufacturingDate = rs.getDate("ManufacturingDate").toLocalDate();
            LocalDate expireDate = rs.getDate("ExpireDate").toLocalDate();

            long daysLeft = ChronoUnit.DAYS.between(LocalDate.now(), expireDate);

            if (daysLeft >= 0 && daysLeft <= 30) {
                // Tính số ngày còn lại đến ngày hết hạn
                long daysUntilExpired = ChronoUnit.DAYS.between(LocalDate.now(), expireDate);

                // Tạo đối tượng Product và thêm vào danh sách
                ProductExpiryTable product = new ProductExpiryTable(medicineID, medicineName, manufacturingDate, expireDate, daysUntilExpired);
                expiringProductsList.add(product);
            }
        }

    } catch (Exception e) {
        System.out.println("Error fetching product details: " + e.getMessage());
    }

    return expiringProductsList;
}

@FXML
private void handleLogout() {
    try {
        // Load login.fxml
        Parent root = FXMLLoader.load(getClass().getResource("/Login/Login.fxml"));
        Scene scene = new Scene(root);
        String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
        scene.getStylesheets().add(cssPath);

        Stage stage = (Stage) btnLogOut.getScene().getWindow(); // Get the current stage
        stage.setScene(scene);
        stage.centerOnScreen();
        stage.show();
    } catch (IOException e) {
        e.printStackTrace(); // Handle exception properly in your application
    }
}
```

```java
@Override
    public void initialize(URL location, ResourceBundle resources) {
        //load
        //timeline
        timeline = new Timeline(
                new KeyFrame(Duration.millis(500), event -> {
                    updateTime(); // Cập nhật thời gian mỗi giây
                })
        );
        timeline.setCycleCount(Animation.INDEFINITE); // Lặp vô hạn
        timeline.play();
        //labelBills
        billsController = new BillsController();
        updateLabels();
        //linechar
        revenueChart.getData().clear();
        updateRevenueChart();
        //productExpiry
        checkAllProductsExpiryInfo();
        exit.setOnMouseClicked(event -> {
            System.exit(0);
        });

        opacityPane.setVisible(false);

        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(1);
        fadeTransition.setToValue(0);
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImage.setOnMouseClicked(event -> {
            if (!drawerImageOpen) {
                opacityPane.setVisible(true);
                FadeTransition fadeTransition1 = new FadeTransition(Duration.seconds(0.5), opacityPane);
                fadeTransition1.setFromValue(0);
                fadeTransition1.setToValue(0.15);
                fadeTransition1.play();

                TranslateTransition translateTransition1 = new TranslateTransition(Duration.seconds(0.1), drawerPane);
                translateTransition1.setByX(600);
                translateTransition1.play();

                drawerImageOpen = true;
            } else {
                FadeTransition fadeTransition1 = new FadeTransition(Duration.seconds(0.5), opacityPane);
                fadeTransition1.setFromValue(0.15);
                fadeTransition1.setToValue(0);
                fadeTransition1.setOnFinished(event1 -> opacityPane.setVisible(false));
                fadeTransition1.play();

                TranslateTransition translateTransition1 = new TranslateTransition(Duration.seconds(0.1), drawerPane);
                translateTransition1.setByX(-600); // Move drawerPane to the left
                translateTransition1.play();

                drawerImageOpen = false;
            }
        });
        Home.setOnMouseClicked(event -> {
            loadFXML("FXMLMain.fxml");

        });
        Staff.setOnMouseClicked(event -> {
            loadFXML("/demoproject1/FXMLNhanVien.fxml");
        });
        Customer.setOnMouseClicked(event -> {
            loadFXML("Customer.fxml");
        });
        Product.setOnMouseClicked(event -> {
            loadFXML("/Druglist/Druglist.fxml");
        });
        Bills.setOnMouseClicked(event -> {
            loadFXML("Bills.fxml");
        });
        ProductDetails.setOnMouseClicked(event -> {
            loadFXML("/ImportMedicineDetalls/ImportMedicineDetails.fxml");
        });
        OrderReceipt.setOnMouseClicked(event -> {
            loadFXML("/eproject/ImportOrderReceipt.fxml");
        });
        Setting.setOnMouseClicked(event -> {
            loadFXML("/AccountInfo/AccountInfo.fxml");
        });
        Supplier.setOnMouseClicked(event -> {
            loadFXML("/eproject/Supplier.fxml");
        });
        Selling.setOnMouseClicked(event -> {
            loadFXML("OrderTransaction.fxml");
        });
        Logout.setOnMouseClicked(event -> {
            try {
                // Load login.fxml
                Parent root = FXMLLoader.load(getClass().getResource("/Login/Login.fxml"));
                Scene scene = new Scene(root);
//                String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
//                scene.getStylesheets().add(cssPath);

                Stage stage = (Stage) Logout.getScene().getWindow(); // Get the current stage
                stage.setScene(scene);
                stage.centerOnScreen();
                stage.show();
            } catch (IOException e) {
                e.printStackTrace(); // Handle exception properly in your application
            }
        });
    }

    private void updateTime() {
        LocalDateTime now = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm d MMMM yyyy");
        String formattedDateTime = now.format(formatter);
        Time.setText(formattedDateTime); // Đặt giá trị vào Label có fx:id là Time
    }
}
```

5 Dashboard Staff

```java
public class FXMLStaffController implements Initializable {

    public void setBillsController(BillsController billsController) {
        this.billsController = billsController;
    }
    private CustomerController customerController;
    private OrderTransactionController orderTransactionController;
    private boolean drawerImageOpen = false;
    @FXML
    private ImageView Bill;

    @FXML
    private ImageView Customer;

    @FXML
    private ImageView Home;

    @FXML
    private ImageView Logout;

    @FXML
    private ImageView Order;
    @FXML
    private Label Time;
    private Timeline timeline;
    @FXML
    private Label expiryCountLabel;
    @FXML
    private Button btnLogOut;
    @FXML
    private Button admin;
    @FXML
    private Label totalBillsLabel;
    @FXML
    private Label UserNameLB;
    @FXML
    private Label totalRevenueLabel;
    @FXML
    private Button drawerImage;
    @FXML
    private Button Selling;
    @FXML
    private AnchorPane drawerPane;
    @FXML
    private AnchorPane main;
    private LoginController loginController;
    private BillsController billsController;
    @FXML
    private AnchorPane opacityPane;

    @FXML
    private ImageView exit;

    public void setFullName(String fullName) {
        UserNameLB.setText(fullName);
        UserNameLB.setFont(Font.font("System", 20.0));
    }
```

5.1 Code

```
public void handleDashboardButtonClick() {
        try {

            FXMLLoader loader = new FXMLLoader(getClass().getResource("/sam/FXMLStaff.fxml"));
            Parent mainPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(mainPane);

            opacityPane.setVisible(true);

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

        }

        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600); // Move drawerPane to the left
        translateTransition.play();

        drawerImageOpen = false;
    }

    @FXML
    private void handleCustomer() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("Customer.fxml"));
            Parent customerPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(customerPane);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }

    @FXML
    private void handleBills() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("Bills.fxml"));
            Parent customerPane = loader.load();

            main.getChildren().clear();
            main.getChildren().add(customerPane);
            opacityPane.setVisible(true);

        } catch (IOException e) {
            e.printStackTrace();
        }
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }

```

```java
public void updateLabels() {
        // Call updateTotalBillsAndRevenue from BillsController
        billsController.updateTotalBillsAndRevenue();

        // Retrieve updated values
        int totalBills = billsController.getTotalBills();
        int totalRevenue = billsController.getTotalRevenue();

        // Update labels in FXMLStaff
        totalBillsLabel.setText(String.format("%d Invoices", totalBills));
        totalRevenueLabel.setText(String.format("%dđ", totalRevenue));
    }

    @FXML
    private void handleSelling() {
        try {
            // Load the OrderTransaction.fxml file
            FXMLLoader loader = new FXMLLoader(getClass().getResource("OrderTransaction.fxml"));
            Parent orderTransactionPane = loader.load();

            // Access the controller instance from the loader
            OrderTransactionController orderTransactionController = loader.getController();

            // Set the CustomerController instance in OrderTransactionController
            orderTransactionController.setCustomerController(new CustomerController());

            // Clear the main pane and add OrderTransaction.fxml to it
            main.getChildren().clear();
            main.getChildren().add(orderTransactionPane);

            // Show the opacity pane
            opacityPane.setVisible(true);

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

        // Animation to fade out opacity pane
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(0.15);
        fadeTransition.setToValue(0);
        fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
        fadeTransition.play();

        // Animation to slide drawer pane out of view
        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImageOpen = false;
    }
    @FXML
    private LineChart<String, Number> revenueChart;
    private final CategoryAxis xAxis = new CategoryAxis();
    private final NumberAxis yAxis = new NumberAxis();

    public void updateRevenueChart() {
        revenueChart.getData().clear(); // Xóa dữ liệu hiện tại trên biểu đồ

        // Lấy dữ liệu tổng doanh thu theo từng ngày trong tháng hiện tại
        XYChart.Series<String, Number> series = new XYChart.Series<>();
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        Calendar calendar = Calendar.getInstance();
        calendar.set(Calendar.DAY_OF_MONTH, 1); // Đặt ngày là ngày đầu tiên của tháng

        try {

            Connection conn = DB.ConnectDB.getConnectDB();
            // Lặp qua từng ngày trong tháng và lấy tổng doanh thu
            while (calendar.get(Calendar.YEAR) == Calendar.getInstance().get(Calendar.YEAR)
                    && calendar.get(Calendar.MONTH) <= Calendar.getInstance().get(Calendar.MONTH)) {
                String currentDate = sdf.format(calendar.getTime());
                String sql = "SELECT SUM(TotalPrice) AS TotalRevenue FROM Bills WHERE BillDate = ?";

                PreparedStatement pst = conn.prepareStatement(sql);
                pst.setString(1, currentDate);
                ResultSet rs = pst.executeQuery();

                if (rs.next()) {
                    int totalRevenue = rs.getInt("TotalRevenue");
                    int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
                    // Thêm dữ liệu vào series, chuyển totalRevenue thành Integer
                    series.getData().add(new XYChart.Data<>(currentDate, totalRevenue));
                }

                pst.close();
                calendar.add(Calendar.DAY_OF_MONTH, 1); // Chuyển sang ngày tiếp theo

            }

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

        // Thêm series vào biểu đồ
        revenueChart.getData().add(series);
    }

```

```java
@FXML
    private void handleLogout() {
        try {
            // Load login.fxml
            Parent root = FXMLLoader.load(getClass().getResource("/Login/Login.fxml"));
            Scene scene = new Scene(root);
            String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
            scene.getStylesheets().add(cssPath);

            Stage stage = (Stage) btnLogOut.getScene().getWindow(); // Get the current stage
            stage.setScene(scene);
            stage.centerOnScreen();
            stage.show();
        } catch (IOException e) {
            e.printStackTrace(); // Handle exception properly in your application
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        //timeline
        timeline = new Timeline(
                new KeyFrame(Duration.millis(500), event -> {
                    updateTime(); // Cập nhật thời gian mỗi giây
                })
        );
        timeline.setCycleCount(Animation.INDEFINITE); // Lặp vô hạn
        timeline.play();
        //labelBills
        billsController = new BillsController();
        updateLabels();
        //productExpiry
        checkAllProductsExpiryInfo();
        //linechar
        revenueChart.getData().clear();
        updateRevenueChart();

        exit.setOnMouseClicked(event -> {
            System.exit(0);
        });
        opacityPane.setVisible(false);

        // Initial fade out and translation of the drawer
        FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
        fadeTransition.setFromValue(1);
        fadeTransition.setToValue(0);
        fadeTransition.play();

        TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
        translateTransition.setByX(-600);
        translateTransition.play();

        drawerImage.setOnMouseClicked(event -> {
            if (!drawerImageOpen) {
                opacityPane.setVisible(true);
                FadeTransition fadeTransition1 = new FadeTransition(Duration.seconds(0.5), opacityPane);
                fadeTransition1.setFromValue(0);
                fadeTransition1.setToValue(0.15);
                fadeTransition1.play();

                TranslateTransition translateTransition1 = new TranslateTransition(Duration.seconds(0.1), drawerPane);
                translateTransition1.setByX(600); // Move drawerPane to the right
                translateTransition1.play();

                drawerImageOpen = true; // Set state to open
            } else { // Close drawer if it's open
                FadeTransition fadeTransition1 = new FadeTransition(Duration.seconds(0.5), opacityPane);
                fadeTransition1.setFromValue(0.15);
                fadeTransition1.setToValue(0);
                fadeTransition1.setOnFinished(event1 -> opacityPane.setVisible(false));
                fadeTransition1.play();

                TranslateTransition translateTransition1 = new TranslateTransition(Duration.seconds(0.1), drawerPane);
                translateTransition1.setByX(-600); // Move drawerPane to the left
                translateTransition1.play();

                drawerImageOpen = false; // Set state to close
            }
        });
        Home.setOnMouseClicked(event -> {
            loadFXML("/sam/FXMLMain.fxml");
        });
        Customer.setOnMouseClicked(event -> {
            loadFXML("/sam/Customer.fxml");
        });
        Order.setOnMouseClicked(event -> {
            loadFXML("/sam/OrderTransaction.fxml");
        });
        Bill.setOnMouseClicked(event -> {
            loadFXML("/sam/Bills.fxml");
        });
        Logout.setOnMouseClicked(event -> {
            try {
                // Load login.fxml
                Parent root = FXMLLoader.load(getClass().getResource("/Login/Login.fxml"));
                Scene scene = new Scene(root);
//                String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
//                scene.getStylesheets().add(cssPath);

                Stage stage = (Stage) Logout.getScene().getWindow(); // Get the current stage
                stage.setScene(scene);
                stage.centerOnScreen();
                stage.show();
            } catch (IOException e) {
                e.printStackTrace(); // Handle exception properly in your application
            }
        });
    }
```

```java
@FXML
    private void checkAllProductsExpiryInfo() {
        List<ProductExpiryTable> expiringProductsList = fetchExpiringProductsFromDatabase();

        Platform.runLater(() -> {
            expiryCountLabel.setText("There are " + expiringProductsList.size() + " items that will expire within 1 month.");
        });
    }

    @FXML
    private void showProductExpiryInfoForm() {
        List<ProductExpiryTable> expiringProductsList = fetchExpiringProductsFromDatabase();

        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("/sam/ProductExpiryInfo.fxml"));
            Parent root = loader.load();
            ProductExpiryInfoController productExpiryInfoController = loader.getController();
            productExpiryInfoController.setExpiringProducts(expiringProductsList);

            Stage stage = new Stage();
            stage.setScene(new Scene(root));
            stage.initStyle(StageStyle.UNDECORATED);
            stage.show();

        } catch (IOException ex) {
            System.out.println("Error loading ProductExpiryInfo.fxml: " + ex.getMessage());
        }
    }

    private List<ProductExpiryTable> fetchExpiringProductsFromDatabase() {
        List<ProductExpiryTable> expiringProductsList = new ArrayList<>();
        String sql = "SELECT p.MedicineID, p.MedicineName, pd.ManufacturingDate, pd.ExpireDate "
                + "FROM tblProductDetails pd "
                + "JOIN tblProduct p ON pd.MedicineID = p.MedicineID";

        try (Connection conn = DB.ConnectDB.getConnectDB(); PreparedStatement pst = conn.prepareStatement(sql); ResultSet rs = pst.executeQuery()) {

            while (rs.next()) {
                String medicineID = rs.getString("MedicineID");
                String medicineName = rs.getString("MedicineName");
                LocalDate manufacturingDate = rs.getDate("ManufacturingDate").toLocalDate();
                LocalDate expireDate = rs.getDate("ExpireDate").toLocalDate();

                long daysLeft = ChronoUnit.DAYS.between(LocalDate.now(), expireDate);

                if (daysLeft >= 0 && daysLeft <= 30) {
                    // Tính số ngày còn lại đến ngày hết hạn
                    long daysUntilExpired = ChronoUnit.DAYS.between(LocalDate.now(), expireDate);

                    // Tạo đối tượng Product và thêm vào danh sách
                    ProductExpiryTable product = new ProductExpiryTable(medicineID, medicineName, manufacturingDate, expireDate, daysUntilExpired);
                    expiringProductsList.add(product);
                }
            }

        } catch (Exception e) {
            System.out.println("Error fetching product details: " + e.getMessage());
        }

        return expiringProductsList;
    }

    private void updateTime() {
        LocalDateTime now = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm d MMMM yyyy");
        String formattedDateTime = now.format(formatter);
        Time.setText(formattedDateTime); // Đặt giá trị vào Label có fx:id là Time
    }

        public void loadFXML(String fxml) {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource(fxml));
            Parent root = loader.load();

            main.getChildren().clear();
            main.getChildren().add(root);

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

    }

}
```

6 Add Customer

```
1  public class AddcustomerController implements Initializable {
2
3      private CustomerUpdateListener listener;
4      @FXML
5      private ImageView exit;
6      @FXML
7      private Button btnadd;
8
9      @FXML
10     private TextField txtname;
11
12     @FXML
13     private TextField txtaddress;
14
15     @FXML
16     private RadioButton rdmale;
17
18     @FXML
19     private RadioButton rdfemale;
20
21     @FXML
22     private TextField txtphone;
23
24     @FXML
25     private TextField txtemail;
26
27     @FXML
28     private TextArea txtpoint;
29
30     public void setCustomerUpdateListener(CustomerUpdateListener listener) {
31         this.listener = listener;
32     }
33
```

6.1 Code

```
@FXML
    private void addCustomer() {
        AlertMessage alert = new AlertMessage();
        if (txtname.getText().isEmpty() || txtaddress.getText().isEmpty() || txtphone.getText().isEmpty() || txtemail.getText().isEmpty()
                || txtpoint.getText().isEmpty()) {
            alert.errorMessage("Please enter complete information");
            return;
        }

        // Kiểm tra xem 'Nam' hoặc 'Nữ' đã được chọn chưa
        if (!(rdmale.isSelected() ^ rdfemale.isSelected())) {
            alert.errorMessage("Please select 'Male' or 'Female'");
            return;
        }

        // Kiểm tra định dạng email có hợp lệ không
        if (!txtemail.getText().matches("^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$")) {
            alert.errorMessage("Please enter the correct Email");
            return;
        }

        // Kiểm tra xem số điện thoại đã tồn tại chưa
        String checkPhoneQuery = "SELECT * FROM Customer WHERE Phone = ?";
        try {

            Connection conn = DB.ConnectDB.getConnectDB();

            PreparedStatement checkPhoneStmt = conn.prepareStatement(checkPhoneQuery);
            checkPhoneStmt.setString(1, txtphone.getText());
            ResultSet existingPhone = checkPhoneStmt.executeQuery();
            if (existingPhone.next()) {
                alert.errorMessage("Phone number already exists");
                return; // Thoát phương thức nếu số điện thoại đã tồn tại
            }

            // Kiểm tra xem email đã tồn tại chưa
            String checkEmailQuery = "SELECT * FROM Customer WHERE Email = ?";
            PreparedStatement checkEmailStmt = conn.prepareStatement(checkEmailQuery);
            checkEmailStmt.setString(1, txtemail.getText());
            ResultSet existingEmail = checkEmailStmt.executeQuery();
            if (existingEmail.next()) {
                alert.errorMessage("Email address already exists");
                return; // Thoát phương thức nếu địa chỉ email đã tồn tại
            }

            // Nếu các kiểm tra đều đúng, tiến hành thêm vào cơ sở dữ liệu
            String sql = "INSERT INTO Customer(CustomerName, Address, Gender, Phone, Email, Point) VALUES (?, ?, ?, ?, ?, ?)";
            PreparedStatement pst = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
            pst.setString(1, txtname.getText());
            pst.setString(2, txtaddress.getText());
            String gender = rdmale.isSelected() ? "Male" : "Female";
            pst.setString(3, gender);
            pst.setString(4, txtphone.getText());
            pst.setString(5, txtemail.getText());
            pst.setString(6, txtpoint.getText());
            pst.executeUpdate();

```

```
ResultSet rs = pst.getGeneratedKeys();
            if (rs.next()) {
                int id = rs.getInt(1);
                String formattedId = String.format("%04d", id);
                TableCustomer newCustomer = new TableCustomer(formattedId,
                        txtname.getText(),
                        txtaddress.getText(),
                        gender,
                        txtpoint.getText(),
                        txtphone.getText(),
                        txtemail.getText(),
                        "Hành động");
                if (listener != null) {
                    listener.onUpdate(); // Thông báo cho CustomerController cập nhật dữ liệu
                }
                alert.successMessage("Successfully added customers");
                // Đóng form
                Stage stage = (Stage) btnadd.getScene().getWindow();
                stage.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @FXML
    public void handleCancel() {
        Stage stage = (Stage) txtname.getScene().getWindow();
        stage.close();
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        exit.setOnMouseClicked(event -> {
            Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
            stage.close();
        });
    }
```

codesnap.dev

7 Bills

```java
public class BillsController implements Initializable {

    @FXML
    private Button Export;
    @FXML
    private TextField OrderSearchItem;
    @FXML
    private Text TotalPrice;
    @FXML
    private TableView<TableBills> tableBills;

    @FXML
    private TableColumn<TableBills, String> columnBillID;
    @FXML
    private TableColumn<TableBills, String> columnPhone;
    @FXML
    private TableColumn<TableBills, String> columnCustomerName;

    @FXML
    private TableColumn<TableBills, Integer> columnTotalPrice;

    @FXML
    private TableColumn<TableBills, String> columnPaymentMethod;

    @FXML
    private TableColumn<TableBills, String> columnBillDate;

    @FXML
    private VBox vboxBillItems;

    @FXML
    void Table(MouseEvent event) {

    }

    @FXML
    void osearch_item() {
        String searchText = normalizeSearchText(OrderSearchItem.getText());

        if (searchText.isEmpty()) {
            // If search text is empty, show all bills
            tableBills.setItems(bills);
            return;
        }
        String[] keywords = searchText.split("\\s+");

        ObservableList<TableBills> filteredList = FXCollections.observableArrayList();

        for (TableBills bill : bills) {
            String customerPhone = normalizeSearchText(bill.getCustomerPhone());
            String billID = normalizeSearchText(bill.getBillID());
            boolean matchesAnyKeyword = false;

            // Check if any keyword is found in customer phone or BillID
            for (String keyword : keywords) {
                if (customerPhone.contains(keyword) || billID.contains(keyword)) {
                    matchesAnyKeyword = true;
                    break;
                }
            }

            if (matchesAnyKeyword) {
                filteredList.add(bill);
            }
        }
        // Reset TableView with the filtered list
        tableBills.setItems(filteredList);
    }
```

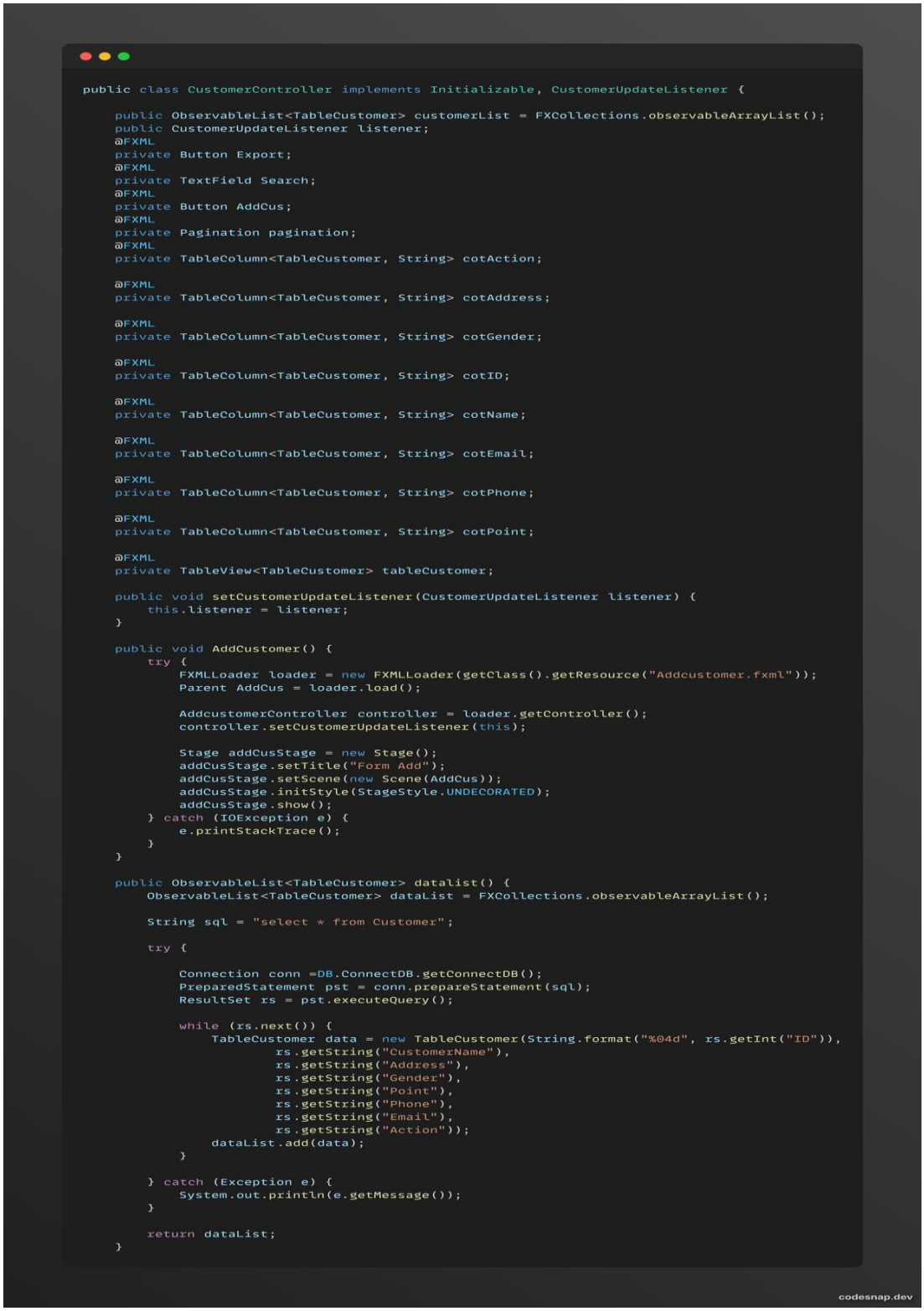
7.1 Code

```java
private String normalizeSearchText(String text) {
        if (text == null) {
            return "";
        }
        // Normalize text: remove accents and convert to lowercase
        String normalizedText = Normalizer.normalize(text, Normalizer.Form.NFD)
                .replaceAll("\\p{InCombiningDiacriticalMarks}+", "");
        return normalizedText.toLowerCase().trim();
    }

    private ObservableList<TableBills> bills = FXCollections.observableArrayList();
    private ObservableList<TableBillsItem> billItems = FXCollections.observableArrayList();

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

    private void initializeBillTable() {
        columnBillID.setCellValueFactory(new PropertyValueFactory<>("billID"));
        columnCustomerName.setCellValueFactory(new PropertyValueFactory<>("customerName"));
        columnPhone.setCellValueFactory(new PropertyValueFactory<>("customerPhone"));
        columnTotalPrice.setCellValueFactory(new PropertyValueFactory<>("totalPrice"));
        columnPaymentMethod.setCellValueFactory(new PropertyValueFactory<>("paymentMethod"));
        columnBillDate.setCellValueFactory(new PropertyValueFactory<>("billDate"));

        loadBillsFromDatabase();
        tableBills.setItems(bills);

        // Double-click to show bill items
        tableBills.setOnMouseClicked(event -> {
            if (event.getClickCount() == 2) {
                TableBills selectedBill = tableBills.getSelectionModel().getSelectedItem();
                if (selectedBill != null) {
                    loadBillItemsFromDatabase(selectedBill.getBillID());
                    displayTotalPayment(selectedBill.getTotalPrice());
                }
            }
        });
    }

    private void displayTotalPayment(double totalPayment) {
        TotalPrice.setText(String.format("%.1f đ", totalPayment));
        TotalPrice.setStyle("-fx-font-size: 20px;");
    }

    private void loadBillsFromDatabase() {
        bills.clear();
        String sql = "SELECT * FROM Bills";
        try {

            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement pst = conn.prepareStatement(sql);
            ResultSet rs = pst.executeQuery();

            while (rs.next()) {
                java.sql.Date billDateSql = rs.getDate("BillDate");
                TableBills bill = new TableBills(
                        rs.getString("BillID"),
                        rs.getString("CustomerId"),
                        rs.getString("CustomerName"),
                        rs.getString("PhoneCustomer"),
                        rs.getInt("TotalPrice"),
                        rs.getString("PaymentMethod"),
                        billDateSql // Giữ nguyên là java.sql.Date
                );
                bills.add(bill);
            }

            pst.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

```

```java
private void loadBillItemsFromDatabase(String billID) {
        vboxBillItems.getChildren().clear(); // Clear previous items
        String sql = "SELECT * FROM BillItems WHERE BillID = ?";
        try {

            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement pst = conn.prepareStatement(sql);
            pst.setString(1, billID);
            ResultSet rs = pst.executeQuery();

            while (rs.next()) {
                String itemName = rs.getString("ItemName");
                int quantity = rs.getInt("Quantity");
                int unitPrice = rs.getInt("UnitPrice");
                String itemType = rs.getString("ItemType");
                String medicineID = rs.getString("MedicineID");

                TableBillsItem billItem = new TableBillsItem(itemName, quantity, unitPrice, itemType ,medicineID);
                AnchorPane pane = createBillItemPane(billItem);
                vboxBillItems.getChildren().add(pane);
            }

            pst.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private AnchorPane createBillItemPane(TableBillsItem billItem) {
        AnchorPane anchorPane = new AnchorPane();
        anchorPane.setPrefHeight(75.0);
        anchorPane.setPrefWidth(285.0);
        anchorPane.setStyle("-fx-border-color: blue; -fx-border-radius: 10px; -fx-border-width: 1px;");
        anchorPane.setMinWidth(285.0);
        anchorPane.setMaxWidth(285.0);
        anchorPane.setMinHeight(75.0);
        anchorPane.setMaxHeight(75.0);

        // Tạo các thành phần trong AnchorPane
        Text itemNameText = new Text(billItem.getItemName());
        itemNameText.setId("itemNameText");
        itemNameText.setLayoutX(14.0);
        itemNameText.setLayoutY(18.0);
        itemNameText.setFont(Font.font("Arial Black", 12.0));
        anchorPane.getChildren().add(itemNameText);

        Text medicineIdText = new Text(billItem.getMedicineID());
        medicineIdText.setId("medicineID");
        medicineIdText.setLayoutX(200.0);
        medicineIdText.setLayoutY(18.0);
        medicineIdText.setFont(Font.font("Arial Black", 12.0));
        anchorPane.getChildren().add(medicineIdText);

        Text quantityText = new Text("Quantity");
        quantityText.setLayoutX(14.0);
        quantityText.setLayoutY(41.0);
        anchorPane.getChildren().add(quantityText);

        TextField quantityField = new TextField(String.valueOf(billItem.getQuantity()));
        quantityField.setId("quantityField");
        quantityField.setLayoutX(64.0);
        quantityField.setLayoutY(20.0);
        quantityField.setPrefHeight(18.0);
        quantityField.setPrefWidth(45.0);
        quantityField.setStyle("-fx-text-fill: black; -fx-border-color: black; -fx-border-width: 1px; -fx-border-radius: 2px; -fx-alignment: center;");
        anchorPane.getChildren().add(quantityField);

        Text unitText = new Text(billItem.getItemtype());
        unitText.setLayoutX(115.0);
        unitText.setLayoutY(41.0);
        anchorPane.getChildren().add(unitText);
        Text priceText = new Text("Price:");
        priceText.setLayoutX(170.0);
        priceText.setLayoutY(67.0);
        priceText.setFont(Font.font("Arial Bold", 14.0));
        anchorPane.getChildren().add(priceText);

        Text itemPriceText = new Text(billItem.getUnitPrice() + "đ");
        itemPriceText.setId("unitPriceText");
        itemPriceText.setLayoutX(220.0);
        itemPriceText.setLayoutY(67.0);
        itemPriceText.setFont(Font.font("Arial Bold", 14.0));
        anchorPane.getChildren().add(itemPriceText);

        VBox.setMargin(anchorPane, new Insets(2.0));

        return anchorPane;
    }

```

```java
private int totalBills;
    private int totalRevenue;

    public void updateTotalBillsAndRevenue() {
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
        String currentDate = sdf.format(new Date());

        String countSql = "SELECT COUNT(*) AS TotalBills FROM Bills WHERE BillDate = ?";
        String sumSql = "SELECT SUM(TotalPrice) AS TotalRevenue FROM Bills WHERE BillDate = ?";

        try {

            Connection conn = DB.ConnectDB.getConnectDB();

            PreparedStatement countPst = conn.prepareStatement(countSql);
            countPst.setString(1, currentDate);
            ResultSet countRs = countPst.executeQuery();
            if (countRs.next()) {
                totalBills = countRs.getInt("TotalBills");
            }

            countPst.close();

            // Sum total revenue
            PreparedStatement sumPst = conn.prepareStatement(sumSql);
            sumPst.setString(1, currentDate);
            ResultSet sumRs = sumPst.executeQuery();
            if (sumRs.next()) {
                totalRevenue = sumRs.getInt("TotalRevenue");
            }

            sumPst.close();
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    // Getter methods for totalBills and totalRevenue
    public int getTotalBills() {
        return totalBills;
    }

    public int getTotalRevenue() {
        return totalRevenue;
    }

    public void exportToPDF() {
        AlertMessage alert = new AlertMessage();
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save PDF");
        String fontPath = "src/font/arial.ttf";
        fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("PDF Files", "*.pdf"));
        File selectedFile = fileChooser.showSaveDialog(Export.getScene().getWindow());

        if (selectedFile != null) {
            try {

                PdfFont font = PdfFontFactory.createFont(fontPath, PdfEncodings.IDENTITY_H, true);

                PdfWriter writer = new PdfWriter(selectedFile.getAbsolutePath());
                PdfDocument pdf = new PdfDocument(writer);
                Document document = new Document(pdf);

                Paragraph title = new Paragraph("Bills Transaction").setBold().setFontSize(18);
                document.add(title);

                // Tạo tiêu đề cho các cột


                float[] columnWidths = {100, 100, 100, 100, 100, 100};
                Table table = new Table(columnWidths);

                TableView<TableBills> tableView = tableBills;
                for (TableColumn<TableBills, ?> column : tableView.getColumns()) {

                        table.addCell(new Cell().add(new Paragraph(column.getText()).setFont(font)));

                }


                ObservableList<TableBills> dataList = tableView.getItems();
                for (TableBills data : dataList) {
                    table.addCell(new Cell().add(new Paragraph(data.getBillID())));
                    table.addCell(new Cell().add(new Paragraph(data.getCustomerName())));
                    table.addCell(new Cell().add(new Paragraph(data.getCustomerPhone())));
                    table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getTotalPrice()))));
                    table.addCell(new Cell().add(new Paragraph(data.getPaymentMethod())));
                    table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getBillDate()))));

                }
                document.add(table);
                document.close();
                Desktop.getDesktop().open(selectedFile);

                alert.successMessage("Export successful!");
            } catch (IOException e) {
                alert.errorMessage("Export failed: " + e.getMessage());
            }
        }
    }
}
```

8 Customer



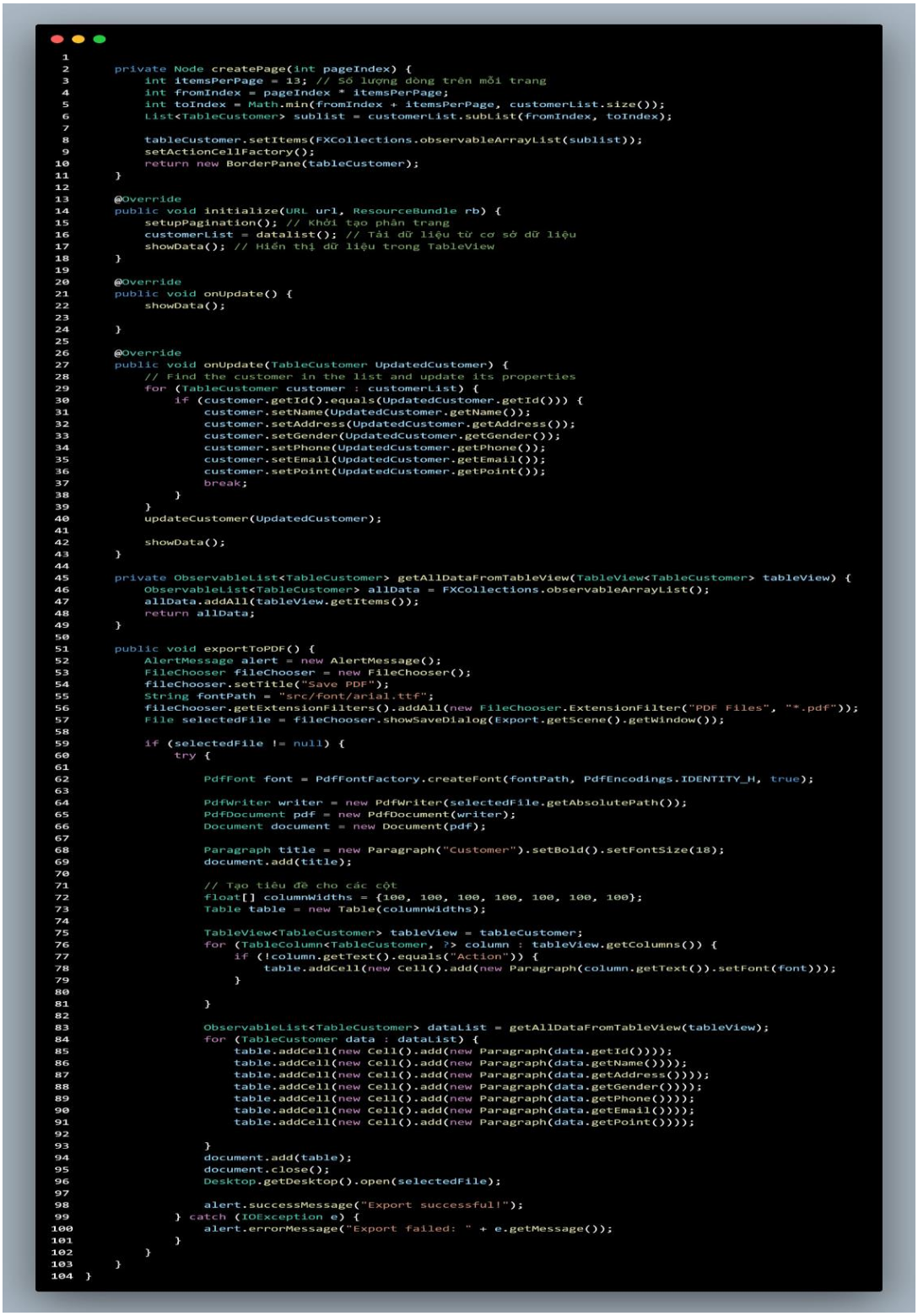
```
public class CustomerController implements Initializable, CustomerUpdateListener {

    public ObservableList<TableCustomer> customerList = FXCollections.observableArrayList();
    public CustomerUpdateListener listener;
    @FXML
    private Button Export;
    @FXML
    private TextField Search;
    @FXML
    private Button AddCus;
    @FXML
    private Pagination pagination;
    @FXML
    private TableColumn<TableCustomer, String> cotAction;

    @FXML
    private TableColumn<TableCustomer, String> cotAddress;

    @FXML
    private TableColumn<TableCustomer, String> cotGender;

    @FXML
    private TableColumn<TableCustomer, String> cotID;

    @FXML
    private TableColumn<TableCustomer, String> cotName;

    @FXML
    private TableColumn<TableCustomer, String> cotEmail;

    @FXML
    private TableColumn<TableCustomer, String> cotPhone;

    @FXML
    private TableColumn<TableCustomer, String> cotPoint;

    @FXML
    private TableView<TableCustomer> tableCustomer;

    public void setCustomerUpdateListener(CustomerUpdateListener listener) {
        this.listener = listener;
    }

    public void AddCustomer() {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("Addcustomer.fxml"));
            Parent AddCus = loader.load();

            AddcustomerController controller = loader.getController();
            controller.setCustomerUpdateListener(this);

            Stage addCusStage = new Stage();
            addCusStage.setTitle("Form Add");
            addCusStage.setScene(new Scene(AddCus));
            addCusStage.initStyle(StageStyle.UNDECORATED);
            addCusStage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public ObservableList<TableCustomer> datalist() {
        ObservableList<TableCustomer> dataList = FXCollections.observableArrayList();

        String sql = "select * from Customer";

        try {

            Connection conn =DB.ConnectDB.getConnectDB();
            PreparedStatement pst = conn.prepareStatement(sql);
            ResultSet rs = pst.executeQuery();

            while (rs.next()) {
                TableCustomer data = new TableCustomer(String.format("%04d", rs.getInt("ID")),
                        rs.getString("CustomerName"),
                        rs.getString("Address"),
                        rs.getString("Gender"),
                        rs.getString("Point"),
                        rs.getString("Phone"),
                        rs.getString("Email"),
                        rs.getString("Action"));
                dataList.add(data);
            }

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        return dataList;
    }
```

8.1 Code

```java
public void showData() {
        // Update observable list from database
        customerList = datalist();

        // Set cell value factories and cell factories
        cotID.setCellValueFactory(new PropertyValueFactory<>("id"));
        cotName.setCellValueFactory(new PropertyValueFactory<>("name"));
        cotAddress.setCellValueFactory(new PropertyValueFactory<>("address"));
        cotGender.setCellValueFactory(new PropertyValueFactory<>("gender"));
        cotPhone.setCellValueFactory(new PropertyValueFactory<>("phone"));
        cotEmail.setCellValueFactory(new PropertyValueFactory<>("email"));
        cotPoint.setCellValueFactory(new PropertyValueFactory<>("point"));
        cotAction.setCellValueFactory(new PropertyValueFactory<>("action"));

        // Set cell factories for action buttons
        setActionCellFactory();

        // Set TableView items and height
        tableCustomer.setItems(customerList);
        tableCustomer.setFixedCellSize(40); // Fixed cell height
        tableCustomer.prefHeightProperty().bind(Bindings.size(tableCustomer.getItems()).multiply(tableCustomer.getFixedCellSize()).add(30));

        // Update pagination and set to first page
        pagination.setPageCount(getPageCount());
        pagination.setPageFactory(this::createPage);
        pagination.setCurrentPageIndex(0);
    }

    public void setActionCellFactory() {
        Callback<TableColumn<TableCustomer, String>, TableCell<TableCustomer, String>> cellFactory
                = new Callback<TableColumn<TableCustomer, String>, TableCell<TableCustomer, String>>() {
            @Override
            public TableCell<TableCustomer, String> call(final TableColumn<TableCustomer, String> param) {
                final TableCell<TableCustomer, String> cell = new TableCell<>() {
                    private final Button updateButton = new Button("Update");
                    private final Button deleteButton = new Button("Delete");
                    private final HBox hbox = new HBox(10, updateButton, deleteButton);

                    {
                        // Button styling and actions
                        updateButton.getStyleClass().add("update-button");
                        deleteButton.getStyleClass().add("delete-button");

                        updateButton.setOnAction(event -> {
                            TableCustomer customer = getTableView().getItems().get(getIndex());
                            try {
                                FXMLLoader loader = new FXMLLoader(getClass().getResource("UpdateCustomer.fxml"));
                                Parent updateCustomerParent = loader.load();


                                UpdateCustomerController controller = loader.getController();
                                controller.setCustomer(customer);
                                controller.setCustomerUpdateListener(CustomerController.this); // Pass listener reference

                                Stage updateStage = new Stage();
                                updateStage.setTitle("Update Customer");
                                updateStage.setScene(new Scene(updateCustomerParent));
                                updateStage.initStyle(StageStyle.UNDECORATED);
                                updateStage.show();

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

                        deleteButton.setOnAction(event -> {
                            TableCustomer customer = getTableView().getItems().get(getIndex());
                            deleteCustomer(customer);
                        });
                        StackPane.setAlignment(updateButton, Pos.CENTER);
                        StackPane.setAlignment(deleteButton, Pos.CENTER);
                    }

                    @Override
                    public void updateItem(String item, boolean empty) {
                        super.updateItem(item, empty);

                        // Set the cell's graphic based on whether the cell is empty
                        if (empty) {
                            setGraphic(null);
                        } else {
                            setGraphic(hbox);
                        }
                    }
                };

                return cell;
            }
        };

        // Set cell factory for action column
        cotAction.setCellFactory(cellFactory);
    }
```

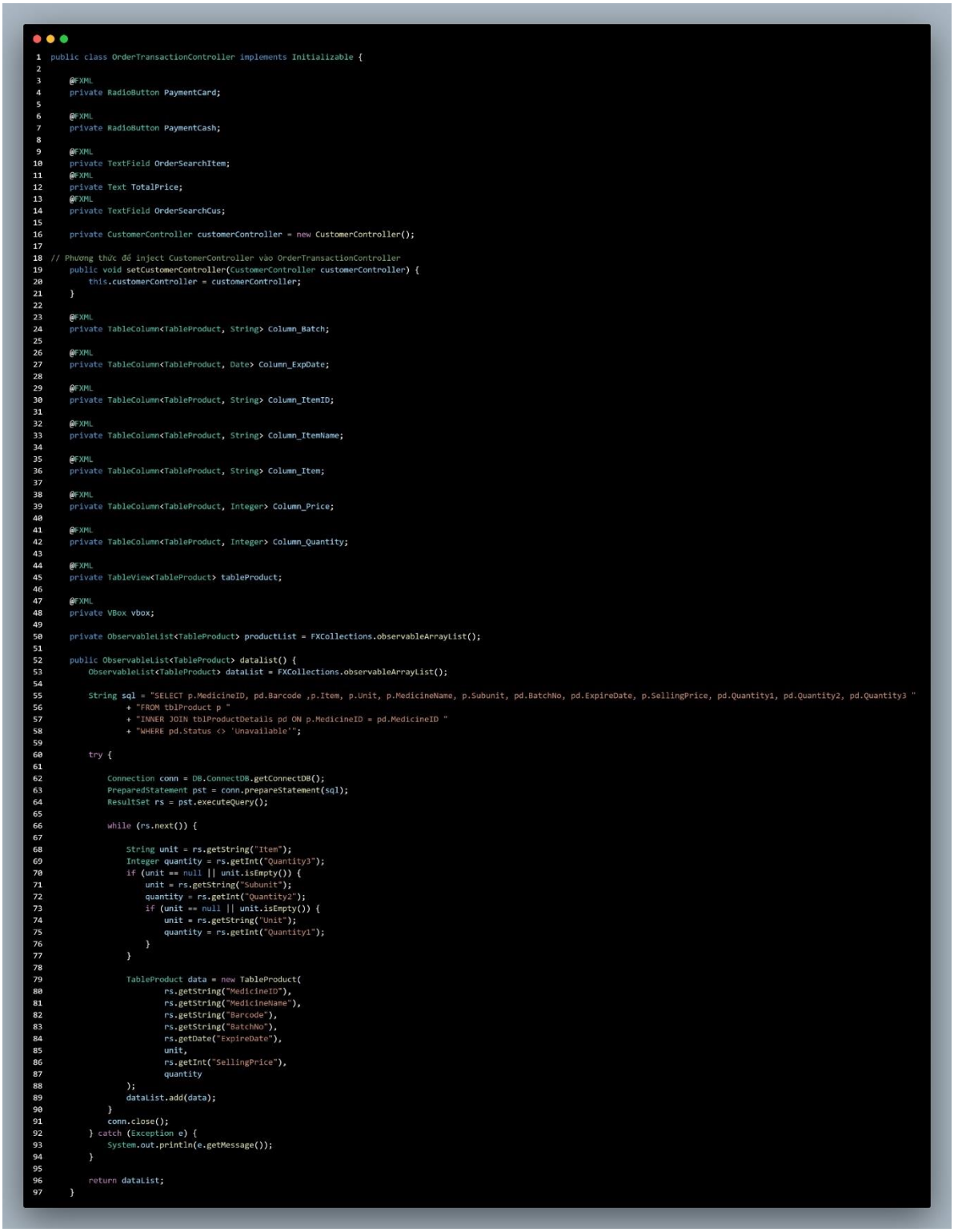
```java
    @FXML
    public void deleteCustomer(TableCustomer customer) {
        AlertMessage alert = new AlertMessage();
        String sql = "DELETE FROM Customer WHERE ID = ?";
        try {

            Connection conn =DB.ConnectDB.getConnectDB();
            PreparedStatement pst = conn.prepareStatement(sql);
            pst.setInt(1, Integer.parseInt(customer.getId()));
            pst.executeUpdate();

            // Remove customer from ObservableList and update TableView
            customerList.remove(customer);
            showData(); // Update TableView
            alert.successMessage("Customer deleted successfully");

        } catch (Exception e) {
            e.printStackTrace();
            alert.errorMessage("Error deleting customer: " + e.getMessage());
        }
    }

    @FXML
    public void updateCustomer(TableCustomer updatedCustomer) {
        AlertMessage alert = new AlertMessage();
        String sql = "UPDATE Customer SET CustomerName=?, Address=?, Gender=?, Phone=?, Email=?, Point=? WHERE ID=?";

        try {

            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement pst = conn.prepareStatement(sql);
            pst.setString(1, updatedCustomer.getName());
            pst.setString(2, updatedCustomer.getAddress());
            pst.setString(3, updatedCustomer.getGender());
            pst.setString(4, updatedCustomer.getPhone());
            pst.setString(5, updatedCustomer.getEmail());
            pst.setString(6, updatedCustomer.getPoint());
//            pst.setString(7, updatedCustomer.getAction());
            pst.setInt(7, Integer.parseInt(updatedCustomer.getId())); // ID is int type in database

            int affectedRows = pst.executeUpdate();
            if (affectedRows > 0) {

                if (listener != null) {
                    listener.onUpdate(updatedCustomer); // Notify listener to update TableView
                }
            } else {
                System.out.println("Failed to update customer.");
            }

        } catch (SQLException e) {
            System.out.println("Error updating customer: " + e.getMessage());
        }
    }

//    private void refreshTableView() {
//        // Refresh the TableView to display updated data
//        tableCustomer.getItems().clear();
//        tableCustomer.getItems().addAll(customerList);
//    }
    @FXML
    private void handleSearch() {
        String searchText = normalizeSearchText(Search.getText());

        if (searchText.isEmpty()) {
            // If search text is empty, show all customers
            tableCustomer.setItems(customerList);
            pagination.setPageFactory(this::createPage);
            return;
        }
        String[] keywords = searchText.split("\\s+");

        ObservableList<TableCustomer> filteredList = FXCollections.observableArrayList();

        for (TableCustomer customer : customerList) {
            String customerName = normalizeSearchText(customer.getName());
            boolean matchesAllKeywords = true;

            // Check if all keywords are found in customer name
            for (String keyword : keywords) {
                if (!customerName.contains(keyword)) {
                    matchesAllKeywords = false;
                    break;
                }
            }

            if (matchesAllKeywords) {
                filteredList.add(customer);
            }
        }

        // Reset TableView with the filtered list
        tableCustomer.setItems(filteredList);

    }
```

```java
private String normalizeSearchText(String text) {
        // Normalize text: remove accents and convert to lowercase
        String normalizedText = Normalizer.normalize(text, Normalizer.Form.NFD)
                .replaceAll("\\p{InCombiningDiacriticalMarks}+", "");
        return normalizedText.toLowerCase().trim();
    }

    public void refreshCustomerList() {
        customerList.clear(); // Xóa danh sách khách hàng hiện tại
        customerList.addAll(datalist()); // Cập nhật lại danh sách từ cơ sở dữ liệu

    }

    public TableCustomer findCustomerByPhone(String phone) {
        refreshCustomerList();
        for (TableCustomer customer : customerList) {
            if (customer.getPhone().equals(phone)) {
                return customer;
            }
        }
        return null; // Trả về null nếu không tìm thấy khách hàng
    }

    public void updateCustomerPoints(String phoneNumber, int pointsToAdd) {
        // Tìm khách hàng dựa trên số điện thoại
        TableCustomer customer = findCustomerByPhone(phoneNumber);
        if (customer != null) {
            // Cập nhật điểm tích lũy cho khách hàng
            int currentPoints = Integer.parseInt(customer.getPoint());
            int newPoints = currentPoints + pointsToAdd;

            // Update điểm tích lũy vào cơ sở dữ liệu
            String sql = "UPDATE Customer SET Point = ? WHERE Phone = ?";
            try {

                Connection conn = DB.ConnectDB.getConnectDB();
                PreparedStatement pst = conn.prepareStatement(sql);
                pst.setInt(1, newPoints);
                pst.setString(2, phoneNumber);
                pst.executeUpdate();

            } catch (SQLException e) {
                System.out.println(e.getMessage());
            }
        } else {
            System.out.println("Không tìm thấy khách hàng với số điện thoại " + phoneNumber);
        }
    }

    private static final int itemsPerPage = 13; // Số lượng mục trên mỗi trang

    private ObservableList<TableCustomer> paginatedData; // Danh sách dữ liệu được phân trang

    private void setupPagination() {
        int pageCount = getPageCount();
        pagination.setPageCount(pageCount); // Thiết lập số lượng trang

        pagination.setPageFactory(this::createPage); // Thiết lập page factory để tạo nội dung từng trang
    }

    private int getPageCount() {
        // Return the number of pages based on your data and itemsPerPage
        // For example:
        int itemsPerPage = 13; // Số lượng dòng trên mỗi trang
        int pageCount = (int) Math.ceil((double) customerList.size() / itemsPerPage);
        return pageCount;
    }
```



```java

    private Node createPage(int pageIndex) {
        int itemsPerPage = 13; // Số lượng dòng trên mỗi trang
        int fromIndex = pageIndex * itemsPerPage;
        int toIndex = Math.min(fromIndex + itemsPerPage, customerList.size());
        List<TableCustomer> sublist = customerList.subList(fromIndex, toIndex);

        tableCustomer.setItems(FXCollections.observableArrayList(sublist));
        setActionCellFactory();
        return new BorderPane(tableCustomer);
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        setupPagination(); // Khởi tạo phân trang
        customerList = datalist(); // Tải dữ liệu từ cơ sở dữ liệu
        showData(); // Hiển thị dữ liệu trong TableView
    }

    @Override
    public void onUpdate() {
        showData();

    }

    @Override
    public void onUpdate(TableCustomer UpdatedCustomer) {
        // Find the customer in the list and update its properties
        for (TableCustomer customer : customerList) {
            if (customer.getId().equals(UpdatedCustomer.getId())) {
                customer.setName(UpdatedCustomer.getName());
                customer.setAddress(UpdatedCustomer.getAddress());
                customer.setGender(UpdatedCustomer.getGender());
                customer.setPhone(UpdatedCustomer.getPhone());
                customer.setEmail(UpdatedCustomer.getEmail());
                customer.setPoint(UpdatedCustomer.getPoint());
                break;
            }
        }
        updateCustomer(UpdatedCustomer);

        showData();
    }

    private ObservableList<TableCustomer> getAllDataFromTableView(TableView<TableCustomer> tableView) {
        ObservableList<TableCustomer> allData = FXCollections.observableArrayList();
        allData.addAll(tableView.getItems());
        return allData;
    }

    public void exportToPDF() {
        AlertMessage alert = new AlertMessage();
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save PDF");
        String fontPath = "src/font/arial.ttf";
        fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("PDF Files", "*.pdf"));
        File selectedFile = fileChooser.showSaveDialog(Export.getScene().getWindow());

        if (selectedFile != null) {
            try {

                PdfFont font = PdfFontFactory.createFont(fontPath, PdfEncodings.IDENTITY_H, true);

                PdfWriter writer = new PdfWriter(selectedFile.getAbsolutePath());
                PdfDocument pdf = new PdfDocument(writer);
                Document document = new Document(pdf);

                Paragraph title = new Paragraph("Customer").setBold().setFontSize(18);
                document.add(title);

                // Tạo tiêu đề cho các cột
                float[] columnWidths = {100, 100, 100, 100, 100, 100, 100};
                Table table = new Table(columnWidths);

                TableView<TableCustomer> tableView = tableCustomer;
                for (TableColumn<TableCustomer, ?> column : tableView.getColumns()) {
                    if (!column.getText().equals("Action")) {
                        table.addCell(new Cell().add(new Paragraph(column.getText()).setFont(font)));
                    }

                }

                ObservableList<TableCustomer> dataList = getAllDataFromTableView(tableView);
                for (TableCustomer data : dataList) {
                    table.addCell(new Cell().add(new Paragraph(data.getId())));
                    table.addCell(new Cell().add(new Paragraph(data.getName())));
                    table.addCell(new Cell().add(new Paragraph(data.getAddress())));
                    table.addCell(new Cell().add(new Paragraph(data.getGender())));
                    table.addCell(new Cell().add(new Paragraph(data.getPhone())));
                    table.addCell(new Cell().add(new Paragraph(data.getEmail())));
                    table.addCell(new Cell().add(new Paragraph(data.getPoint())));

                }
                document.add(table);
                document.close();
                Desktop.getDesktop().open(selectedFile);

                alert.successMessage("Export successful!");
            } catch (IOException e) {
                alert.errorMessage("Export failed: " + e.getMessage());
            }
        }
    }
}
```

9 Order Transaction



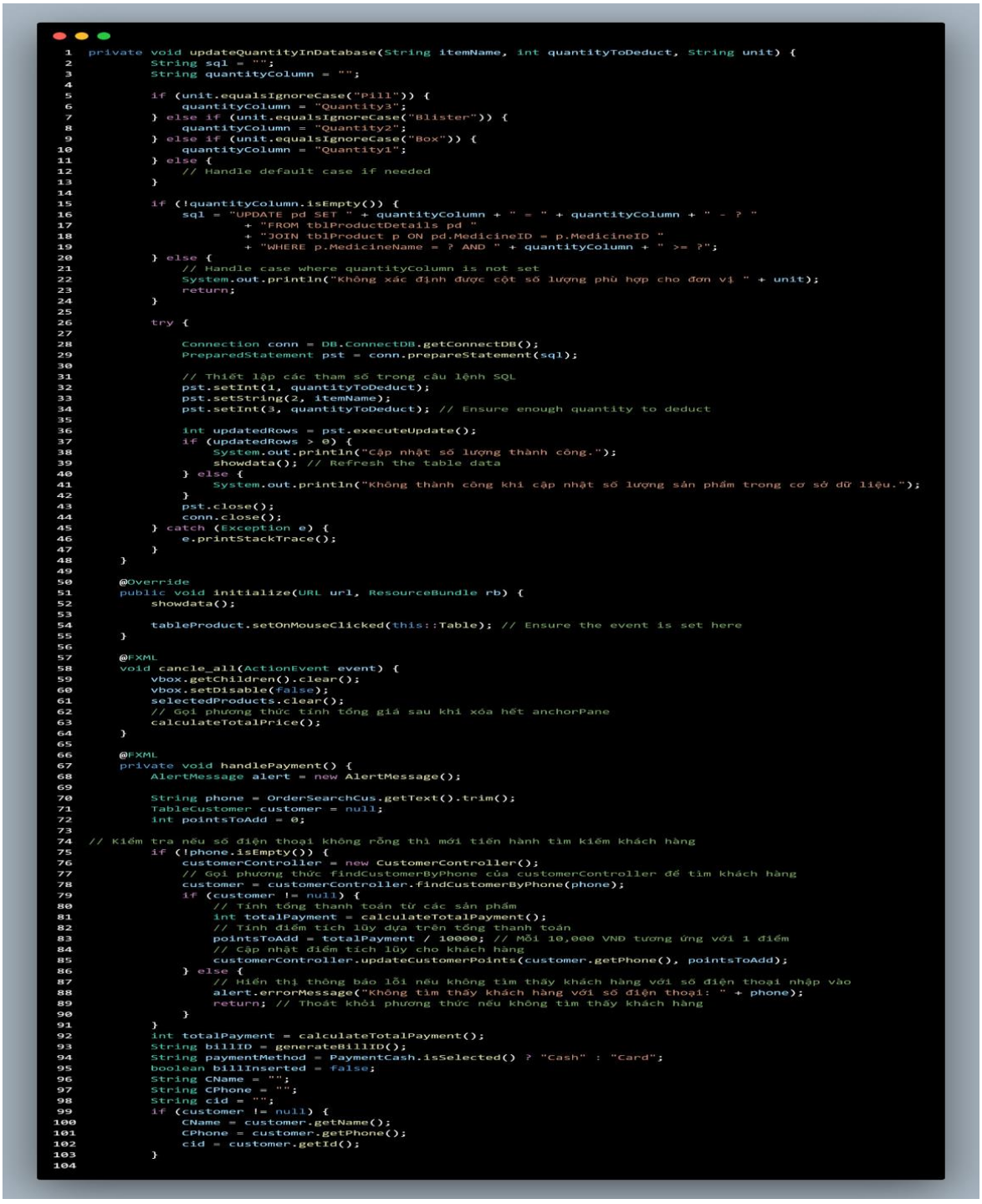
```java
public class OrderTransactionController implements Initializable {

    @FXML
    private RadioButton PaymentCard;

    @FXML
    private RadioButton PaymentCash;

    @FXML
    private TextField OrderSearchItem;
    @FXML
    private Text TotalPrice;
    @FXML
    private TextField OrderSearchCus;

    private CustomerController customerController = new CustomerController();

// Phương thức để inject CustomerController vào OrderTransactionController
    public void setCustomerController(CustomerController customerController) {
        this.customerController = customerController;
    }

    @FXML
    private TableColumn<TableProduct, String> Column_Batch;

    @FXML
    private TableColumn<TableProduct, Date> Column_ExpDate;

    @FXML
    private TableColumn<TableProduct, String> Column_ItemID;

    @FXML
    private TableColumn<TableProduct, String> Column_ItemName;

    @FXML
    private TableColumn<TableProduct, String> Column_Item;

    @FXML
    private TableColumn<TableProduct, Integer> Column_Price;

    @FXML
    private TableColumn<TableProduct, Integer> Column_Quantity;

    @FXML
    private TableView<TableProduct> tableProduct;

    @FXML
    private VBox vbox;

    private ObservableList<TableProduct> productList = FXCollections.observableArrayList();

    public ObservableList<TableProduct> datalist() {
        ObservableList<TableProduct> dataList = FXCollections.observableArrayList();

        String sql = "SELECT p.MedicineID, pd.Barcode ,p.Item, p.Unit, p.MedicineName, p.Subunit, pd.BatchNo, pd.ExpireDate, p.SellingPrice, pd.Quantity1, pd.Quantity2, pd.Quantity3 "
                + "FROM tblProduct p "
                + "INNER JOIN tblProductDetails pd ON p.MedicineID = pd.MedicineID "
                + "WHERE pd.Status <> 'Unavailable'";

        try {

            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement pst = conn.prepareStatement(sql);
            ResultSet rs = pst.executeQuery();

            while (rs.next()) {

                String unit = rs.getString("Item");
                Integer quantity = rs.getInt("Quantity3");
                if (unit == null || unit.isEmpty()) {
                    unit = rs.getString("Subunit");
                    quantity = rs.getInt("Quantity2");
                    if (unit == null || unit.isEmpty()) {
                        unit = rs.getString("Unit");
                        quantity = rs.getInt("Quantity1");
                    }
                }

                TableProduct data = new TableProduct(
                        rs.getString("MedicineID"),
                        rs.getString("MedicineName"),
                        rs.getString("Barcode"),
                        rs.getString("BatchNo"),
                        rs.getDate("ExpireDate"),
                        unit,
                        rs.getInt("SellingPrice"),
                        quantity
                );
                dataList.add(data);
            }
            conn.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }

        return dataList;
    }
```

9.1 Code

```java
private void showdata() {
        productList = datalist();
        Column_ItemID.setCellValueFactory(new PropertyValueFactory<>("productId"));
        Column_ItemName.setCellValueFactory(new PropertyValueFactory<>("productName"));
        Column_Batch.setCellValueFactory(new PropertyValueFactory<>("batchNo"));
        Column_ExpDate.setCellValueFactory(new PropertyValueFactory<>("expDate"));
        Column_Item.setCellValueFactory(new PropertyValueFactory<>("item"));
        Column_Price.setCellValueFactory(new PropertyValueFactory<>("unitPrice"));
        Column_Quantity.setCellValueFactory(new PropertyValueFactory<>("quantityInStock"));

        tableProduct.setItems(productList);
    }

    public AnchorPane createProductAnchorPane(String itemName, String itemPrice, String itemType, String medicineID) {
        // Tạo AnchorPane mới
        AnchorPane anchorPane = new AnchorPane();
        anchorPane.setPrefHeight(75.0);
        anchorPane.setPrefWidth(290.0);
        anchorPane.setStyle("-fx-border-color: blue; -fx-border-radius: 8px; -fx-border-width: 1px;");
        anchorPane.setMinWidth(290.0);
        anchorPane.setMaxWidth(290.0);
        anchorPane.setMinHeight(75.0);
        anchorPane.setMaxHeight(75.0);

        // Tạo các thành phần trong AnchorPane
        Text itemNameText = new Text(itemName);
        itemNameText.setId("itemNameText");
        itemNameText.setLayoutX(14.0);
        itemNameText.setLayoutY(18.0);
        itemNameText.setFont(Font.font("Arial Black", 12.0));
        anchorPane.getChildren().add(itemNameText);

        Text medicineIdText = new Text(medicineID);
        medicineIdText.setId("medicineID");
        medicineIdText.setLayoutX(200.0);
        medicineIdText.setLayoutY(18.0);
        medicineIdText.setFont(Font.font("Arial Black", 12.0));
        anchorPane.getChildren().add(medicineIdText);

        Text quantityText = new Text("Quantity");
        quantityText.setLayoutX(14.0);
        quantityText.setLayoutY(41.0);
        anchorPane.getChildren().add(quantityText);

        TextField quantityField = new TextField("1");
        quantityField.setId("quantityField");
        quantityField.setLayoutX(64.0);
        quantityField.setLayoutY(20.0);
        quantityField.setPrefHeight(18.0);
        quantityField.setPrefWidth(45.0);
        quantityField.setStyle("-fx-text-fill: black; -fx-border-color: black; -fx-border-width: 1px; -fx-border-radius: 2px; -fx-alignment: center;");
        anchorPane.getChildren().add(quantityField);

        Text unitText = new Text(itemType);
        unitText.setId("itemTypeText");
        unitText.setLayoutX(115.0);
        unitText.setLayoutY(41.0);
        anchorPane.getChildren().add(unitText);

        Text priceText = new Text("Price:");
        priceText.setLayoutX(170.0);
        priceText.setLayoutY(67.0);
        priceText.setFont(Font.font("Arial Bold", 14.0));
        anchorPane.getChildren().add(priceText);

        Text itemPriceText = new Text(itemPrice + "đ");
        itemPriceText.setId("unitPriceText");
        itemPriceText.setLayoutX(220.0);
        itemPriceText.setLayoutY(67.0);
        itemPriceText.setFont(Font.font("Arial Bold", 14.0));
        anchorPane.getChildren().add(itemPriceText);

        FontAwesomeIcon trashIcon = new FontAwesomeIcon();
        trashIcon.setGlyphName("TRASH");
        trashIcon.setLayoutX(14.0);
        trashIcon.setLayoutY(69.0);
        trashIcon.setSize("20px");
        anchorPane.getChildren().add(trashIcon);

        VBox.setMargin(anchorPane, new Insets(2.0));
```

```java
trashIcon.setOnMouseClicked(event -> {
            Node source = (Node) event.getSource();
            AnchorPane paneToRemove = (AnchorPane) source.getParent();

            Text itemNameTextToRemove = (Text) paneToRemove.lookup("#itemNameText");
            if (itemNameTextToRemove != null) {
                String itemNameToRemove = itemNameTextToRemove.getText().trim();

                // Tìm và xóa sản phẩm trong selectedProducts dựa trên productName
                TableProduct productToRemove = null;
                for (TableProduct product : selectedProducts) {
                    if (product.getProductName().equals(itemNameToRemove)) {
                        productToRemove = product;
                        break;
                    }
                }

                if (productToRemove != null) {
                    selectedProducts.remove(productToRemove);
                }

                vbox.getChildren().remove(paneToRemove);
                calculateTotalPrice();

                // Cho phép thêm sản phẩm mới sau khi xóa
                vbox.setDisable(false); // Cho phép sự kiện double-click hoạt động lại
            }
        });

        // Thêm sự kiện thay đổi số lượng
        quantityField.textProperty().addListener((observable, oldValue, newValue) -> {
            try {
                int quantity = Integer.parseInt(newValue);
                if (quantity < 0) {
                    quantityField.setText(oldValue);
                    return;
                }
                double totalPrice = quantity * Double.parseDouble(itemPrice);
                itemPriceText.setText(totalPrice + "đ");

                // Gọi phương thức tính tổng giá sau khi cập nhật giá
                calculateTotalPrice();
            } catch (NumberFormatException e) {
                if (!newValue.isEmpty()) {
                    quantityField.setText(oldValue);
                }
            }
        });

        return anchorPane;
    }

    private List<TableProduct> selectedProducts = new ArrayList<>();

    @FXML
    public void Table(MouseEvent event) {
        if (!vbox.isDisabled() && event.getButton() == MouseButton.PRIMARY && event.getClickCount() == 2) {
            TableProduct product = tableProduct.getSelectionModel().getSelectedItem();
            if (product != null && !selectedProducts.contains(product)) {
                // Tạo AnchorPane cho sản phẩm
                String itemType = product.getItem();
                AnchorPane anchorPane = createProductAnchorPane(product.getProductName(), String.valueOf(product.getUnitPrice()), itemType, product.getProductId());
                vbox.getChildren().add(anchorPane); // Thêm anchorPane vào VBox

                // Thêm sản phẩm vào danh sách đã chọn
                selectedProducts.add(product);

                // Tính toán tổng giá sau khi thêm sản phẩm
                calculateTotalPrice();
            }
        }
    }
```

```java
@FXML
    private void searchItem() {
        String searchItem = OrderSearchItem.getText().trim().replaceAll("\\s+", ""); // Loại bỏ khoảng trắng trong từ khóa tìm kiếm
        productList.clear(); // Xóa dữ liệu cũ

        if (searchItem.isEmpty()) {
            // Nếu không có gì được nhập, hiển thị lại toàn bộ dữ liệu
            productList.addAll(datalist());
        } else {
            // Nếu có từ khóa tìm kiếm, thực hiện truy vấn để lấy dữ liệu tương ứng
            String sql = "SELECT p.MedicineID, pd.Barcode, p.Item ,p.MedicineName, pd.BatchNo, pd.ExpireDate, p.SellingPrice, pd.Quantity1 "
                    + "FROM tblProduct p "
                    + "INNER JOIN tblProductDetails pd ON p.MedicineID = pd.MedicineID";

            try {

                Connection conn = DB.ConnectDB.getConnectDB();
                PreparedStatement pst = conn.prepareStatement(sql);
                ResultSet rs = pst.executeQuery();

                while (rs.next()) {
                    String medicineName = rs.getString("MedicineName").replaceAll("\\s+", ""); // Loại bỏ khoảng trắng trong dữ liệu
                    String barCode = rs.getString("Barcode").replaceAll("\\s+", "");
                    if (medicineName.toLowerCase().contains(searchItem.toLowerCase()) || barCode.toLowerCase().contains(searchItem.toLowerCase())) {
                        TableProduct data = new TableProduct(rs.getString("MedicineID"),
                                rs.getString("MedicineName"),
                                rs.getString("Barcode"),
                                rs.getString("BatchNo"),
                                rs.getDate("ExpireDate"),
                                rs.getString("Item"),
                                rs.getInt("SellingPrice"),
                                rs.getInt("Quantity1"));
                        productList.add(data);
                    }
                }
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }

        tableProduct.setItems(productList);
    }

    private void calculateTotalPrice() {
        double totalPrice = 0;

        for (Node node : vbox.getChildren()) {
            if (node instanceof AnchorPane) {
                AnchorPane anchorPane = (AnchorPane) node;
                for (Node child : anchorPane.getChildren()) {
                    if (child instanceof Text && ((Text) child).getText().contains("đ")) {
                        String priceText = ((Text) child).getText().replace("đ", "").trim();
                        try {
                            double price = Double.parseDouble(priceText);
                            totalPrice += price;
                        } catch (NumberFormatException e) {
                            // Ignore invalid price text
                        }
                    }
                }
            }
        }

        TotalPrice.setText(totalPrice + "đ");
    }
```



```java
private void updateQuantityInDatabase(String itemName, int quantityToDeduct, String unit) {
        String sql = "";
        String quantityColumn = "";

        if (unit.equalsIgnoreCase("Pill")) {
            quantityColumn = "Quantity3";
        } else if (unit.equalsIgnoreCase("Blister")) {
            quantityColumn = "Quantity2";
        } else if (unit.equalsIgnoreCase("Box")) {
            quantityColumn = "Quantity1";
        } else {
            // Handle default case if needed
        }

        if (!quantityColumn.isEmpty()) {
            sql = "UPDATE pd SET " + quantityColumn + " = " + quantityColumn + " - ? "
                    + "FROM tblProductDetails pd "
                    + "JOIN tblProduct p ON pd.MedicineID = p.MedicineID "
                    + "WHERE p.MedicineName = ? AND " + quantityColumn + " >= ?";
        } else {
            // Handle case where quantityColumn is not set
            System.out.println("Không xác định được cột số lượng phù hợp cho đơn vị " + unit);
            return;
        }

        try {

            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement pst = conn.prepareStatement(sql);

            // Thiết lập các tham số trong câu lệnh SQL
            pst.setInt(1, quantityToDeduct);
            pst.setString(2, itemName);
            pst.setInt(3, quantityToDeduct); // Ensure enough quantity to deduct

            int updatedRows = pst.executeUpdate();
            if (updatedRows > 0) {
                System.out.println("Cập nhật số lượng thành công.");
                showdata(); // Refresh the table data
            } else {
                System.out.println("Không thành công khi cập nhật số lượng sản phẩm trong cơ sở dữ liệu.");
            }
            pst.close();
            conn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        showdata();

        tableProduct.setOnMouseClicked(this::Table); // Ensure the event is set here
    }

    @FXML
    void cancle_all(ActionEvent event) {
        vbox.getChildren().clear();
        vbox.setDisable(false);
        selectedProducts.clear();
        // Gọi phương thức tính tổng giá sau khi xóa hết anchorPane
        calculateTotalPrice();
    }

    @FXML
    private void handlePayment() {
        AlertMessage alert = new AlertMessage();

        String phone = OrderSearchCus.getText().trim();
        TableCustomer customer = null;
        int pointsToAdd = 0;

// Kiểm tra nếu số điện thoại không rỗng thì mới tiến hành tìm kiếm khách hàng
        if (!phone.isEmpty()) {
            customerController = new CustomerController();
            // Gọi phương thức findCustomerByPhone của customerController để tìm khách hàng
            customer = customerController.findCustomerByPhone(phone);
            if (customer != null) {
                // Tính tổng thanh toán từ các sản phẩm
                int totalPayment = calculateTotalPayment();
                // Tính điểm tích lũy dựa trên tổng thanh toán
                pointsToAdd = totalPayment / 10000; // Mỗi 10,000 VND tương ứng với 1 điểm
                // Cập nhật điểm tích lũy cho khách hàng
                customerController.updateCustomerPoints(customer.getPhone(), pointsToAdd);
            } else {
                // Hiển thị thông báo lỗi nếu không tìm thấy khách hàng với số điện thoại nhập vào
                alert.errorMessage("Không tìm thấy khách hàng với số điện thoại: " + phone);
                return; // Thoát khỏi phương thức nếu không tìm thấy khách hàng
            }
        }
        int totalPayment = calculateTotalPayment();
        String billID = generateBillID();
        String paymentMethod = PaymentCash.isSelected() ? "Cash" : "Card";
        boolean billInserted = false;
        String CName = "";
        String CPhone = "";
        String cid = "";
        if (customer != null) {
            CName = customer.getName();
            CPhone = customer.getPhone();
            cid = customer.getId();
        }

```

```java
// Thêm hóa đơn vào cơ sở dữ liệu
    billInserted = insertIntoBills(billID, CName, CPhone, totalPayment, paymentMethod, new Date(System.currentTimeMillis()), cid);

    // Kiểm tra kết quả và xử lý khi không thêm được hóa đơn
    if (!billInserted) {
        // Xử lý khi không thêm được hóa đơn vào cơ sở dữ liệu
        System.out.println("Failed to insert bill into database.");
        // Hoặc có thể hiển thị thông báo lỗi cho người dùng
    } else {
        // Xử lý khi thêm hóa đơn thành công (ví dụ: hiển thị thông báo thành công)
        System.out.println("Bill inserted successfully.");
    }
    for (Node node : vbox.getChildren()) {
        if (node instanceof AnchorPane) {
            AnchorPane anchorPane = (AnchorPane) node;

            // Lấy thông tin sản phẩm từ AnchorPane
            Text itemNameText = (Text) anchorPane.lookup("#itemNameText");
            TextField quantityField = (TextField) anchorPane.lookup("#quantityField");
            Text unitPriceText = (Text) anchorPane.lookup("#unitPriceText");
            Text itemTypeText = (Text) anchorPane.lookup("#itemTypeText");
            Text medicineIdText = (Text) anchorPane.lookup("#medicineID");

            if (itemNameText != null && quantityField != null && unitPriceText != null && itemTypeText != null && medicineIdText != null) {
                String itemName = itemNameText.getText().trim();
                int quantity = 0;
                try {
                    quantity = Integer.parseInt(quantityField.getText().trim());
                } catch (NumberFormatException e) {
                    e.printStackTrace();
                    continue; // Bỏ qua nếu không thể chuyển đổi thành số nguyên
                }

                int unitPrice = 0;
                try {
                    String priceText = unitPriceText.getText().replace("đ", "").trim();
                    double unitPriceDouble = Double.parseDouble(priceText);
                    unitPrice = (int) unitPriceDouble;
                } catch (NumberFormatException e) {
                    e.printStackTrace();
                    continue;
                }

                String itemType = itemTypeText.getText().trim();
                String productId = medicineIdText.getText().trim();
                // Thêm vào bảng BillItems
                boolean itemInserted = insertIntoBillItems(billID, itemName, quantity, unitPrice, itemType, productId);
                if (!itemInserted) {
                    alert.errorMessage("Lỗi khi lưu thông tin chi tiết hóa đơn vào cơ sở dữ liệu.");
                    return;
                }
            }
        }
    }
```

10 Expiry

```java
public class ProductExpiryInfoController {
    @FXML
    private Button exit;
    @FXML
    private TableColumn<ProductExpiryTable, Long> daysLeftColumn;

    @FXML
    private TableColumn<ProductExpiryTable, LocalDate> expireDateColumn;

    @FXML
    private TableView<ProductExpiryTable> expiringProductsTable;

    @FXML
    private TableColumn<ProductExpiryTable, LocalDate> manufacturingDateColumn;

    @FXML
    private TableColumn<ProductExpiryTable, String> medicineIDColumn;

    @FXML
    private TableColumn<ProductExpiryTable, String> medicineNameColumn;

    private List<ProductExpiryTable> expiringProducts;

    public void initialize() {
        // Initialize table columns
        medicineIDColumn.setCellValueFactory(new PropertyValueFactory<>("medicineID"));
        medicineNameColumn.setCellValueFactory(new PropertyValueFactory<>("medicineName"));
        manufacturingDateColumn.setCellValueFactory(new PropertyValueFactory<>("manufacturingDate"));
        expireDateColumn.setCellValueFactory(new PropertyValueFactory<>("expireDate"));
        daysLeftColumn.setCellValueFactory(new PropertyValueFactory<>("daysLeft"));

        exit.setOnMouseClicked(event -> {
            Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
            stage.close();
        });
    }

    public void setExpiringProducts(List<ProductExpiryTable> products) {
        expiringProductsTable.setItems(FXCollections.observableArrayList(products));
    }

}

```

11 Update Customer

```java
public class UpdateCustomerController implements Initializable {

    @FXML
    private ImageView exit;
    @FXML
    private TextField txtName;

    @FXML
    private TextField txtAddress;

    @FXML
    private TextField txtPhone;

    @FXML
    private TextField txtEmail;

    @FXML
    private RadioButton radioMale;

    @FXML
    private RadioButton radioFemale;

    @FXML
    private TextArea txtpoint;
    @FXML
    private TableView<TableCustomer> tableCustomer;
    public TableCustomer customer;
    public CustomerUpdateListener listener;

    public void setCustomerUpdateListener(CustomerUpdateListener listener) {
        this.listener = listener;
    }


    public void setCustomer(TableCustomer customer) {
        this.customer = customer;

        txtName.setText(customer.getName());
        txtAddress.setText(customer.getAddress());
        txtPhone.setText(customer.getPhone());
        txtEmail.setText(customer.getEmail());
        if ("Male".equalsIgnoreCase(customer.getGender())) {
            radioMale.setSelected(true);
        } else if ("Female".equalsIgnoreCase(customer.getGender())) {
            radioFemale.setSelected(true);
        }
        txtpoint.setText(customer.getPoint());
    }

    @FXML
    public void handleUpdate() {
        AlertMessage alert = new AlertMessage();

        // Kiểm tra dữ liệu nhập vào
        if (txtName.getText().isEmpty() || txtAddress.getText().isEmpty() || txtPhone.getText().isEmpty() || txtEmail.getText().isEmpty()
                || txtpoint.getText().isEmpty()) {
            alert.errorMessage("Please enter complete information");
            return;
        }

        // Kiểm tra xem 'Nam' hoặc 'Nữ' đã được chọn chưa
        if (!(radioMale.isSelected() ^ radioFemale.isSelected())) {
            alert.errorMessage("Please select 'Male' or 'Female'");
            return;
        }

        // Kiểm tra định dạng email có hợp lệ không
        if (!txtEmail.getText().matches("^[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$")) {
            alert.errorMessage("Please enter the correct Email");
            return;
        }

```

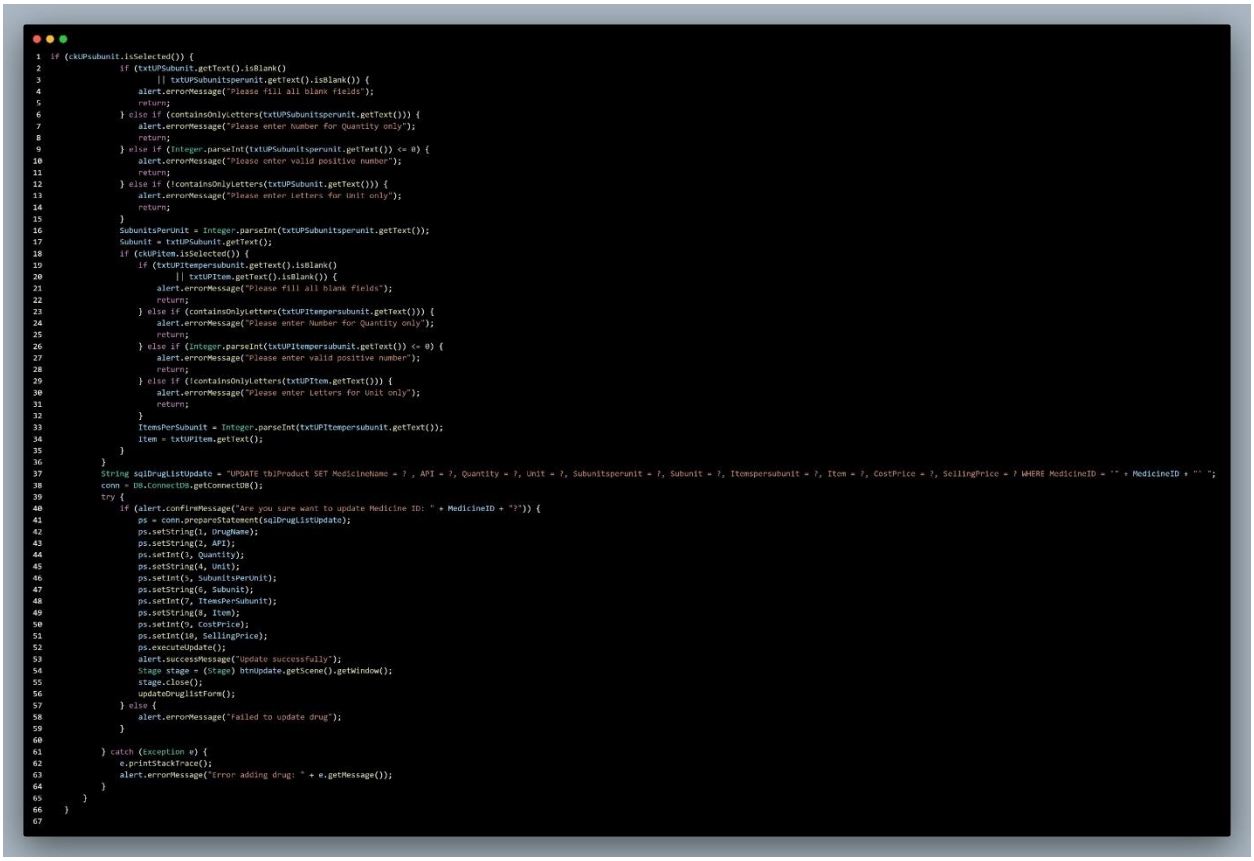
```java
// Kiểm tra số điện thoại trùng
        String checkPhoneQuery = "SELECT * FROM Customer WHERE Phone = ? AND ID != ?";
        try {

            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement checkPhoneStmt = conn.prepareStatement(checkPhoneQuery);
            checkPhoneStmt.setString(1, txtPhone.getText());
            checkPhoneStmt.setString(2, customer.getId()); // customer.getId() là phương thức lấy ID của khách hàng
            ResultSet existingPhone = checkPhoneStmt.executeQuery();
            if (existingPhone.next()) {
                alert.errorMessage("Phone number already exists");
                return; // Thoát phương thức nếu số điện thoại đã tồn tại
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }

        // Kiểm tra email trùng
        String checkEmailQuery = "SELECT * FROM Customer WHERE Email = ? AND ID != ?";
        try {

            Connection conn = DB.ConnectDB.getConnectDB();
            PreparedStatement checkEmailStmt = conn.prepareStatement(checkEmailQuery);
            checkEmailStmt.setString(1, txtEmail.getText());
            checkEmailStmt.setString(2, customer.getId()); // customer.getId() là phương thức lấy ID của khách hàng
            ResultSet existingEmail = checkEmailStmt.executeQuery();
            if (existingEmail.next()) {
                alert.errorMessage("Email address already exists");
                return; // Thoát phương thức nếu địa chỉ email đã tồn tại
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }

        // Cập nhật thông tin khách hàng
        customer.setName(txtName.getText());
        customer.setAddress(txtAddress.getText());
        customer.setPhone(txtPhone.getText());
        customer.setEmail(txtEmail.getText());
        customer.setGender(radioMale.isSelected() ? "Nam" : "Nữ");
        customer.setPoint(txtpoint.getText());

        // Gọi phương thức onUpdate của listener để thông báo cập nhật
        if (listener != null) {
            listener.onUpdate(customer);
        }

        // Hiển thị thông báo cập nhật thành công và đóng cửa sổ
        alert.successMessage("Successfully update Customers");
        Stage stage = (Stage) txtName.getScene().getWindow();
        stage.close();
    }

    @FXML
    public void handleCancel() {
        Stage stage = (Stage) txtName.getScene().getWindow();
        stage.close();
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        exit.setOnMouseClicked(event -> {
            Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
            stage.close();
        });
    }
}

```

12 Update Drug

```java
public class UpdateDrugController implements Initializable {

    @FXML
    private Button btnUpdateClose;

    @FXML
    private Button btnCancelUpdate;

    @FXML
    private Button btnUpdate;

    @FXML
    private CheckBox ckUPitem;

    @FXML
    private CheckBox ckUPsubunit;

    @FXML
    private TextField txtUPAPI;

    @FXML
    private TextField txtUPCostprice;

    @FXML
    private TextField txtUPDrugName;

    @FXML
    private TextField txtUPItem;

    @FXML
    private TextField txtUPItempersubunit;

    @FXML
    private TextField txtUPQuantity;

    @FXML
    private TextField txtUPSellingprice;

    @FXML
    private TextField txtUPSubunit;

    @FXML
    private TextField txtUPSubunitsperunit;

    @FXML
    private TextField txtUPUnit;

    @FXML
    void Cancelbtn(ActionEvent event) {
        updateDruglistForm();
        Stage stage = (Stage) btnCancelUpdate.getScene().getWindow();
        stage.close();
    }

    @FXML
    void Closebtn(ActionEvent event) {
        updateDruglistForm();
        Stage stage = (Stage) btnUpdateClose.getScene().getWindow();
        stage.close();
    }
```

12.1 Code

```
private String MedicineID;

    Message alert = new Message();
    private Connection conn;
    private PreparedStatement ps;
    private ResultSet rs;

    private boolean isPositiveNumber(String value) {
        try {
            int intValue = Integer.parseInt(value);
            return intValue >= 0;
        } catch (Exception e) {
            return false;
        }
    }

    private boolean containsOnlyLetters(String text) {
        return text.matches("[a-zA-Z\\s]+");

    }

    private void updateDruglistForm() {

        DruglistController druglistController = ControllerHolder.getInstance().getDruglistController();
        if (druglistController != null) {
            druglistController.showData();
            druglistController.Pagination();
        }
    }

    public boolean isDrugNameExists(String drugName, String medicineID) {
        try {
            String sqlCheck = "SELECT COUNT(*) FROM tblProduct WHERE MedicineName = ? AND MedicineID <> ?";
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sqlCheck);
            ps.setString(1, drugName);
            ps.setString(2, medicineID);
            rs = ps.executeQuery();
            if (rs.next()) {
                int count = rs.getInt(1);
                return count > 0;
            }
        } catch (Exception e) {
            e.printStackTrace();
            alert.errorMessage("Error checking drug name: " + e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        return false;
    }

    public void updatebtn() {
        if (txtUPDrugName.getText().isEmpty()
                || txtUPAPI.getText().isEmpty()
                || txtUPAPI.getText().isEmpty()
                || txtUPCostprice.getText().isEmpty()
                || txtUPSellingprice.getText().isEmpty()
                || txtUPQuantity.getText().isEmpty()
                || txtUPUnit.getText().isEmpty()) {
            alert.errorMessage("Please fill all blank fields");
        } else if (containsOnlyLetters(txtUPCostprice.getText())
                || containsOnlyLetters(txtUPSellingprice.getText())
                || containsOnlyLetters(txtUPQuantity.getText())) {
            alert.errorMessage("Please enter Number for price only");
        } else if (containsOnlyLetters(txtUPQuantity.getText())) {
            alert.errorMessage("Please enter Number for Quantity only");
        } else if (!isPositiveNumber(txtUPCostprice.getText())
                || !isPositiveNumber(txtUPSellingprice.getText())
                || Integer.parseInt(txtUPQuantity.getText()) <= 0) {
            alert.errorMessage("Please enter valid positive number");
        } else if (!containsOnlyLetters(txtUPUnit.getText())) {
            System.out.println(txtUPUnit.getText());
            alert.errorMessage("Please enter Letters for Unit 1 only");
        } else {

            String DrugName = txtUPDrugName.getText();
            if (isDrugNameExists(DrugName, MedicineID)) {
                alert.errorMessage("Drug name already exists. Please enter a different name.");
                return;
            }
            String API = txtUPAPI.getText();
            int CostPrice = Integer.parseInt(txtUPCostprice.getText());
            int SellingPrice = Integer.parseInt(txtUPSellingprice.getText());
            int Quantity = Integer.parseInt(txtUPQuantity.getText());
            String Unit = txtUPUnit.getText();
            int SubunitsPerUnit = 0;
            String Subunit = "";
            int ItemsPerSubunit = 0;
            String Item = "";

```



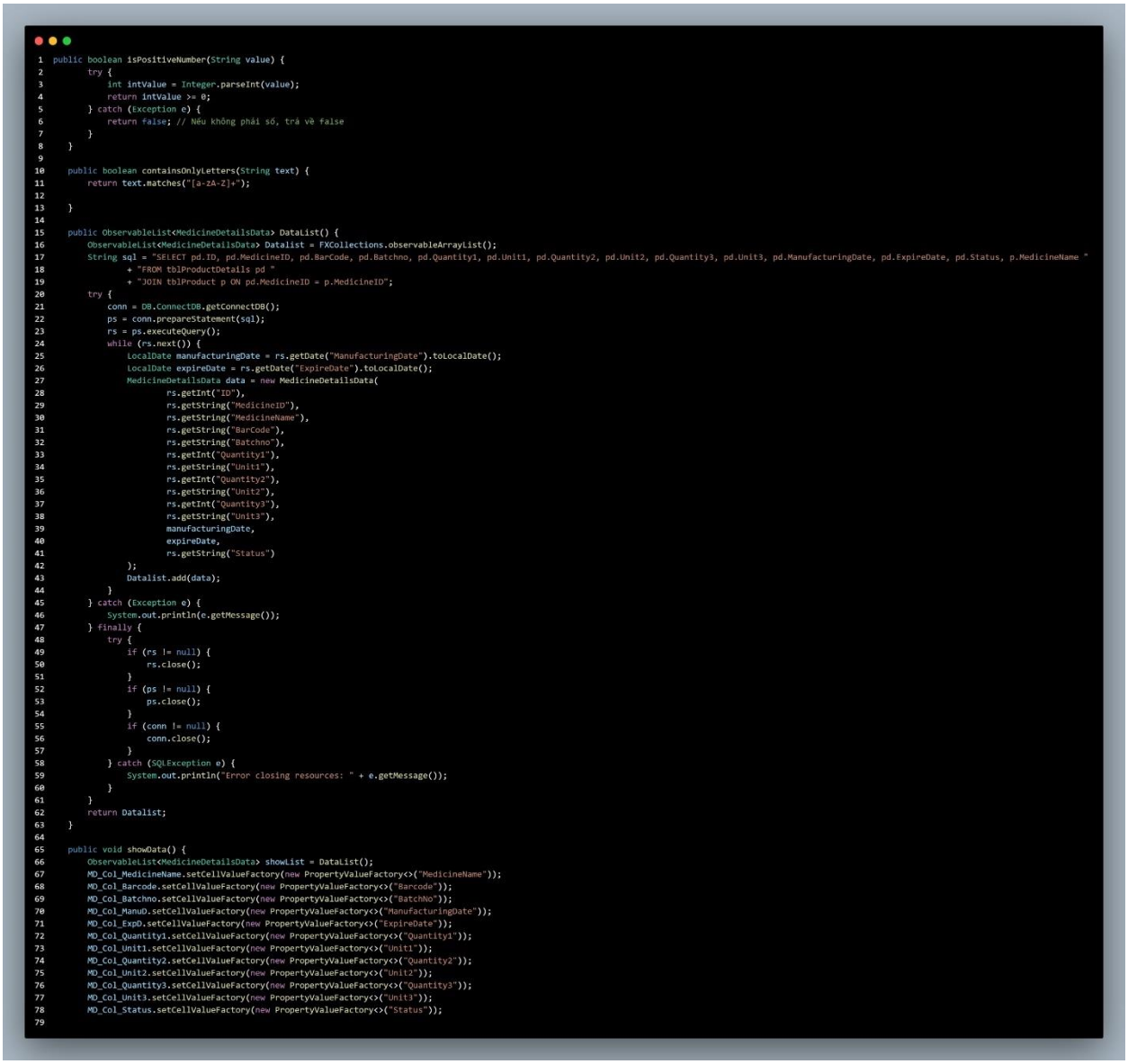
```
if (ckUPsubunit.isSelected()) {
            if (txtUPSubunit.getText().isBlank()
                    || txtUPSubunitsperunit.getText().isBlank()) {
                alert.errorMessage("Please fill all blank fields");
                return;
            } else if (containsOnlyLetters(txtUPSubunitsperunit.getText())) {
                alert.errorMessage("Please enter Number for Quantity only");
                return;
            } else if (Integer.parseInt(txtUPSubunitsperunit.getText()) <= 0) {
                alert.errorMessage("Please enter valid positive number");
                return;
            } else if (!containsOnlyLetters(txtUPSubunit.getText())) {
                alert.errorMessage("Please enter Letters for Unit only");
                return;
            }
            SubunitsPerUnit = Integer.parseInt(txtUPSubunitsperunit.getText());
            Subunit = txtUPSubunit.getText();
            if (ckUPitem.isSelected()) {
                if (txtUPItempersubunit.getText().isBlank()
                        || txtUPItem.getText().isBlank()) {
                    alert.errorMessage("Please fill all blank fields");
                    return;
                } else if (containsOnlyLetters(txtUPItempersubunit.getText())) {
                    alert.errorMessage("Please enter Number for Quantity only");
                    return;
                } else if (Integer.parseInt(txtUPItempersubunit.getText()) <= 0) {
                    alert.errorMessage("Please enter valid positive number");
                    return;
                } else if (!containsOnlyLetters(txtUPItem.getText())) {
                    alert.errorMessage("Please enter Letters for Unit only");
                    return;
                }
                ItemsPerSubunit = Integer.parseInt(txtUPItempersubunit.getText());
                Item = txtUPItem.getText();
            }
        }
        String sqlDrugListUpdate = "UPDATE tblProduct SET MedicineName = ? , API = ?, Quantity = ?, Unit = ?, Subunitsperunit = ?, Subunit = ?, Itemspersubunit = ?, Item = ?, CostPrice = ?, SellingPrice = ? WHERE MedicineID = '" + MedicineID + "' ";
        conn = DB.ConnectDB.getConnectDB();
        try {
            if (alert.confirmMessage("Are you sure want to update Medicine ID: " + MedicineID + "?")) {
                ps = conn.prepareStatement(sqlDrugListUpdate);
                ps.setString(1, DrugName);
                ps.setString(2, API);
                ps.setInt(3, Quantity);
                ps.setString(4, Unit);
                ps.setInt(5, SubunitsPerUnit);
                ps.setString(6, Subunit);
                ps.setInt(7, ItemsPerSubunit);
                ps.setString(8, Item);
                ps.setInt(9, CostPrice);
                ps.setInt(10, SellingPrice);
                ps.executeUpdate();
                alert.successMessage("Update successfully");
                Stage stage = (Stage) btnUpdate.getScene().getWindow();
                stage.close();
                updateDruglistForm();
            } else {
                alert.errorMessage("Failed to update drug");
            }

        } catch (Exception e) {
            e.printStackTrace();
            alert.errorMessage("Error adding drug: " + e.getMessage());
        }
    }
}
```

```java
public void setField() {
        MedicineID = Data.dl_MedicineID;
        txtUPDrugName.setText(Data.dl_MedicineName);
        txtUPAPI.setText(Data.dl_API);
        txtUPCostprice.setText(String.valueOf(Data.dl_Costprice));
        txtUPSellingprice.setText(String.valueOf(Data.dl_Sellingprice));
        txtUPQuantity.setText(String.valueOf(Data.dl_Quantity));
        txtUPUnit.setText(Data.dl_Unit);
        txtUPSubunitsperunit.setText(String.valueOf(Data.dl_Subunitsperunit));
        txtUPSubunit.setText(Data.dl_Subunit);
        txtUPItempersubunit.setText(String.valueOf(Data.dl_Itemspersubunit));
        txtUPItem.setText(Data.dl_Item);
        if (Data.dl_Subunitsperunit == 0) {
            txtUPSubunitsperunit.setDisable(true);
            txtUPSubunit.setDisable(true);

        } else {
            txtUPSubunitsperunit.setDisable(false);
            txtUPSubunit.setDisable(false);
            ckUPsubunit.setSelected(true);

        }
        if (Data.dl_Itemspersubunit == 0) {
            txtUPItempersubunit.setDisable(true);
            txtUPItem.setDisable(true);
        } else {
            txtUPItempersubunit.setDisable(false);
            txtUPItem.setDisable(false);
            ckUPitem.setSelected(true);
            ckUPitem.setDisable(false);
        }

        ckUPsubunit.selectedProperty().addListener((obs, wasSelected, isNowSelected) -> handleSubunitCheck(isNowSelected));
        ckUPitem.selectedProperty().addListener((obs, wasSelected, isNowSelected) -> handleItemCheck(isNowSelected));
    }

    private void handleSubunitCheck(boolean isChecked) {
        if (isChecked) {
            txtUPSubunitsperunit.setDisable(false);
            txtUPSubunit.setDisable(false);
            ckUPitem.setDisable(false);
        } else {
            txtUPSubunitsperunit.setDisable(true);
            txtUPSubunit.setDisable(true);
            txtUPSubunitsperunit.clear();
            txtUPSubunit.clear();
            ckUPitem.setSelected(false);
            ckUPitem.setDisable(true);
            txtUPItempersubunit.setDisable(true);
            txtUPItem.setDisable(true);
            txtUPItempersubunit.clear();
            txtUPItem.clear();
        }
    }

    private void handleItemCheck(boolean isChecked) {
        if (isChecked) {
            txtUPItempersubunit.setDisable(false);
            txtUPItem.setDisable(false);
        } else {
            txtUPItempersubunit.setDisable(true);
            txtUPItem.setDisable(true);
            txtUPItempersubunit.clear();
            txtUPItem.clear();
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        setField();
    }

}

```

13 mport Medicine Details

```java
public class ImportMedicineDetailsController implements Initializable {

    @FXML
    private TableColumn<MedicineDetailsData, Void> MD_Col_AC;

    @FXML
    private TableColumn<MedicineDetailsData, String> MD_Col_Barcode;

    @FXML
    private TableColumn<MedicineDetailsData, String> MD_Col_Batchno;

    @FXML
    private TableColumn<MedicineDetailsData, String> MD_Col_MedicineName;

    @FXML
    private TableColumn<MedicineDetailsData, Integer> MD_Col_Quantity1;

    @FXML
    private TableColumn<MedicineDetailsData, Integer> MD_Col_Quantity2;

    @FXML
    private TableColumn<MedicineDetailsData, Integer> MD_Col_Quantity3;

    @FXML
    private TableColumn<MedicineDetailsData, String> MD_Col_Status;

    @FXML
    private TableColumn<MedicineDetailsData, String> MD_Col_Unit1;

    @FXML
    private TableColumn<MedicineDetailsData, String> MD_Col_Unit2;

    @FXML
    private TableColumn<MedicineDetailsData, String> MD_Col_Unit3;

    @FXML
    private TableColumn<MedicineDetailsData, Date> MD_Col_ExpD;

    @FXML
    private TableColumn<MedicineDetailsData, Date> MD_Col_ManuD;

    @FXML
    private Button btnBG;

    @FXML
    private Button btnMDAdd;

    @FXML
    private Button btnMDUpdate;

    @FXML
    private Button btnMDcancel;

    @FXML
    private Button btnTransHistory;

    @FXML
    private Pagination pgtMD;

    @FXML
    private TextField searchMD;

    @FXML
    private TableView<MedicineDetailsData> tablemedicinedeteailslist;

    @FXML
    private TextField txtMDBarcode;

    @FXML
    private TextField txtMDBatchno;

    @FXML
    private DatePicker txtMDExpireDate;

    @FXML
    private DatePicker txtMDManDate;

    @FXML
    private TextField txtMDMedicineName;

    @FXML
    private ComboBox<String> txtMDStatus;

    @FXML
    private TextField txtQuan1;

    @FXML
    private TextField txtQuan2;

    @FXML
    private TextField txtQuan3;

    @FXML
    private TextField txtUnit1;

    @FXML
    private TextField txtUnit2;

    @FXML
    private TextField txtUnit3;

    private Connection conn;
    private PreparedStatement ps;
    private ResultSet rs;
    Message alert = new M
```

13.1 Code



```java
public boolean isPositiveNumber(String value) {
        try {
            int intValue = Integer.parseInt(value);
            return intValue >= 0;
        } catch (Exception e) {
            return false; // Nếu không phải số, trả về false
        }
    }

    public boolean containsOnlyLetters(String text) {
        return text.matches("[a-zA-Z]+");

    }

    public ObservableList<MedicineDetailsData> DataList() {
        ObservableList<MedicineDetailsData> Datalist = FXCollections.observableArrayList();
        String sql = "SELECT pd.ID, pd.MedicineID, pd.BarCode, pd.Batchno, pd.Quantity1, pd.Unit1, pd.Quantity2, pd.Unit2, pd.Quantity3, pd.Unit3, pd.ManufacturingDate, pd.ExpireDate, pd.Status, p.MedicineName "
                + "FROM tblProductDetails pd "
                + "JOIN tblProduct p ON pd.MedicineID = p.MedicineID";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            rs = ps.executeQuery();
            while (rs.next()) {
                LocalDate manufacturingDate = rs.getDate("ManufacturingDate").toLocalDate();
                LocalDate expireDate = rs.getDate("ExpireDate").toLocalDate();
                MedicineDetailsData data = new MedicineDetailsData(
                        rs.getInt("ID"),
                        rs.getString("MedicineID"),
                        rs.getString("MedicineName"),
                        rs.getString("BarCode"),
                        rs.getString("Batchno"),
                        rs.getInt("Quantity1"),
                        rs.getString("Unit1"),
                        rs.getInt("Quantity2"),
                        rs.getString("Unit2"),
                        rs.getInt("Quantity3"),
                        rs.getString("Unit3"),
                        manufacturingDate,
                        expireDate,
                        rs.getString("Status")
                );
                Datalist.add(data);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
        return Datalist;
    }

    public void showData() {
        ObservableList<MedicineDetailsData> showList = DataList();
        MD_Col_MedicineName.setCellValueFactory(new PropertyValueFactory<>("MedicineName"));
        MD_Col_Barcode.setCellValueFactory(new PropertyValueFactory<>("Barcode"));
        MD_Col_Batchno.setCellValueFactory(new PropertyValueFactory<>("BatchNo"));
        MD_Col_ManuD.setCellValueFactory(new PropertyValueFactory<>("ManufacturingDate"));
        MD_Col_ExpD.setCellValueFactory(new PropertyValueFactory<>("ExpireDate"));
        MD_Col_Quantity1.setCellValueFactory(new PropertyValueFactory<>("Quantity1"));
        MD_Col_Unit1.setCellValueFactory(new PropertyValueFactory<>("Unit1"));
        MD_Col_Quantity2.setCellValueFactory(new PropertyValueFactory<>("Quantity2"));
        MD_Col_Unit2.setCellValueFactory(new PropertyValueFactory<>("Unit2"));
        MD_Col_Quantity3.setCellValueFactory(new PropertyValueFactory<>("Quantity3"));
        MD_Col_Unit3.setCellValueFactory(new PropertyValueFactory<>("Unit3"));
        MD_Col_Status.setCellValueFactory(new PropertyValueFactory<>("Status"));

```

```
MD_Col_AC.setCellFactory(new Callback<TableColumn<MedicineDetailsData, Void>, TableCell<MedicineDetailsData, Void>>() {
            @Override
            public TableCell<MedicineDetailsData, Void> call(final TableColumn<MedicineDetailsData, Void> param) {
                final TableCell<MedicineDetailsData, Void> cell = new TableCell<MedicineDetailsData, Void>() {
                    private final Button btnDelete = new Button("Delete");

                    {
                        btnDelete.getStyleClass().add("btn_delete");
                        btnDelete.setOnAction((ActionEvent event) -> {
                            try {
                                MedicineDetailsData pdata = getTableView().getItems().get(getIndex());
                                if (alert.confirmMessage("Are you sure you want to delete ID: " + pdata.getID() + "?")) {
                                    deleteData(pdata.getID());
                                    showData(); // Refresh table
                                    Pagination(); // Refresh Pagination
                                }
                            } catch (Exception e) {
                                e.printStackTrace();
                            }
                        });
                        btnDelete.setAlignment(Pos.CENTER);
                    }

                    @Override
                    public void updateItem(Void item, boolean empty) {
                        super.updateItem(item, empty);
                        if (empty) {
                            setGraphic(null);
                        } else {
                            HBox hbox = new HBox(5);
                            hbox.getChildren().addAll(btnDelete);
                            hbox.setAlignment(Pos.CENTER);
                            setGraphic(hbox);
                        }
                    }
                };
                return cell;

            }
        });

        tablemedicinedeteailslist.setItems(showList);
    }

    public void MedicineDetailsSelect() {
        tablemedicinedeteailslist.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) -> {
            if (newSelection != null) {
                // Lấy dữ liệu từ hàng được chọn và đổ vào các TextField và DatePicker
                MedicineDetailsData selectedData = newSelection;
                txtMDMedicineName.setText(selectedData.getMedicineName());
                txtMDMedicineName.setDisable(true);
                txtMDBarcode.setText(selectedData.getBarcode());
                txtMDBatchno.setText(selectedData.getBatchNo());
                txtMDManDate.setValue(selectedData.getManufacturingDate());
                txtMDExpireDate.setValue(selectedData.getExpireDate());
                txtQuan1.setText(String.valueOf(selectedData.getQuantity1()));
                txtUnit1.setText(selectedData.getUnit1());
                txtQuan2.setText(String.valueOf(selectedData.getQuantity2()));
                txtUnit2.setText(selectedData.getUnit2());
                txtQuan3.setText(String.valueOf(selectedData.getQuantity3()));
                txtUnit3.setText(selectedData.getUnit3());
                txtMDStatus.getSelectionModel().select(selectedData.getStatus());

```

```
        // Disable fields based on txtUnit2 and txtUnit3 conditions
             if (txtUnit2.getText().isEmpty()) {
                 txtQuan2.setDisable(true);
                 txtUnit2.setDisable(true);
                 txtQuan3.setDisable(true);
                 txtUnit3.setDisable(true);
             } else {
                 txtQuan2.setDisable(false);
                 txtUnit2.setDisable(false);
             }

             if (txtUnit3.getText().isEmpty()) {
                 txtQuan3.setDisable(true);
                 txtUnit3.setDisable(true);
             } else {
                 txtQuan3.setDisable(false);
                 txtUnit3.setDisable(false);
             }
         } else {
             // Nếu không có hàng được chọn, xóa trắng các TextField và DatePicker
             clearFields();
         }
     });
 }

 @FXML
 void MDCancel(ActionEvent event) {
     showData();
     Pagination();
     clearFields();
 }

 private void deleteData(int id) {
     String sqlDeleteProductDetails = "DELETE FROM tblProductDetails WHERE ID = ?";

     try {

         conn = DB.ConnectDB.getConnectDB();
         ps = conn.prepareStatement(sqlDeleteProductDetails);
         ps.setInt(1, id);
         ps.executeUpdate();
         ps.close();

         alert.successMessage("Deleted Successfully");
     } catch (SQLException e) {
         e.printStackTrace();
     } finally {
         try {
             if (ps != null) {
                 ps.close();
             }
             if (conn != null) {
                 conn.close();
             }
         } catch (SQLException e) {
             e.printStackTrace();
         }
     }
 }

 private void setupAutoSuggestion() {
     TextFields.bindAutoCompletion(txtMDMedicineName, suggestionRequest -> {
         ObservableList<String> suggestions = FXCollections.observableArrayList();
         String query = suggestionRequest.getUserText();
         if (query.length() > 1) { // Fetch suggestions when input length is greater than 2
             suggestions.addAll(fetchMedicineNames(query));
         }
         return suggestions;
     }).setOnAutoCompleted(event -> {
         String selectedMedicineName = event.getCompletion();
         fetchAndSetMedicineID(selectedMedicineName);
     });
 }

```

```java
private ObservableList<String> fetchMedicineNames(String query) {
        ObservableList<String> medicineNames = FXCollections.observableArrayList();
        String sql = "SELECT MedicineName FROM tblProduct WHERE MedicineName LIKE ?";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            ps.setString(1, "%" + query + "%");
            rs = ps.executeQuery();
            while (rs.next()) {
                medicineNames.add(rs.getString("MedicineName"));
            }
        } catch (SQLException e) {
            System.out.println("Error fetching medicine names: " + e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
        return medicineNames;
    }

    private void fetchAndSetMedicineID(String medicineName) {
        String sql = "SELECT MedicineID, Unit, Subunit, Item FROM tblProduct WHERE MedicineName = ?";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            ps.setString(1, medicineName);
            rs = ps.executeQuery();
            if (rs.next()) {
                String medicineID = rs.getString("MedicineID");
                String unit = rs.getString("Unit");
                String subunit = rs.getString("Subunit");
                String item = rs.getString("Item");
                txtUnit1.setText(unit);

                if (subunit != null && !subunit.isEmpty()) {
                    txtUnit2.setText(subunit);
                    txtQuan2.setDisable(false);
                    txtUnit2.setDisable(false);
                } else {
                    txtQuan2.setDisable(true);
                    txtUnit2.setDisable(true);
                }

                if (item != null && !item.isEmpty()) {
                    txtUnit3.setText(item);
                    txtQuan3.setDisable(false);
                    txtUnit3.setDisable(false);
                } else {
                    txtQuan3.setDisable(true);
                    txtUnit3.setDisable(true);
                }
            }
        } catch (SQLException e) {
            System.out.println("Error fetching medicine ID: " + e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
    }
```

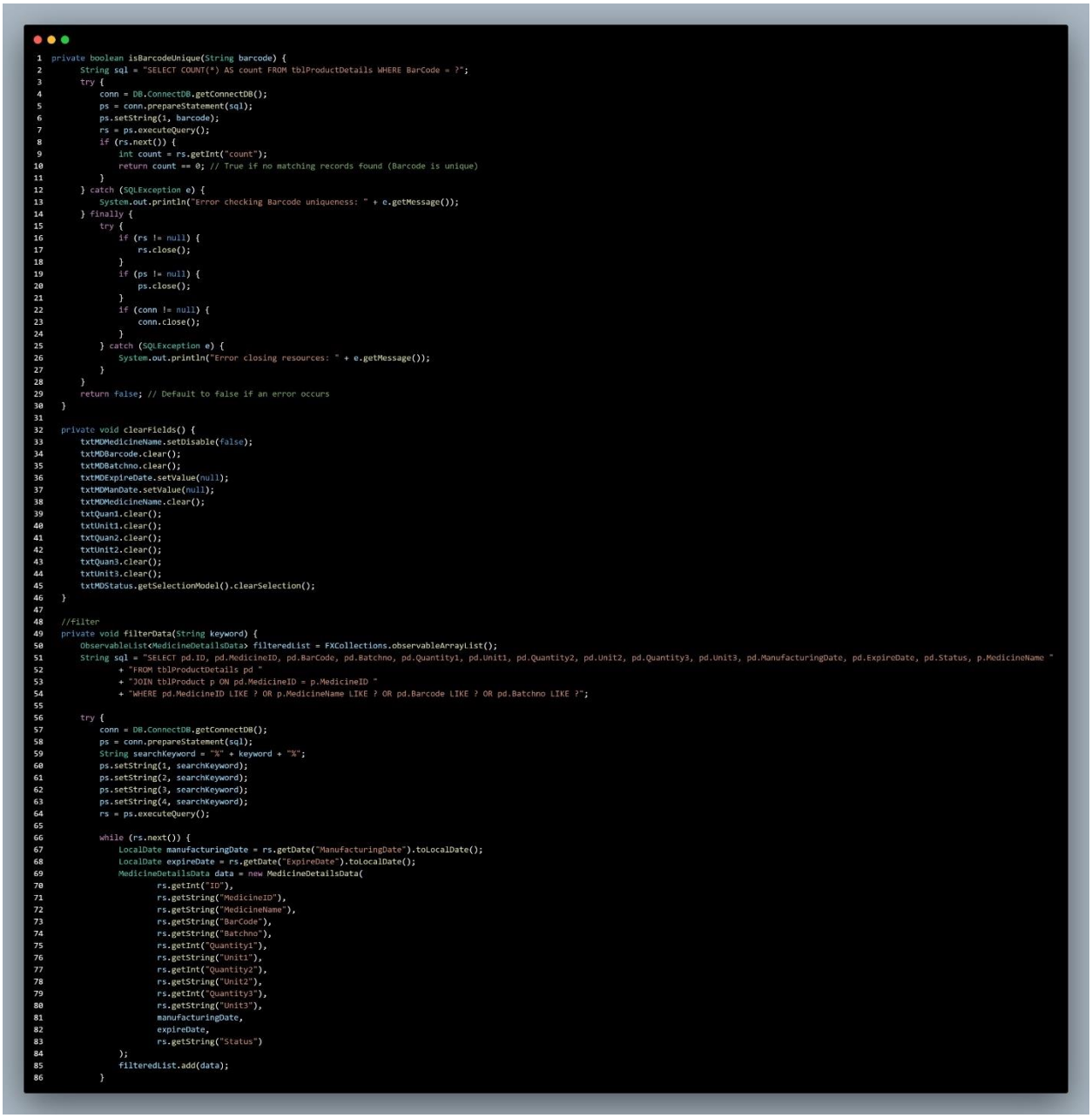
```java
private String fetchAndValidateMedicineID(String medicineName) {
        String sql = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            ps.setString(1, medicineName);
            rs = ps.executeQuery();
            if (rs.next()) {
                return rs.getString("MedicineID");
            }
        } catch (SQLException e) {
            System.out.println("Error fetching medicine ID: " + e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
        return null; // Return null if MedicineID not found
    }

    @FXML
    void AddMD(ActionEvent event) {
        // Fetch or validate MedicineID

        String medicineID = fetchAndValidateMedicineID(txtMDMedicineName.getText());

        if (medicineID == null) {
            // Handle case when MedicineID is not found or invalid
            alert.errorMessage("Invalid Medicine Name. Please select a valid medicine.");
            return;
        }

        if (txtMDMedicineName.getText().isEmpty()
                || txtMDBarcode.getText().isEmpty()
                || txtMDBatchno.getText().isEmpty()
                || txtMDManDate.getValue() == null
                || txtMDExpireDate.getValue() == null
                || txtQuan1.getText().isEmpty()
                || txtUnit1.getText().isEmpty()
                || txtMDStatus.getValue() == null) {
            alert.errorMessage("Please fill all blank fields");
            return;
        } else if (containsOnlyLetters(txtQuan1.getText())) {
            alert.errorMessage("Please enter Number for Quantity only");
            return;
        } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
            alert.errorMessage("Please enter valid positive number");
            return;
        } else if (!containsOnlyLetters(txtUnit1.getText())) {
            alert.errorMessage("Please enter Letters for Unit only");
            return;
        }
        String barcode = txtMDBarcode.getText();
        String batchNo = txtMDBatchno.getText();
        LocalDate manufacturingDate = txtMDManDate.getValue();
        LocalDate expireDate = txtMDExpireDate.getValue();
//        if (!isValidDate(txtMDManDate.getEditor().getText()) || !isValidDate(txtMDExpireDate.getEditor().getText())) {
//            alert.errorMessage("Invalid date format. Please enter dates in the format dd/MM/yyyy.");
//            return;
//        }

        if (manufacturingDate.isAfter(expireDate)) {
            alert.errorMessage("Expiration date must be after manufacturing date.");
            return;
        }

        int quantity1 = Integer.parseInt(txtQuan1.getText());
        String unit1 = txtUnit1.getText();
        int quantity2 = 0;
        String unit2 = "";
        int quantity3 = 0;
        String unit3 = "";
        String status = txtMDStatus.getValue();
        if (!txtUnit2.isDisable()) {
            if (txtUnit2.getText().isBlank()
                    || txtQuan2.getText().isBlank()) {
                alert.errorMessage("Please fill all blank 1 fields");
                return;
            } else if (containsOnlyLetters(txtQuan2.getText())) {
                alert.errorMessage("Please enter Number for Quantity only");
                return;
            } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
                alert.errorMessage("Please enter valid positive number");
                return;
            } else if (!containsOnlyLetters(txtUnit2.getText())) {
                alert.errorMessage("Please enter Letters for Unit only");
                return;
            }
```

```java
private String fetchAndValidateMedicineID(String medicineName) {
        String sql = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            ps.setString(1, medicineName);
            rs = ps.executeQuery();
            if (rs.next()) {
                return rs.getString("MedicineID");
            }
        } catch (SQLException e) {
            System.out.println("Error fetching medicine ID: " + e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
        return null; // Return null if MedicineID not found
    }

    @FXML
    void AddMD(ActionEvent event) {
        // Fetch or validate MedicineID

        String medicineID = fetchAndValidateMedicineID(txtMDMedicineName.getText());

        if (medicineID == null) {
            // Handle case when MedicineID is not found or invalid
            alert.errorMessage("Invalid Medicine Name. Please select a valid medicine.");
            return;
        }

        if (txtMDMedicineName.getText().isEmpty()
                || txtMDBarcode.getText().isEmpty()
                || txtMDBatchno.getText().isEmpty()
                || txtMDManDate.getValue() == null
                || txtMDExpireDate.getValue() == null
                || txtQuan1.getText().isEmpty()
                || txtUnit1.getText().isEmpty()
                || txtMDStatus.getValue() == null) {
            alert.errorMessage("Please fill all blank fields");
            return;
        } else if (containsOnlyLetters(txtQuan1.getText())) {
            alert.errorMessage("Please enter Number for Quantity only");
            return;
        } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
            alert.errorMessage("Please enter valid positive number");
            return;
        } else if (!containsOnlyLetters(txtUnit1.getText())) {
            alert.errorMessage("Please enter Letters for Unit only");
            return;
        }
        String barcode = txtMDBarcode.getText();
        String batchNo = txtMDBatchno.getText();
        LocalDate manufacturingDate = txtMDManDate.getValue();
        LocalDate expireDate = txtMDExpireDate.getValue();
//        if (!isValidDate(txtMDManDate.getEditor().getText()) || !isValidDate(txtMDExpireDate.getEditor().getText())) {
//            alert.errorMessage("Invalid date format. Please enter dates in the format dd/MM/yyyy.");
//            return;
//        }

        if (manufacturingDate.isAfter(expireDate)) {
            alert.errorMessage("Expiration date must be after manufacturing date.");
            return;
        }

        int quantity1 = Integer.parseInt(txtQuan1.getText());
        String unit1 = txtUnit1.getText();
        int quantity2 = 0;
        String unit2 = "";
        int quantity3 = 0;
        String unit3 = "";
        String status = txtMDStatus.getValue();
        if (!txtUnit2.isDisable()) {
            if (txtUnit2.getText().isBlank()
                    || txtQuan2.getText().isBlank()) {
                alert.errorMessage("Please fill all blank 1 fields");
                return;
            } else if (containsOnlyLetters(txtQuan2.getText())) {
                alert.errorMessage("Please enter Number for Quantity only");
                return;
            } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
                alert.errorMessage("Please enter valid positive number");
                return;
            } else if (!containsOnlyLetters(txtUnit2.getText())) {
                alert.errorMessage("Please enter Letters for Unit only");
                return;
            }
```

```
private String fetchAndValidateMedicineID(String medicineName) {
        String sql = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            ps.setString(1, medicineName);
            rs = ps.executeQuery();
            if (rs.next()) {
                return rs.getString("MedicineID");
            }
        } catch (SQLException e) {
            System.out.println("Error fetching medicine ID: " + e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
        return null; // Return null if MedicineID not found
    }

    @FXML
    void AddMD(ActionEvent event) {
        // Fetch or validate MedicineID

        String medicineID = fetchAndValidateMedicineID(txtMDMedicineName.getText());

        if (medicineID == null) {
            // Handle case when MedicineID is not found or invalid
            alert.errorMessage("Invalid Medicine Name. Please select a valid medicine.");
            return;
        }

        if (txtMDMedicineName.getText().isEmpty()
                || txtMDBarcode.getText().isEmpty()
                || txtMDBatchno.getText().isEmpty()
                || txtMDManDate.getValue() == null
                || txtMDExpireDate.getValue() == null
                || txtQuan1.getText().isEmpty()
                || txtUnit1.getText().isEmpty()
                || txtMDStatus.getValue() == null) {
            alert.errorMessage("Please fill all blank fields");
            return;
        } else if (containsOnlyLetters(txtQuan1.getText())) {
            alert.errorMessage("Please enter Number for Quantity only");
            return;
        } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
            alert.errorMessage("Please enter valid positive number");
            return;
        } else if (!containsOnlyLetters(txtUnit1.getText())) {
            alert.errorMessage("Please enter Letters for Unit only");
            return;
        }
        String barcode = txtMDBarcode.getText();
        String batchNo = txtMDBatchno.getText();
        LocalDate manufacturingDate = txtMDManDate.getValue();
        LocalDate expireDate = txtMDExpireDate.getValue();
//        if (!isValidDate(txtMDManDate.getEditor().getText()) || !isValidDate(txtMDExpireDate.getEditor().getText())) {
//            alert.errorMessage("Invalid date format. Please enter dates in the format dd/MM/yyyy.");
//            return;
//        }

        if (manufacturingDate.isAfter(expireDate)) {
            alert.errorMessage("Expiration date must be after manufacturing date.");
            return;
        }

        int quantity1 = Integer.parseInt(txtQuan1.getText());
        String unit1 = txtUnit1.getText();
        int quantity2 = 0;
        String unit2 = "";
        int quantity3 = 0;
        String unit3 = "";
        String status = txtMDStatus.getValue();
        if (!txtUnit2.isDisable()) {
            if (txtUnit2.getText().isBlank()
                    || txtQuan2.getText().isBlank()) {
                alert.errorMessage("Please fill all blank 1 fields");
                return;
            } else if (containsOnlyLetters(txtQuan2.getText())) {
                alert.errorMessage("Please enter Number for Quantity only");
                return;
            } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
                alert.errorMessage("Please enter valid positive number");
                return;
            } else if (!containsOnlyLetters(txtUnit2.getText())) {
                alert.errorMessage("Please enter Letters for Unit only");
                return;
            }
```

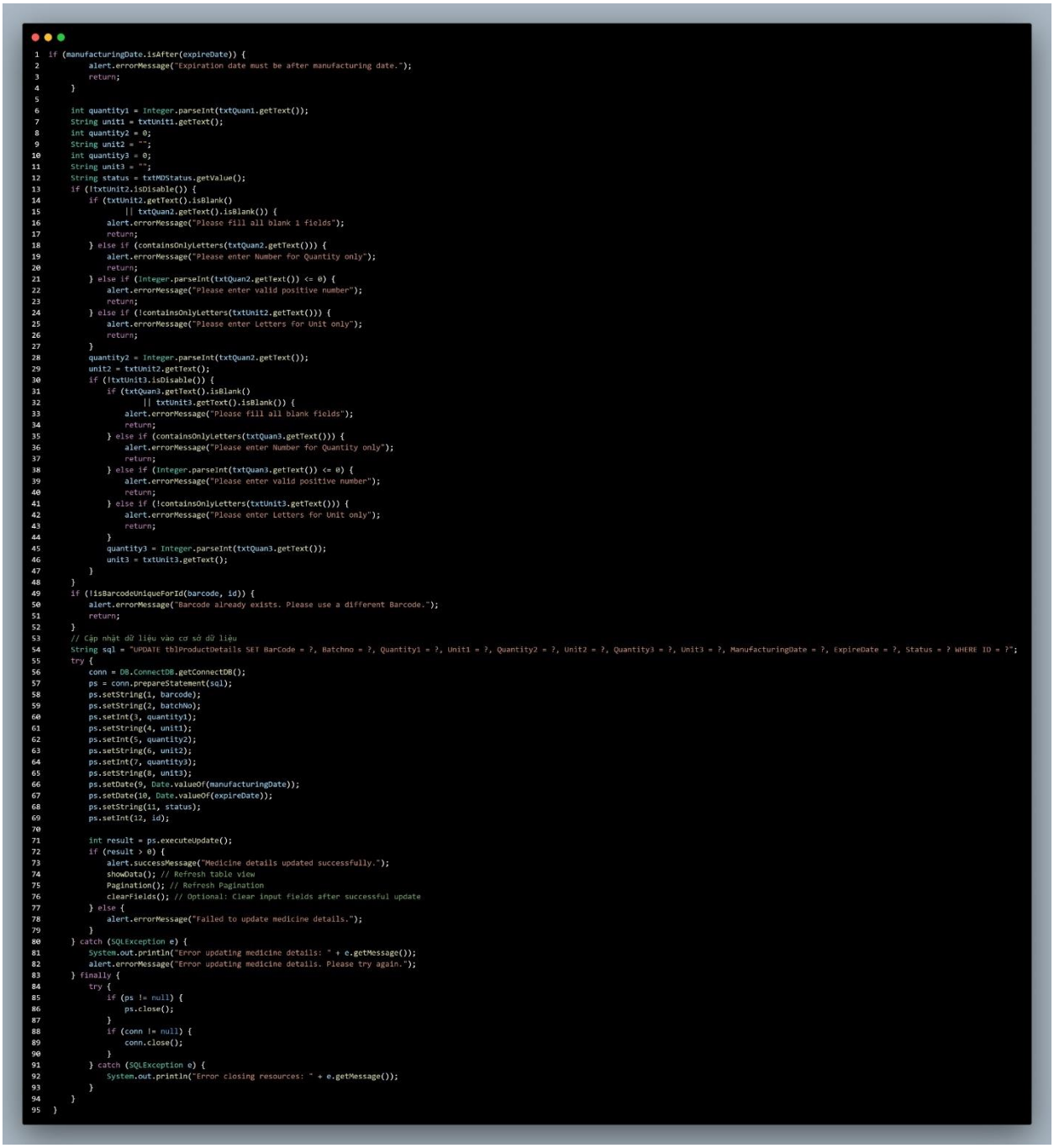
```java
private String fetchAndValidateMedicineID(String medicineName) {
        String sql = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            ps.setString(1, medicineName);
            rs = ps.executeQuery();
            if (rs.next()) {
                return rs.getString("MedicineID");
            }
        } catch (SQLException e) {
            System.out.println("Error fetching medicine ID: " + e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
        return null; // Return null if MedicineID not found
    }

    @FXML
    void AddMD(ActionEvent event) {
        // Fetch or validate MedicineID

        String medicineID = fetchAndValidateMedicineID(txtMDMedicineName.getText());

        if (medicineID == null) {
            // Handle case when MedicineID is not found or invalid
            alert.errorMessage("Invalid Medicine Name. Please select a valid medicine.");
            return;
        }

        if (txtMDMedicineName.getText().isEmpty()
                || txtMDBarcode.getText().isEmpty()
                || txtMDBatchno.getText().isEmpty()
                || txtMDManDate.getValue() == null
                || txtMDExpireDate.getValue() == null
                || txtQuan1.getText().isEmpty()
                || txtUnit1.getText().isEmpty()
                || txtMDStatus.getValue() == null) {
            alert.errorMessage("Please fill all blank fields");
            return;
        } else if (containsOnlyLetters(txtQuan1.getText())) {
            alert.errorMessage("Please enter Number for Quantity only");
            return;
        } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
            alert.errorMessage("Please enter valid positive number");
            return;
        } else if (!containsOnlyLetters(txtUnit1.getText())) {
            alert.errorMessage("Please enter Letters for Unit only");
            return;
        }
        String barcode = txtMDBarcode.getText();
        String batchNo = txtMDBatchno.getText();
        LocalDate manufacturingDate = txtMDManDate.getValue();
        LocalDate expireDate = txtMDExpireDate.getValue();
//        if (!isValidDate(txtMDManDate.getEditor().getText()) || !isValidDate(txtMDExpireDate.getEditor().getText())) {
//            alert.errorMessage("Invalid date format. Please enter dates in the format dd/MM/yyyy.");
//            return;
//        }

        if (manufacturingDate.isAfter(expireDate)) {
            alert.errorMessage("Expiration date must be after manufacturing date.");
            return;
        }

        int quantity1 = Integer.parseInt(txtQuan1.getText());
        String unit1 = txtUnit1.getText();
        int quantity2 = 0;
        String unit2 = "";
        int quantity3 = 0;
        String unit3 = "";
        String status = txtMDStatus.getValue();
        if (!txtUnit2.isDisable()) {
            if (txtUnit2.getText().isBlank()
                    || txtQuan2.getText().isBlank()) {
                alert.errorMessage("Please fill all blank 1 fields");
                return;
            } else if (containsOnlyLetters(txtQuan2.getText())) {
                alert.errorMessage("Please enter Number for Quantity only");
                return;
            } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
                alert.errorMessage("Please enter valid positive number");
                return;
            } else if (!containsOnlyLetters(txtUnit2.getText())) {
                alert.errorMessage("Please enter Letters for Unit only");
                return;
            }
```

```
quantity2 = Integer.parseInt(txtQuan2.getText());
        unit2 = txtUnit2.getText();
        if (!txtUnit3.isDisable()) {
            if (txtQuan3.getText().isBlank()
                    || txtUnit3.getText().isBlank()) {
                alert.errorMessage("Please fill all blank fields");
                return;
            } else if (containsOnlyLetters(txtQuan3.getText())) {
                alert.errorMessage("Please enter Number for Quantity only");
                return;
            } else if (Integer.parseInt(txtQuan3.getText()) <= 0) {
                alert.errorMessage("Please enter valid positive number");
                return;
            } else if (!containsOnlyLetters(txtUnit3.getText())) {
                alert.errorMessage("Please enter Letters for Unit only");
                return;
            }
            quantity3 = Integer.parseInt(txtQuan3.getText());
            unit3 = txtUnit3.getText();
        }
    }
    if (!isBarcodeUnique(barcode)) {
        alert.errorMessage("Barcode already exists. Please use a different Barcode.");
        return;
    }
    // Insert data into database
    String sql = "INSERT INTO tblProductDetails (MedicineID, BarCode, Batchno, Quantity1, Unit1, Quantity2, Unit2, Quantity3, Unit3, ManufacturingDate, ExpireDate, Status) "
            + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
    try {
        conn = DB.ConnectDB.getConnectDB();
        ps = conn.prepareStatement(sql);
        ps.setString(1, medicineID);
        ps.setString(2, barcode);
        ps.setString(3, batchNo);
        ps.setInt(4, quantity1);
        ps.setString(5, unit1);
        ps.setInt(6, quantity2);
        ps.setString(7, unit2);
        ps.setInt(8, quantity3);
        ps.setString(9, unit3);
        ps.setDate(10, Date.valueOf(manufacturingDate));
        ps.setDate(11, Date.valueOf(expireDate));
        ps.setString(12, status);

        int result = ps.executeUpdate();
        if (result > 0) {
            alert.successMessage("Medicine details added successfully.");
            showData(); // Refresh table view
            Pagination(); // Refresh Pagination
            clearFields(); // Optional: Clear input fields after successful addition
        } else {
            alert.errorMessage("Failed to add medicine details.");
        }
    } catch (SQLException e) {
        System.out.println("Error adding medicine details: " + e.getMessage());
        alert.errorMessage("Error adding medicine details. Please try again.");
    } finally {
        try {
            if (ps != null) {
                ps.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            System.out.println("Error closing resources: " + e.getMessage());
        }
    }
}
```



```java
private boolean isBarcodeUnique(String barcode) {
    String sql = "SELECT COUNT(*) AS count FROM tblProductDetails WHERE BarCode = ?";
    try {
        conn = DB.ConnectDB.getConnectDB();
        ps = conn.prepareStatement(sql);
        ps.setString(1, barcode);
        rs = ps.executeQuery();
        if (rs.next()) {
            int count = rs.getInt("count");
            return count == 0; // True if no matching records found (Barcode is unique)
        }
    } catch (SQLException e) {
        System.out.println("Error checking Barcode uniqueness: " + e.getMessage());
    } finally {
        try {
            if (rs != null) {
                rs.close();
            }
            if (ps != null) {
                ps.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            System.out.println("Error closing resources: " + e.getMessage());
        }
    }
    return false; // Default to false if an error occurs
}

private void clearFields() {
    txtMDMedicineName.setDisable(false);
    txtMDBarcode.clear();
    txtMDBatchno.clear();
    txtMDExpireDate.setValue(null);
    txtMDManDate.setValue(null);
    txtMDMedicineName.clear();
    txtQuan1.clear();
    txtUnit1.clear();
    txtQuan2.clear();
    txtUnit2.clear();
    txtQuan3.clear();
    txtUnit3.clear();
    txtMDStatus.getSelectionModel().clearSelection();
}

//filter
private void filterData(String keyword) {
    ObservableList<MedicineDetailsData> filteredList = FXCollections.observableArrayList();
    String sql = "SELECT pd.ID, pd.MedicineID, pd.BarCode, pd.Batchno, pd.Quantity1, pd.Unit1, pd.Quantity2, pd.Unit2, pd.Quantity3, pd.Unit3, pd.ManufacturingDate, pd.ExpireDate, pd.Status, p.MedicineName "
            + "FROM tblProductDetails pd "
            + "JOIN tblProduct p ON pd.MedicineID = p.MedicineID "
            + "WHERE pd.MedicineID LIKE ? OR p.MedicineName LIKE ? OR pd.Barcode LIKE ? OR pd.Batchno LIKE ?";

    try {
        conn = DB.ConnectDB.getConnectDB();
        ps = conn.prepareStatement(sql);
        String searchKeyword = "%" + keyword + "%";
        ps.setString(1, searchKeyword);
        ps.setString(2, searchKeyword);
        ps.setString(3, searchKeyword);
        ps.setString(4, searchKeyword);
        rs = ps.executeQuery();

        while (rs.next()) {
            LocalDate manufacturingDate = rs.getDate("ManufacturingDate").toLocalDate();
            LocalDate expireDate = rs.getDate("ExpireDate").toLocalDate();
            MedicineDetailsData data = new MedicineDetailsData(
                    rs.getInt("ID"),
                    rs.getString("MedicineID"),
                    rs.getString("MedicineName"),
                    rs.getString("BarCode"),
                    rs.getString("Batchno"),
                    rs.getInt("Quantity1"),
                    rs.getString("Unit1"),
                    rs.getInt("Quantity2"),
                    rs.getString("Unit2"),
                    rs.getInt("Quantity3"),
                    rs.getString("Unit3"),
                    manufacturingDate,
                    expireDate,
                    rs.getString("Status")
            );
            filteredList.add(data);
        }
```

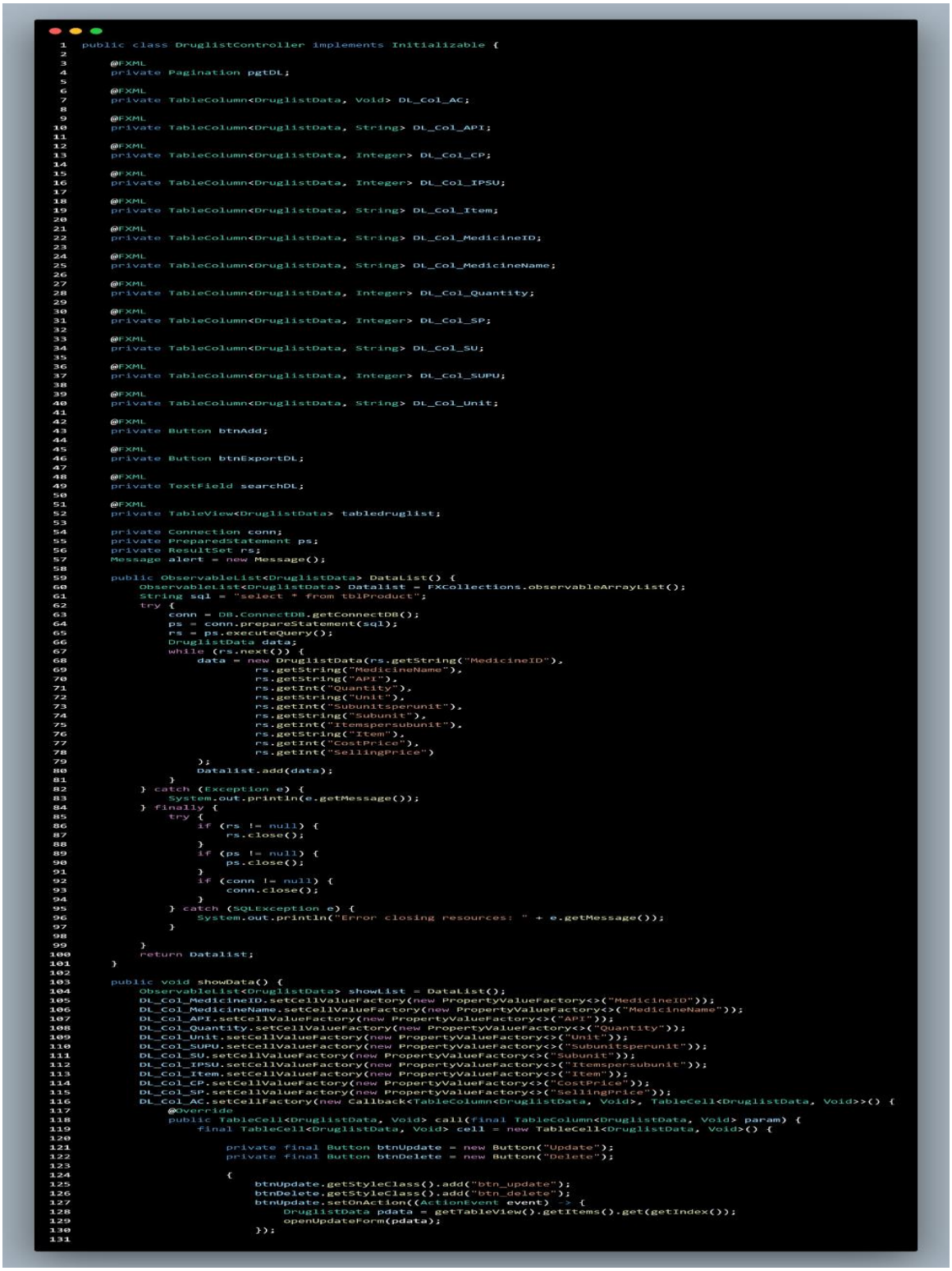
```
} catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        tablemedicinedeteailslist.setItems(filteredList);
    }

    public void Search() {
        searchMD.textProperty().addListener((observable, oldValue, newValue) -> {
            filterData(newValue);
        });
    }

    public void StatusList() {
        List<String> ListS = new ArrayList<>();

        for (String data : Data.Data.status) {
            ListS.add(data);
        }
        ObservableList ListData = FXCollections.observableArrayList(ListS);
        txtMDStatus.setItems(ListData);
    }

    private static final int itemsperpage = 21;

    private int getPageCount() {
        return (int) Math.ceil((double) DataList().size() / itemsperpage);
    }

    private void updateTable(int pageIndex) {
        int fromIndex = pageIndex * itemsperpage;
        int toIndex = Math.min(fromIndex + itemsperpage, DataList().size());
        tablemedicinedeteailslist.setItems(FXCollections.observableArrayList(DataList().subList(fromIndex, toIndex)));
    }

    private Node createPage(int pageIndex) {
        updateTable(pageIndex);
        return tablemedicinedeteailslist;
    }

    public void Pagination() {
        pgtMD.setPageCount(getPageCount());
        pgtMD.setPageFactory(this::createPage);
    }

    @FXML
    void UpdateMD(ActionEvent event) {
        // Kiểm tra nếu người dùng đã chọn một hàng từ bảng
        MedicineDetailsData selectedData = tablemedicinedeteailslist.getSelectionModel().getSelectedItem();
        if (selectedData == null) {
            alert.errorMessage("Please select a medicine detail to update.");
            return;
        }
        if (txtMDMedicineName.getText().isEmpty()
                || txtMDBarcode.getText().isEmpty()
                || txtMDBatchno.getText().isEmpty()
                || txtMDManDate.getValue() == null
                || txtMDExpireDate.getValue() == null
                || txtQuan1.getText().isEmpty()
                || txtUnit1.getText().isEmpty()
                || txtMDStatus.getValue() == null) {
            alert.errorMessage("Please fill all blank fields");
            return;
        } else if (containsOnlyLetters(txtQuan1.getText())) {
            alert.errorMessage("Please enter Number for Quantity only");
            return;
        } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
            alert.errorMessage("Please enter valid positive number");
            return;
        } else if (!containsOnlyLetters(txtUnit1.getText())) {
            alert.errorMessage("Please enter Letters for Unit only");
            return;
        }
        // Lấy dữ liệu từ các trường nhập
        int id = selectedData.getID();
        String barcode = txtMDBarcode.getText();
        String batchNo = txtMDBatchno.getText();
        LocalDate manufacturingDate = txtMDManDate.getValue();
        LocalDate expireDate = txtMDExpireDate.getValue();
        String medicineName = txtMDMedicineName.getText();

```



```
if (manufacturingDate.isAfter(expireDate)) {
        alert.errorMessage("Expiration date must be after manufacturing date.");
        return;
    }

    int quantity1 = Integer.parseInt(txtQuan1.getText());
    String unit1 = txtUnit1.getText();
    int quantity2 = 0;
    String unit2 = "";
    int quantity3 = 0;
    String unit3 = "";
    String status = txtMDStatus.getValue();
    if (!txtUnit2.isDisable()) {
        if (txtUnit2.getText().isBlank()
                || txtQuan2.getText().isBlank()) {
            alert.errorMessage("Please fill all blank 1 fields");
            return;
        } else if (containsOnlyLetters(txtQuan2.getText())) {
            alert.errorMessage("Please enter Number for Quantity only");
            return;
        } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
            alert.errorMessage("Please enter valid positive number");
            return;
        } else if (!containsOnlyLetters(txtUnit2.getText())) {
            alert.errorMessage("Please enter Letters for Unit only");
            return;
        }
        quantity2 = Integer.parseInt(txtQuan2.getText());
        unit2 = txtUnit2.getText();
        if (!txtUnit3.isDisable()) {
            if (txtQuan3.getText().isBlank()
                    || txtUnit3.getText().isBlank()) {
                alert.errorMessage("Please fill all blank fields");
                return;
            } else if (containsOnlyLetters(txtQuan3.getText())) {
                alert.errorMessage("Please enter Number for Quantity only");
                return;
            } else if (Integer.parseInt(txtQuan3.getText()) <= 0) {
                alert.errorMessage("Please enter valid positive number");
                return;
            } else if (!containsOnlyLetters(txtUnit3.getText())) {
                alert.errorMessage("Please enter Letters for Unit only");
                return;
            }
            quantity3 = Integer.parseInt(txtQuan3.getText());
            unit3 = txtUnit3.getText();
        }
    }
    if (!isBarcodeUniqueForId(barcode, id)) {
        alert.errorMessage("Barcode already exists. Please use a different Barcode.");
        return;
    }
    // Cập nhật dữ liệu vào cơ sở dữ liệu
    String sql = "UPDATE tblProductDetails SET BarCode = ?, Batchno = ?, Quantity1 = ?, Unit1 = ?, Quantity2 = ?, Unit2 = ?, Quantity3 = ?, Unit3 = ?, ManufacturingDate = ?, ExpireDate = ?, Status = ? WHERE ID = ?";
    try {
        conn = DB.ConnectDB.getConnectDB();
        ps = conn.prepareStatement(sql);
        ps.setString(1, barcode);
        ps.setString(2, batchNo);
        ps.setInt(3, quantity1);
        ps.setString(4, unit1);
        ps.setInt(5, quantity2);
        ps.setString(6, unit2);
        ps.setInt(7, quantity3);
        ps.setString(8, unit3);
        ps.setDate(9, Date.valueOf(manufacturingDate));
        ps.setDate(10, Date.valueOf(expireDate));
        ps.setString(11, status);
        ps.setInt(12, id);

        int result = ps.executeUpdate();
        if (result > 0) {
            alert.successMessage("Medicine details updated successfully.");
            showData(); // Refresh table view
            Pagination(); // Refresh Pagination
            clearFields(); // Optional: Clear input fields after successful update
        } else {
            alert.errorMessage("Failed to update medicine details.");
        }
    } catch (SQLException e) {
        System.out.println("Error updating medicine details: " + e.getMessage());
        alert.errorMessage("Error updating medicine details. Please try again.");
    } finally {
        try {
            if (ps != null) {
                ps.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            System.out.println("Error closing resources: " + e.getMessage());
        }
    }
}
```

```
    private boolean isBarcodeUniqueForId(String barcode, int id) {
        String sql = "SELECT COUNT(*) AS count FROM tblProductDetails WHERE BarCode = ? AND ID <> ?";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            ps.setString(1, barcode);
            ps.setInt(2, id);
            rs = ps.executeQuery();
            if (rs.next()) {
                int count = rs.getInt("count");
                return count == 0; // True if no matching records found (Barcode is unique for given ID)
            }
        } catch (SQLException e) {
            System.out.println("Error checking Barcode uniqueness for ID: " + e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }
        }
        return false; // Default to false if an error occurs
    }

    @FXML
    private void generateAndSaveBarcode() throws SQLException {
        // Generate a unique numeric barcode
        String barcode = generateBarcode();

        // Check if barcode generation failed
        if (barcode == null) {
            alert.errorMessage("Failed to generate a unique numeric barcode.");
            return;
        }

        // Save generated barcode image to file
        try {
            Code128Bean barcodeGenerator = new Code128Bean();
            final int dpi = 90;
            barcodeGenerator.setModuleWidth(0.6);
            barcodeGenerator.doQuietZone(true);

            BitmapCanvasProvider canvas = new BitmapCanvasProvider(dpi, BufferedImage.TYPE_BYTE_BINARY, false, 0);
            barcodeGenerator.generateBarcode(canvas, barcode);

            FileChooser fileChooser = new FileChooser();
            fileChooser.setTitle("Save Barcode Image");
            FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter("PNG files (*.png)", "*.png");
            fileChooser.getExtensionFilters().add(extFilter);
            File barcodeFile = fileChooser.showSaveDialog(null);

            if (barcodeFile != null) {
                FileOutputStream fos = new FileOutputStream(barcodeFile);
                canvas.finish();
                ImageIO.write(canvas.getBufferedImage(), "png", fos);
                fos.close();

                // Save barcode details to the database
                saveBarcodeDetails(barcode);

                alert.successMessage("Barcode Saved Successfully.");
                Desktop.getDesktop().open(barcodeFile); // Open the saved barcode image
            } else {
                alert.errorMessage("File save cancelled.");
            }
        } catch (IOException e) {
            e.printStackTrace();
            alert.errorMessage("Error generating or saving barcode: " + e.getMessage());
        }
    }

    private String generateBarcode() throws SQLException {
        String barcode;
        do {
            barcode = generateNumericBarcode();
        } while (!BarcodeUnique(barcode));

        return barcode;
    }

    private String generateNumericBarcode() {
        // Generate a random numeric barcode (10 digits)
        Random random = new Random();
        int numDigits = 12;
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < numDigits; i++) {
            sb.append(random.nextInt(10)); // Generate random digit (0-9)
        }
        return sb.toString();
    }

    private boolean BarcodeUnique(String barcode) throws SQLException {
        // Check if the barcode exists in the database
        String sql = "SELECT * FROM tblBarcode WHERE Barcode = ?";
        conn = DB.ConnectDB.getConnectDB();
        ps = conn.prepareStatement(sql);
        ps.setString(1, barcode);
        ResultSet rs = ps.executeQuery();
        boolean unique = !rs.next(); // true if ResultSet is empty (barcode is unique)
        rs.close();
        ps.close();
        conn.close();
        return unique;
    }

    private void saveBarcodeDetails(String barcode) throws SQLException {
        // Save barcode details to the database
        String sql = "INSERT INTO tblBarcode (Barcode) VALUES (?)";
        conn = DB.ConnectDB.getConnectDB();
        ps = conn.prepareStatement(sql);
        ps.setString(1, barcode);
        ps.executeUpdate();
        ps.close();
        conn.close();
    }

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        showData();
        setupAutoSuggestion();
        StatusList();
        MedicineDetailsSelect();
        Search();
        Pagination();
    }
}
```

14 Drug List



```java
public class DruglistController implements Initializable {

    @FXML
    private Pagination pgtDL;

    @FXML
    private TableColumn<DruglistData, Void> DL_Col_AC;

    @FXML
    private TableColumn<DruglistData, String> DL_Col_API;

    @FXML
    private TableColumn<DruglistData, Integer> DL_Col_CP;

    @FXML
    private TableColumn<DruglistData, Integer> DL_Col_IPSU;

    @FXML
    private TableColumn<DruglistData, String> DL_Col_Item;

    @FXML
    private TableColumn<DruglistData, String> DL_Col_MedicineID;

    @FXML
    private TableColumn<DruglistData, String> DL_Col_MedicineName;

    @FXML
    private TableColumn<DruglistData, Integer> DL_Col_Quantity;

    @FXML
    private TableColumn<DruglistData, Integer> DL_Col_SP;

    @FXML
    private TableColumn<DruglistData, String> DL_Col_SU;

    @FXML
    private TableColumn<DruglistData, Integer> DL_Col_SUPU;

    @FXML
    private TableColumn<DruglistData, String> DL_Col_Unit;

    @FXML
    private Button btnAdd;

    @FXML
    private Button btnExportDL;

    @FXML
    private TextField searchDL;

    @FXML
    private TableView<DruglistData> tabledruglist;

    private Connection conn;
    private PreparedStatement ps;
    private ResultSet rs;
    Message alert = new Message();

    public ObservableList<DruglistData> DataList() {
        ObservableList<DruglistData> Datalist = FXCollections.observableArrayList();
        String sql = "select * from tblProduct";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            rs = ps.executeQuery();
            DruglistData data;
            while (rs.next()) {
                data = new DruglistData(rs.getString("MedicineID"),
                        rs.getString("MedicineName"),
                        rs.getString("API"),
                        rs.getInt("Quantity"),
                        rs.getString("Unit"),
                        rs.getInt("Subunitsperunit"),
                        rs.getString("Subunit"),
                        rs.getInt("Itemspersubunit"),
                        rs.getString("Item"),
                        rs.getInt("CostPrice"),
                        rs.getInt("SellingPrice")
                );
                Datalist.add(data);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }

        }
        return Datalist;
    }

    public void showData() {
        ObservableList<DruglistData> showList = DataList();
        DL_Col_MedicineID.setCellValueFactory(new PropertyValueFactory<>("MedicineID"));
        DL_Col_MedicineName.setCellValueFactory(new PropertyValueFactory<>("MedicineName"));
        DL_Col_API.setCellValueFactory(new PropertyValueFactory<>("API"));
        DL_Col_Quantity.setCellValueFactory(new PropertyValueFactory<>("Quantity"));
        DL_Col_Unit.setCellValueFactory(new PropertyValueFactory<>("Unit"));
        DL_Col_SUPU.setCellValueFactory(new PropertyValueFactory<>("Subunitsperunit"));
        DL_Col_SU.setCellValueFactory(new PropertyValueFactory<>("Subunit"));
        DL_Col_IPSU.setCellValueFactory(new PropertyValueFactory<>("Itemspersubunit"));
        DL_Col_Item.setCellValueFactory(new PropertyValueFactory<>("Item"));
        DL_Col_CP.setCellValueFactory(new PropertyValueFactory<>("CostPrice"));
        DL_Col_SP.setCellValueFactory(new PropertyValueFactory<>("SellingPrice"));
        DL_Col_AC.setCellFactory(new Callback<TableColumn<DruglistData, Void>, TableCell<DruglistData, Void>>() {
            @Override
            public TableCell<DruglistData, Void> call(final TableColumn<DruglistData, Void> param) {
                final TableCell<DruglistData, Void> cell = new TableCell<DruglistData, Void>() {

                    private final Button btnUpdate = new Button("Update");
                    private final Button btnDelete = new Button("Delete");

                    {
                        btnUpdate.getStyleClass().add("btn_update");
                        btnDelete.getStyleClass().add("btn_delete");
                        btnUpdate.setOnAction((ActionEvent event) -> {
                            DruglistData pdata = getTableView().getItems().get(getIndex());
                            openUpdateForm(pdata);
                        });

```

14.1 Code

```
                    btnDelete.setOnAction((ActionEvent event) -> {
                        DruglistData pdata = getTableView().getItems().get(getIndex());
                        if (alert.confirmMessage("Are you sure want to delete Medicine ID: " + pdata.getMedicineID() + "?")) {
                            deleteData(pdata.getMedicineID());
                            showData(); // Refresh table
                            Pagination();
                        }
                    });
                    btnUpdate.setAlignment(Pos.CENTER);
                    btnDelete.setAlignment(Pos.CENTER);
                }

                @Override
                public void updateItem(Void item, boolean empty) {
                    super.updateItem(item, empty);
                    if (empty) {
                        setGraphic(null);
                    } else {
                        HBox hbox = new HBox(5);
                        hbox.getChildren().addAll(btnUpdate, btnDelete);
                        hbox.setAlignment(Pos.CENTER);
                        setGraphic(hbox);
                    }
                }
            };
            return cell;
        }
    });

    tabledruglist.setItems(showList);
}

private void deleteData(String medicineID) {
    String sqlDeleteProductDetails = "DELETE FROM tblProductDetails WHERE MedicineID = ?";
    String sqlDeleteProduct = "DELETE FROM tblProduct WHERE MedicineID = ?";
    try {

        conn = DB.ConnectDB.getConnectDB();
        ps = conn.prepareStatement(sqlDeleteProductDetails);
        ps.setString(1, medicineID);
        ps.executeUpdate();
        ps.close();

        ps = conn.prepareStatement(sqlDeleteProduct);
        ps.setString(1, medicineID);
        ps.executeUpdate();
        alert.successMessage("Deleted Successfully");
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        try {
            if (ps != null) {
                ps.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

private void openUpdateForm(DruglistData pdata) {
    try {
        Data.dl_MedicineID = pdata.getMedicineID();
        Data.dl_MedicineName = pdata.getMedicineName();
        Data.dl_API = pdata.getAPI();
        Data.dl_Costprice = pdata.getCostPrice();
        Data.dl_Sellingprice = pdata.getSellingPrice();
        Data.dl_Quantity = pdata.getQuantity();
        Data.dl_Subunitsperunit = pdata.getSubunitsperunit();
        Data.dl_Itemspersubunit = pdata.getItemspersubunit();
        Data.dl_Unit = pdata.getUnit();
        Data.dl_Subunit = pdata.getSubunit();
        Data.dl_Item = pdata.getItem();
        FXMLLoader loader = new FXMLLoader(getClass().getResource("/UpdateDrug/UpdateDrug.fxml"));

        Parent root = loader.load();

        Stage stage = new Stage();
        stage.initStyle(StageStyle.UNDECORATED);
        stage.setScene(new Scene(root));
        stage.show();
    } catch (Exception e) {
        e.printStackTrace();
    }

}

@FXML
void showformAdd(MouseEvent event) {
    try {

        FXMLLoader loader = new FXMLLoader(getClass().getResource("/AddDrug/AddDrug.fxml"));
        Parent root = loader.load();
        AddDrugController addDrugController = ControllerHolder.getInstance().getAddDrugController();
        addDrugController.CheckBoxHandlers();

        Stage stage = new Stage();
        stage.initStyle(StageStyle.UNDECORATED);
        stage.setScene(new Scene(root));
        stage.show();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

```java
private void filterData(String keyword) {
        ObservableList<DruglistData> filteredList = FXCollections.observableArrayList();
        String sql = "SELECT * FROM tblProduct WHERE MedicineID LIKE ? OR MedicineName LIKE ? OR API LIKE ?";

        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            String searchKeyword = "%" + keyword + "%";
            ps.setString(1, searchKeyword);
            ps.setString(2, searchKeyword);
            ps.setString(3, searchKeyword);
            rs = ps.executeQuery();

            while (rs.next()) {
                DruglistData data = new DruglistData(
                        rs.getString("MedicineID"),
                        rs.getString("MedicineName"),
                        rs.getString("API"),
                        rs.getInt("Quantity"),
                        rs.getString("Unit"),
                        rs.getInt("Subunitsperunit"),
                        rs.getString("Subunit"),
                        rs.getInt("Itemspersubunit"),
                        rs.getString("Item"),
                        rs.getInt("CostPrice"),
                        rs.getInt("SellingPrice")
                );
                filteredList.add(data);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        tabledruglist.setItems(filteredList);
    }

    public void Search() {
        searchDL.textProperty().addListener((observable, oldValue, newValue) -> {
            filterData(newValue);
        });
    }
    private static final int itemsperpage = 23;

    private int getPageCount() {
        return (int) Math.ceil((double) DataList().size() / itemsperpage);
    }

    private void updateTable(int pageIndex) {
        int fromIndex = pageIndex * itemsperpage;
        int toIndex = Math.min(fromIndex + itemsperpage, DataList().size());
        tabledruglist.setItems(FXCollections.observableArrayList(DataList().subList(fromIndex, toIndex)));
    }

    private Node createPage(int pageIndex) {
        updateTable(pageIndex);
        return tabledruglist;
    }
```

```java
public void Pagination() {
        pgtDL.setPageCount(getPageCount());
        pgtDL.setPageFactory(this::createPage);
    }

    public void exportToPDF() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save PDF");
        String fontPath = "src/font/arial.ttf";
        fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("PDF Files", "*.pdf"));
        File selectedFile = fileChooser.showSaveDialog(btnExportDL.getScene().getWindow());

        if (selectedFile != null) {
            try {

                PdfFont font = PdfFontFactory.createFont(fontPath, PdfEncodings.IDENTITY_H, true);

                PdfWriter writer = new PdfWriter(selectedFile.getAbsolutePath());
                PdfDocument pdf = new PdfDocument(writer);
                Document document = new Document(pdf);

                Paragraph title = new Paragraph("Drug List").setBold().setFontSize(18);
                document.add(title);

                float[] columnWidths = {100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100};
                Table table = new Table(columnWidths);

                TableView<DruglistData> tableView = tabledruglist;
                for (TableColumn<DruglistData, ?> column : tableView.getColumns()) {
                    if (!column.getText().equals("Action")) {
                        table.addCell(new Cell().add(new Paragraph(column.getText()).setFont(font)));
                    }
                }

                ObservableList<DruglistData> dataList = tableView.getItems();
                for (DruglistData data : dataList) {
                    table.addCell(new Cell().add(new Paragraph(data.getMedicineID())));
                    table.addCell(new Cell().add(new Paragraph(data.getMedicineName())));
                    table.addCell(new Cell().add(new Paragraph(data.getAPI())));
                    table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getQuantity()))));
                    table.addCell(new Cell().add(new Paragraph(data.getUnit())));
                    table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getSubunitsperunit()))));
                    table.addCell(new Cell().add(new Paragraph(data.getSubunit())));
                    table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getItemspersubunit()))));
                    table.addCell(new Cell().add(new Paragraph(data.getItem())));
                    table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getCostPrice()))));
                    table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getSellingPrice()))));
                }
                document.add(table);
                document.close();
                Desktop.getDesktop().open(selectedFile);

                alert.successMessage("Export successful!");
            } catch (IOException e) {
                alert.errorMessage("Export failed: " + e.getMessage());
            }
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        ControllerHolder.getInstance().setDruglistController(this);
        showData();
        Search();
        Pagination();
    }

}
```

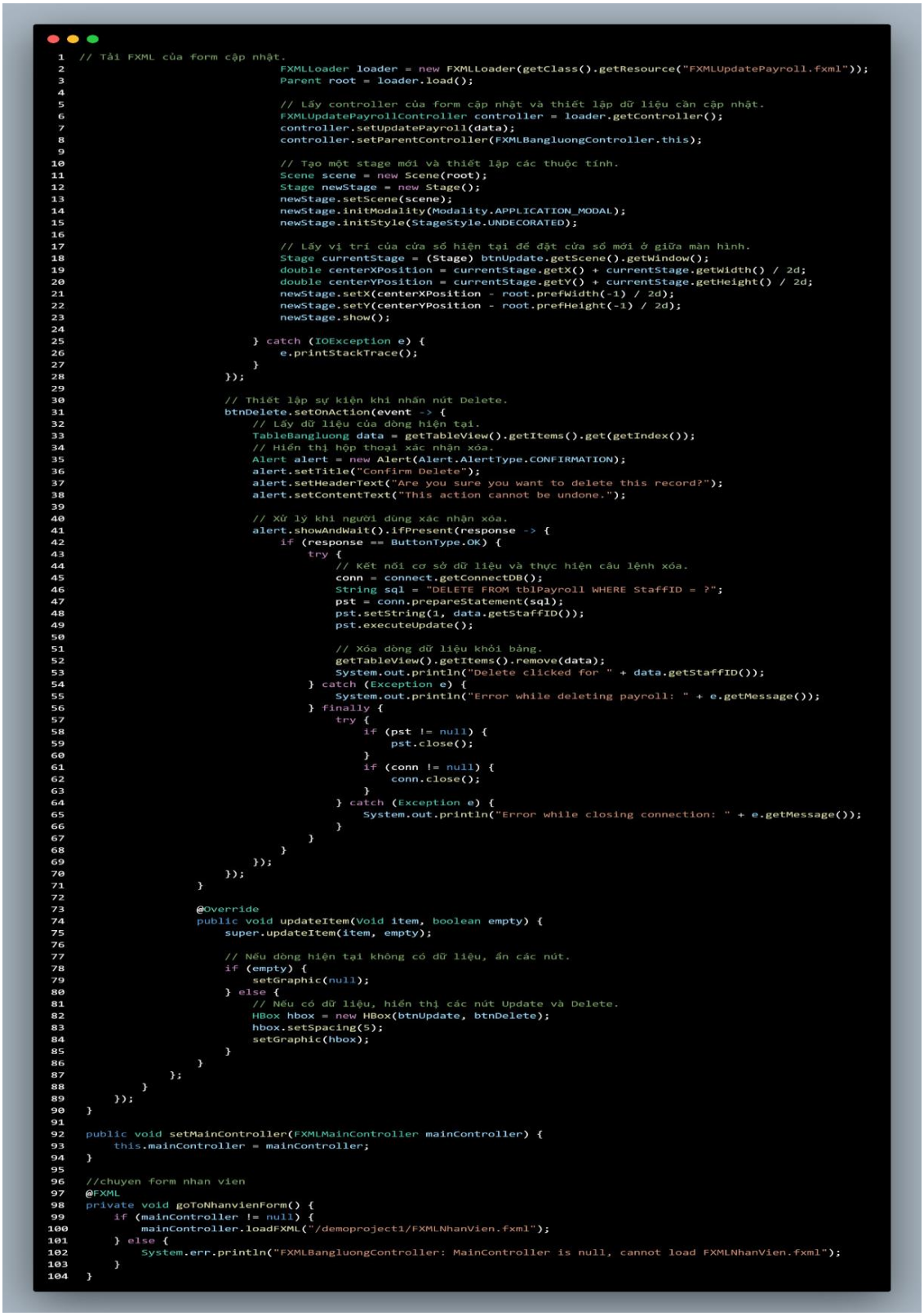
15 Payroll

```java
public class FXMLBangluongController implements Initializable {

    @FXML
    private FXMLMainController mainController;
    @FXML
    private Button btnExport;

    @FXML
    private Button btnthempayroll;

    @FXML
    private TableColumn<TableBangluong, Integer> cotgiotieuchuan;

    @FXML
    private TableColumn<TableBangluong, String> cotstaffid;

    @FXML
    private TableColumn<TableBangluong, Date> cotngaybatdau;

    @FXML
    private TableColumn<TableBangluong, Date> cotngayketthuc;

    @FXML
    private TableColumn<TableBangluong, Integer> cotngaytieuchuan;

    @FXML
    private TableColumn<TableBangluong, BigDecimal> cotsongaylam;

    @FXML
    private TableColumn<TableBangluong, BigDecimal> cottangca;

    @FXML
    private TableColumn<TableBangluong, BigDecimal> cottraluong;

    @FXML
    private TableColumn<TableBangluong, Integer> cotID;

    @FXML
    private TableColumn<TableBangluong, BigDecimal> cotSalary;

    @FXML
    private TableView<TableBangluong> tablepayroll;

    @FXML
    private TextField txttukhoa;

    @FXML
    private Button btnStaff;

    @FXML
    private Button btnTimekeeping;

    @FXML
    private TableColumn<TableBangluong, Void> cotaction; // Để sử dụng Void vì không cần dữ liệu, chỉ cần hiển thị nút

    public ConnectDB connect = new ConnectDB();
    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;

    public ObservableList<TableBangluong> dataList() {
        ObservableList<TableBangluong> dataList = FXCollections.observableArrayList();
        String sql = "SELECT * FROM tblPayroll";

        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();

            while (rs.next()) {
                TableBangluong data = new TableBangluong(
                        rs.getInt("ID"),
                        rs.getString("StaffID"),
                        rs.getDate("PayPeriodStart"),
                        rs.getDate("PayPeriodEnd"),
                        rs.getInt("StandardDays"),
                        rs.getInt("StandardHours"),
                        rs.getBigDecimal("TotalWorkDays"),
                        rs.getBigDecimal("Overtime"),
                        rs.getBigDecimal("TotalPay"),
                        rs.getBigDecimal("Salary")
                );
                dataList.add(data);
            }
        } catch (Exception e) {
            System.out.println("Error while fetching data: " + e.getMessage());
        }

        return dataList;
    }
```

15.1 Code

```
public void showDataBangluong() {
        ObservableList<TableBangluong> showList = dataList();
        cotID.setCellValueFactory(new PropertyValueFactory<>("ID"));
        cotstaffid.setCellValueFactory(new PropertyValueFactory<>("staffID"));
        cotngaybatdau.setCellValueFactory(new PropertyValueFactory<>("payPeriodStart"));
        cotngayketthuc.setCellValueFactory(new PropertyValueFactory<>("payPeriodEnd"));
        cotngaytieuchuan.setCellValueFactory(new PropertyValueFactory<>("standardDays"));
        cotgiotieuchuan.setCellValueFactory(new PropertyValueFactory<>("standardHours"));
        cotsongaylam.setCellValueFactory(new PropertyValueFactory<>("totalWorkDays"));
        cottangca.setCellValueFactory(new PropertyValueFactory<>("overtime"));
        cottraluong.setCellValueFactory(new PropertyValueFactory<>("totalPay"));
        cotSalary.setCellValueFactory(new PropertyValueFactory<>("Salary"));

        cotID.setCellFactory(column -> {
            return new TableCell<TableBangluong, Integer>() {
                @Override
                protected void updateItem(Integer item, boolean empty) {
                    super.updateItem(item, empty);
                    if (empty) {
                        setText(null);
                    } else {
                        setText(String.valueOf(getIndex() + 1));
                    }
                }
            };
        });

        tablepayroll.setItems(showList);
    }

    //search
    public void searchStaff() {
        String keyword = txttukhoa.getText().toLowerCase().trim();

        ObservableList<TableBangluong> filteredList = FXCollections.observableArrayList();

        for (TableBangluong staff : dataList()) {
            if (staff.getStaffID().toLowerCase().contains(keyword)) {
                filteredList.add(staff);
            }
        }

        tablepayroll.setItems(filteredList);
    }

    //chức năng xuất dữ liệu ra excel bằng nút Export
    public void exportToCSV() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save CSV File");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "*.csv"));
        File selectedFile = fileChooser.showSaveDialog(btnExport.getScene().getWindow());

        if (selectedFile != null) {
            try {
                Writer writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(selectedFile), StandardCharsets.UTF_8));
                writer.write('\uFEFF'); // BOM for UTF-8

                // Ghi tiêu đề cột
                TableView<TableBangluong> tableView = tablepayroll;
                ObservableList<TableColumn<TableBangluong, ?>> columns = tableView.getColumns();
                for (int i = 0; i < columns.size(); i++) {
                    TableColumn<TableBangluong, ?> column = columns.get(i);
                    if (i > 0) {
                        writer.write(",");
                    }
                    writer.write(column.getText());
                }
                writer.write("\n");

                // Ghi dữ liệu từng dòng
                ObservableList<TableBangluong> dataList = tableView.getItems();
                for (TableBangluong staff : dataList) {
                    for (int i = 0; i < columns.size(); i++) {
                        TableColumn<TableBangluong, ?> column = columns.get(i);
                        if (i > 0) {
                            writer.write(",");
                        }
                        // Lấy giá trị từ cell của bảng và ghi vào writer
                        Object cellValue = column.getCellData(staff);
                        writer.write(cellValue != null ? cellValue.toString() : "");
                    }
                    writer.write("\n");
                }
                writer.flush();
                writer.close();

                Thuvien tv = new Thuvien(); // Thêm dòng này nếu Thuvien cần được sử dụng
                tv.showAlert("Data exported to CSV successfully!");

                // Mở file CSV sau khi export
                Desktop.getDesktop().open(selectedFile);

            } catch (IOException e) {
                System.out.println("Error while exporting to CSV: " + e.getMessage());
            }
        }
    }

    //action
    public void setupActionColumn() {
        // Thiết lập cell factory cho cột hành động.
        cotaction.setCellFactory(new Callback<TableColumn<TableBangluong, Void>, TableCell<TableBangluong, Void>>() {
            @Override
            public TableCell<TableBangluong, Void> call(TableColumn<TableBangluong, Void> param) {
                // Tạo một TableCell mới để chứa các nút hành động.
                return new TableCell<TableBangluong, Void>() {
                    // Khai báo và khởi tạo các nút Update và Delete.
                    private final Button btnUpdate = new Button("Update");
                    private final Button btnDelete = new Button("Delete");

                    {
                        // Đặt style cho các nút.
                        btnUpdate.getStyleClass().add("button-update");
                        btnDelete.getStyleClass().add("button-delete");

                        // Thiết lập sự kiện khi nhấn nút Update.
                        btnUpdate.setOnAction(event -> {
                            // Lấy dữ liệu của dòng hiện tại.
                            TableBangluong data = getTableView().getItems().get(getIndex());
                            try {
```

```java
public void showDataBangluong() {
        ObservableList<TableBangluong> showList = dataList();
        cotID.setCellValueFactory(new PropertyValueFactory<>("ID"));
        cotstaffid.setCellValueFactory(new PropertyValueFactory<>("staffID"));
        cotngaybatdau.setCellValueFactory(new PropertyValueFactory<>("payPeriodStart"));
        cotngayketthuc.setCellValueFactory(new PropertyValueFactory<>("payPeriodEnd"));
        cotngaytieuchuan.setCellValueFactory(new PropertyValueFactory<>("standardDays"));
        cotgiotieuchuan.setCellValueFactory(new PropertyValueFactory<>("standardHours"));
        cotsongaylam.setCellValueFactory(new PropertyValueFactory<>("totalWorkDays"));
        cottangca.setCellValueFactory(new PropertyValueFactory<>("overtime"));
        cottraluong.setCellValueFactory(new PropertyValueFactory<>("totalPay"));
        cotSalary.setCellValueFactory(new PropertyValueFactory<>("Salary"));

        cotID.setCellFactory(column -> {
            return new TableCell<TableBangluong, Integer>() {
                @Override
                protected void updateItem(Integer item, boolean empty) {
                    super.updateItem(item, empty);
                    if (empty) {
                        setText(null);
                    } else {
                        setText(String.valueOf(getIndex() + 1));
                    }
                }
            };
        });

        tablepayroll.setItems(showList);
    }

    //search
    public void searchStaff() {
        String keyword = txttukhoa.getText().toLowerCase().trim();

        ObservableList<TableBangluong> filteredList = FXCollections.observableArrayList();

        for (TableBangluong staff : dataList()) {
            if (staff.getStaffID().toLowerCase().contains(keyword)) {
                filteredList.add(staff);
            }
        }

        tablepayroll.setItems(filteredList);
    }

    //chức năng xuất dữ liệu ra excel bằng nút Export
    public void exportToCSV() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save CSV File");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "*.csv"));
        File selectedFile = fileChooser.showSaveDialog(btnExport.getScene().getWindow());

        if (selectedFile != null) {
            try {
                Writer writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(selectedFile), StandardCharsets.UTF_8));
                writer.write('\uFEFF'); // BOM for UTF-8

                // Ghi tiêu đề cột
                TableView<TableBangluong> tableView = tablepayroll;
                ObservableList<TableColumn<TableBangluong, ?>> columns = tableView.getColumns();
                for (int i = 0; i < columns.size(); i++) {
                    TableColumn<TableBangluong, ?> column = columns.get(i);
                    if (i > 0) {
                        writer.write(",");
                    }
                    writer.write(column.getText());
                }
                writer.write("\n");

                // Ghi dữ liệu từng dòng
                ObservableList<TableBangluong> dataList = tableView.getItems();
                for (TableBangluong staff : dataList) {
                    for (int i = 0; i < columns.size(); i++) {
                        TableColumn<TableBangluong, ?> column = columns.get(i);
                        if (i > 0) {
                            writer.write(",");
                        }
                        // Lấy giá trị từ cell của bảng và ghi vào writer
                        Object cellValue = column.getCellData(staff);
                        writer.write(cellValue != null ? cellValue.toString() : "");
                    }
                    writer.write("\n");
                }
                writer.flush();
                writer.close();

                Thuvien tv = new Thuvien(); // Thêm dòng này nếu Thuvien cần được sử dụng
                tv.showAlert("Data exported to CSV successfully!");

                // Mở file CSV sau khi export
                Desktop.getDesktop().open(selectedFile);

            } catch (IOException e) {
                System.out.println("Error while exporting to CSV: " + e.getMessage());
            }
        }
    }

    //action
    public void setupActionColumn() {
        // Thiết lập cell factory cho cột hành động.
        cotaction.setCellFactory(new Callback<TableColumn<TableBangluong, Void>, TableCell<TableBangluong, Void>>() {
            @Override
            public TableCell<TableBangluong, Void> call(TableColumn<TableBangluong, Void> param) {
                // Tạo một TableCell mới để chứa các nút hành động.
                return new TableCell<TableBangluong, Void>() {
                    // Khai báo và khởi tạo các nút Update và Delete.
                    private final Button btnUpdate = new Button("Update");
                    private final Button btnDelete = new Button("Delete");

                    {
                        // Đặt style cho các nút.
                        btnUpdate.getStyleClass().add("button-update");
                        btnDelete.getStyleClass().add("button-delete");

                        // Thiết lập sự kiện khi nhấn nút Update.
                        btnUpdate.setOnAction(event -> {
                            // Lấy dữ liệu của dòng hiện tại.
                            TableBangluong data = getTableView().getItems().get(getIndex());
                            try {
```



```
// Tải FXML của form cập nhật.
                            FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLUpdatePayroll.fxml"));
                            Parent root = loader.load();

                            // Lấy controller của form cập nhật và thiết lập dữ liệu cần cập nhật.
                            FXMLUpdatePayrollController controller = loader.getController();
                            controller.setUpdatePayroll(data);
                            controller.setParentController(FXMLBangluongController.this);

                            // Tạo một stage mới và thiết lập các thuộc tính.
                            Scene scene = new Scene(root);
                            Stage newStage = new Stage();
                            newStage.setScene(scene);
                            newStage.initModality(Modality.APPLICATION_MODAL);
                            newStage.initStyle(StageStyle.UNDECORATED);

                            // Lấy vị trí của cửa sổ hiện tại để đặt cửa sổ mới ở giữa màn hình.
                            Stage currentStage = (Stage) btnUpdate.getScene().getWindow();
                            double centerXPosition = currentStage.getX() + currentStage.getWidth() / 2d;
                            double centerYPosition = currentStage.getY() + currentStage.getHeight() / 2d;
                            newStage.setX(centerXPosition - root.prefWidth(-1) / 2d);
                            newStage.setY(centerYPosition - root.prefHeight(-1) / 2d);
                            newStage.show();

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

                    // Thiết lập sự kiện khi nhấn nút Delete.
                    btnDelete.setOnAction(event -> {
                        // Lấy dữ liệu của dòng hiện tại.
                        TableBangluong data = getTableView().getItems().get(getIndex());
                        // Hiển thị hộp thoại xác nhận xóa.
                        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
                        alert.setTitle("Confirm Delete");
                        alert.setHeaderText("Are you sure you want to delete this record?");
                        alert.setContentText("This action cannot be undone.");

                        // Xử lý khi người dùng xác nhận xóa.
                        alert.showAndWait().ifPresent(response -> {
                            if (response == ButtonType.OK) {
                                try {
                                    // Kết nối cơ sở dữ liệu và thực hiện câu lệnh xóa.
                                    conn = connect.getConnectDB();
                                    String sql = "DELETE FROM tblPayroll WHERE StaffID = ?";
                                    pst = conn.prepareStatement(sql);
                                    pst.setString(1, data.getStaffID());
                                    pst.executeUpdate();

                                    // Xóa dòng dữ liệu khỏi bảng.
                                    getTableView().getItems().remove(data);
                                    System.out.println("Delete clicked for " + data.getStaffID());
                                } catch (Exception e) {
                                    System.out.println("Error while deleting payroll: " + e.getMessage());
                                } finally {
                                    try {
                                        if (pst != null) {
                                            pst.close();
                                        }
                                        if (conn != null) {
                                            conn.close();
                                        }
                                    } catch (Exception e) {
                                        System.out.println("Error while closing connection: " + e.getMessage());
                                    }
                                }
                            }
                        });
                    });
                }

                @Override
                public void updateItem(Void item, boolean empty) {
                    super.updateItem(item, empty);

                    // Nếu dòng hiện tại không có dữ liệu, ẩn các nút.
                    if (empty) {
                        setGraphic(null);
                    } else {
                        // Nếu có dữ liệu, hiển thị các nút Update và Delete.
                        HBox hbox = new HBox(btnUpdate, btnDelete);
                        hbox.setSpacing(5);
                        setGraphic(hbox);
                    }
                }
            };
        }
    });
}

public void setMainController(FXMLMainController mainController) {
    this.mainController = mainController;
}

//chuyen form nhan vien
@FXML
private void goToNhanvienForm() {
    if (mainController != null) {
        mainController.loadFXML("/demoproject1/FXMLNhanVien.fxml");
    } else {
        System.err.println("FXMLBangluongController: MainController is null, cannot load FXMLNhanVien.fxml");
    }
}
```

```java
@FXML
    private void goToChamluongFormFromBangluong() {
        if (mainController != null) {
            mainController.loadFXML("/demoproject1/FXMLChamluong.fxml");
        } else {
            System.err.println("FXMLBangluongController: MainController is null, cannot load FXMLChamluong.fxml");
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        showDataBangluong();
        setupActionColumn(); //them nut vao action
        //search
        txttukhoa.textProperty().addListener((observable, oldValue, newValue) -> {
            searchStaff();
        });
        //xuất ra excel
        btnExport.setOnAction(event -> exportToCSV());

        // Bắt sự kiện khi nhấn nút Chuyển form
        btnthempayroll.setOnAction(event -> {
            try {
                // Load file FXML của form mới
                FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLThemPayroll.fxml"));
                Parent root = loader.load();

                // Tạo một Scene mới
                Scene scene = new Scene(root);

                // Tạo một Stage mới cho form mới
                Stage newStage = new Stage();

                // Đặt modality để đảm bảo form mới đè lên và không thao tác được form cũ
                newStage.initModality(Modality.APPLICATION_MODAL);

                // Đặt style để không có nút đóng góc
                newStage.initStyle(StageStyle.UNDECORATED);

                // Đặt Scene vào Stage mới
                newStage.setScene(scene);

                // Lấy Stage hiện tại từ Button
                Stage currentStage = (Stage) btnthempayroll.getScene().getWindow();

                // Tính toán vị trí để đưa form mới vào giữa màn hình
                double centerXPosition = currentStage.getX() + currentStage.getWidth() / 2d;
                double centerYPosition = currentStage.getY() + currentStage.getHeight() / 2d;

                // Thiết lập vị trí của Stage mới ở giữa màn hình
                newStage.setX(centerXPosition - root.prefWidth(-1) / 2d);
                newStage.setY(centerYPosition - root.prefHeight(-1) / 2d);

                // Hiển thị Stage mới và không đóng Stage cũ
                newStage.show();

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

    }
}

```

16 TimeKeeping

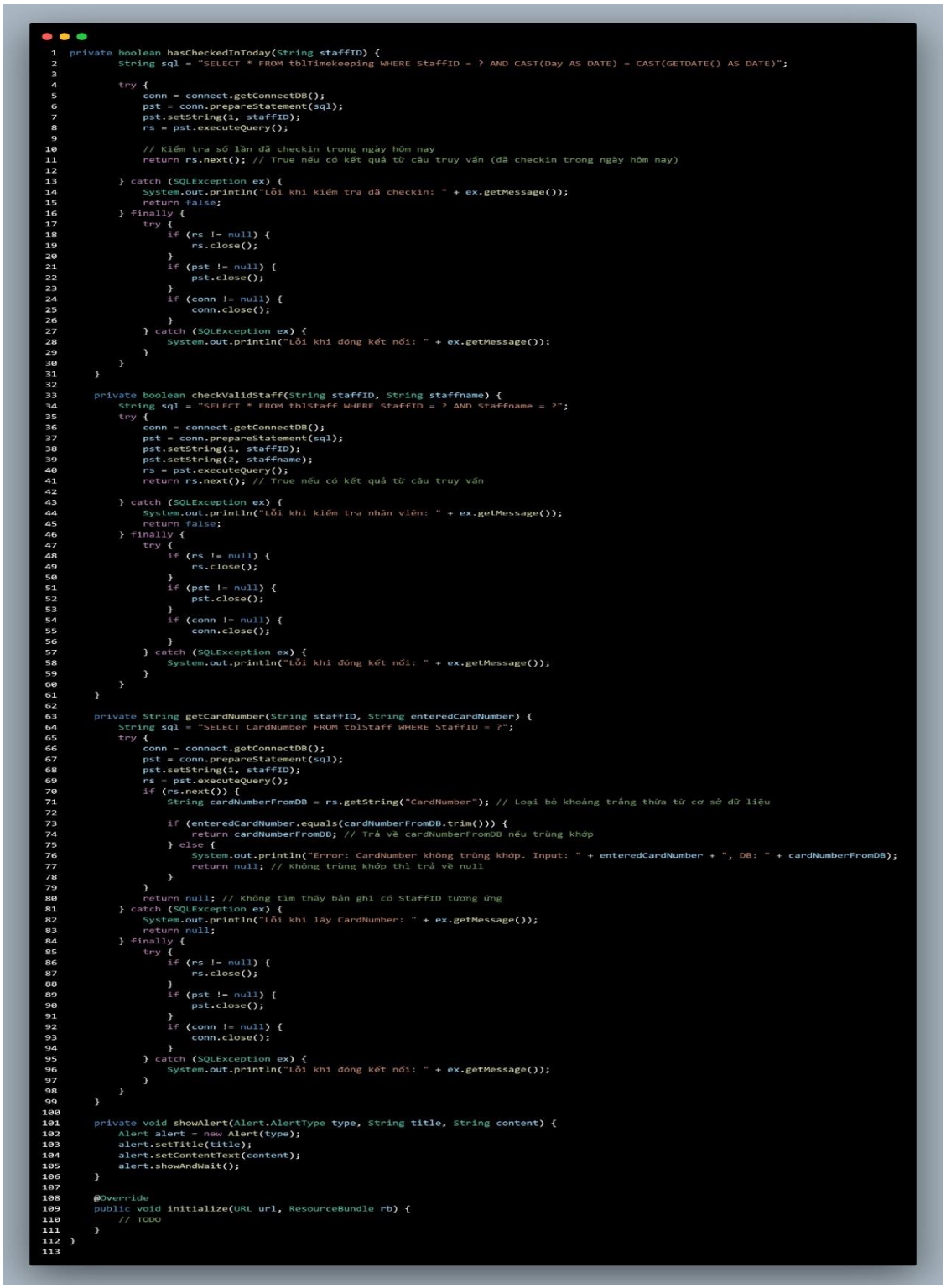
```java
public class FXMLChamluongController implements Initializable {

    @FXML
    private FXMLMainController mainController;
    @FXML
    private Button btnPayroll;

    @FXML
    private Button btnCheckin;

    @FXML
    private Button btnStaff;
    @FXML
    private Button btnExport;

    @FXML
    private TableColumn<TableChamluong, Integer> cotID;

    @FXML
    private TableColumn<TableChamluong, String> cotStaffid;

    @FXML
    private TableColumn<TableChamluong, String> cotStaffname;

    @FXML
    private TableColumn<TableChamluong, Date> cotcheckin;

    @FXML
    private TableColumn<TableChamluong, Date> cotcheckout;

    @FXML
    private TableColumn<TableChamluong, Date> cotday;

    @FXML
    private TableView<TableChamluong> tableTimekeeping;

    @FXML
    private TextField txttukhoa;

    public ConnectDB connect = new ConnectDB();
    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;

    /**
     * Initializes the controller class.
     */
    public ObservableList<TableChamluong> dataList() {
        ObservableList<TableChamluong> dataList = FXCollections.observableArrayList();
        String sql = "SELECT * FROM tblTimekeeping";

        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();

            while (rs.next()) {
                TableChamluong data = new TableChamluong(
                        rs.getInt("ID"),
                        rs.getString("StaffID"),
                        rs.getString("Staffname"),
                        rs.getDate("Day"),
                        rs.getTime("Checkin"),
                        rs.getTime("Checkout")
                );
                dataList.add(data);
            }
        } catch (SQLException e) {
            System.out.println("Lỗi khi lấy dữ liệu: " + e.getMessage());
        } finally {
            closeResources();
        }

        return dataList;
    }

    public void showDataChamluong() {
        ObservableList<TableChamluong> showList = dataList();

        cotID.setCellValueFactory(new PropertyValueFactory<>("ID"));
        cotStaffid.setCellValueFactory(new PropertyValueFactory<>("StaffID"));
        cotStaffname.setCellValueFactory(new PropertyValueFactory<>("Staffname"));
        cotday.setCellValueFactory(new PropertyValueFactory<>("Day"));
        cotcheckin.setCellValueFactory(new PropertyValueFactory<>("Checkin"));
        cotcheckout.setCellValueFactory(new PropertyValueFactory<>("Checkout"));

        cotID.setCellFactory(column -> {
            return new TableCell<TableChamluong, Integer>() {
                @Override
                protected void updateItem(Integer item, boolean empty) {
                    super.updateItem(item, empty);
                    if (empty) {
                        setText(null);
                    } else {
                        setText(String.valueOf(getIndex() + 1));
                    }
                }
            };
        });

        tableTimekeeping.setItems(showList);
    }
```

16.1 Code

```
@FXML
    public void refreshTable() {
        showDataChamluong();
    }

    public void searchStaff() {
        String keyword = txttukhoa.getText().toLowerCase().trim();

        if (keyword.isEmpty()) {
            showDataChamluong(); // Nếu ô tìm kiếm trống, hiển thị lại dữ liệu ban đầu
            return;
        }

        ObservableList<TableChamluong> filteredList = FXCollections.observableArrayList();

        // Thực hiện truy vấn tìm kiếm
        String sql = "SELECT * FROM tblTimekeeping WHERE StaffID LIKE ?";

        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);
            pst.setString(1, "%" + keyword + "%");
            rs = pst.executeQuery();

            while (rs.next()) {
                TableChamluong data = new TableChamluong(
                        rs.getInt("ID"),
                        rs.getString("StaffID"),
                        rs.getString("Staffname"),
                        rs.getDate("Day"),
                        rs.getTime("Checkin"),
                        rs.getTime("Checkout")
                );
                filteredList.add(data);
            }
        } catch (SQLException e) {
            System.out.println("Lỗi khi tìm kiếm: " + e.getMessage());
        } finally {
            closeResources();
        }

        // Hiển thị kết quả tìm kiếm trong TableView
        tableTimekeeping.setItems(filteredList);
    }

    public void setMainController(FXMLMainController mainController) {
        this.mainController = mainController;
    }

     @FXML
    private void goToNhanvienForm() {
        if (mainController != null) {
            mainController.loadFXML("/demoproject1/FXMLNhanVien.fxml");
        } else {
            System.err.println("FXMLChamluongController: MainController is null, cannot load FXMLNhanVien.fxml");
        }
    }

    @FXML
    private void goToBangluongFormFromChamluong() {
        if (mainController != null) {
            mainController.loadFXML("/demoproject1/FXMLBangluong.fxml");
        } else {
            System.err.println("FXMLChamluongController: MainController is null, cannot load FXMLBangluong.fxml");
        }
    }

    //chức năng xuất dữ liệu ra excel bằng nút Export
    public void exportToCSV() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save CSV File");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "*.csv"));
        File selectedFile = fileChooser.showSaveDialog(btnExport.getScene().getWindow());

        if (selectedFile != null) {
            try {
                Writer writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(selectedFile), StandardCharsets.UTF_8));
                writer.write('\uFEFF'); // BOM for UTF-8

                // Ghi tiêu đề cột
                TableView<TableChamluong> tableView = tableTimekeeping;
                ObservableList<TableColumn<TableChamluong, ?>> columns = tableView.getColumns();
                for (int i = 0; i < columns.size(); i++) {
                    TableColumn<TableChamluong, ?> column = columns.get(i);
                    if (i > 0) {
                        writer.write(",");
                    }
                    writer.write(column.getText());
                }
                writer.write("\n");

                // Ghi dữ liệu từng dòng
                ObservableList<TableChamluong> dataList = tableView.getItems();
                for (TableChamluong staff : dataList) {
                    for (int i = 0; i < columns.size(); i++) {
                        TableColumn<TableChamluong, ?> column = columns.get(i);
                        if (i > 0) {
                            writer.write(",");
                        }
                        // Lấy giá trị từ cell của bảng và ghi vào writer
                        Object cellValue = column.getCellData(staff);
                        writer.write(cellValue != null ? cellValue.toString() : "");
                    }
                    writer.write("\n");
                }
                writer.flush();
                writer.close();

                Thuvien tv = new Thuvien(); // Thêm dòng này nếu Thuvien cần được sử dụng
                tv.showAlert("Data exported to CSV successfully!");

                // Mở file CSV sau khi export
                Desktop.getDesktop().open(selectedFile);

            } catch (IOException e) {
                System.out.println("Error while exporting to CSV: " + e.getMessage());
            }
        }
    }
```

```java
@Override
    public void initialize(URL url, ResourceBundle rb) {

        txttukhoa.textProperty().addListener((observable, oldValue, newValue) -> {
            searchStaff();
        });

        showDataChamluong();

        // Bắt sự kiện khi nhấn nút Chuyển form
        btnCheckin.setOnAction(event -> {
            try {
                FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLCheckin.fxml"));
                Parent root = loader.load();

                FXMLCheckinController controller = loader.getController();
                controller.setParentController(this); // Truyền FXMLChamluongController vào đây

                Scene scene = new Scene(root);
                Stage newStage = new Stage();
                newStage.initModality(Modality.APPLICATION_MODAL);
                newStage.initStyle(StageStyle.UNDECORATED);
                newStage.setScene(scene);

                Stage currentStage = (Stage) btnCheckin.getScene().getWindow();
                double centerXPosition = currentStage.getX() + currentStage.getWidth() / 2d;
                double centerYPosition = currentStage.getY() + currentStage.getHeight() / 2d;
                newStage.setX(centerXPosition - root.prefWidth(-1) / 2d);
                newStage.setY(centerYPosition - root.prefHeight(-1) / 2d);

                newStage.show();
            } catch (IOException e) {
                e.printStackTrace();
            }
        });

        // Bắt sự kiện khi nhấn đúp vào hàng trong bảng
        tableTimekeeping.setOnMouseClicked(event -> {
            if (event.getClickCount() == 2) {
                TableChamluong selectedRecord = tableTimekeeping.getSelectionModel().getSelectedItem();
                if (selectedRecord != null && selectedRecord.getCheckout() == null) {
                    try {
                        FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLCheckout.fxml"));
                        Parent root = loader.load();

                        FXMLCheckoutController controller = loader.getController();
                        controller.setParentController(this); // Truyền FXMLChamluongController vào đây
                        controller.setSelectedRecord(selectedRecord); // Truyền dữ liệu dòng được chọn vào controller của FXMLCheckout

                        Scene scene = new Scene(root);
                        Stage newStage = new Stage();
                        newStage.initModality(Modality.APPLICATION_MODAL);
                        newStage.initStyle(StageStyle.UNDECORATED);
                        newStage.setScene(scene);

                        Stage currentStage = (Stage) tableTimekeeping.getScene().getWindow();
                        double centerXPosition = currentStage.getX() + currentStage.getWidth() / 2d;
                        double centerYPosition = currentStage.getY() + currentStage.getHeight() / 2d;
                        newStage.setX(centerXPosition - root.prefWidth(-1) / 2d);
                        newStage.setY(centerYPosition - root.prefHeight(-1) / 2d);

                        newStage.show();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                } else {
                    showAlert(Alert.AlertType.ERROR, "Error", "This employee has checked out.");
                }
            }
        });
    }

    private void showAlert(Alert.AlertType type, String title, String content) {
        Alert alert = new Alert(type);
        alert.setTitle(title);
        alert.setContentText(content);
        alert.showAndWait();
    }

    private void closeResources() {
        try {
            if (rs != null) {
                rs.close();
            }
            if (pst != null) {
                pst.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            System.out.println("Lỗi khi đóng các tài nguyên: " + e.getMessage());
        }
    }
}

```

17 Checkin



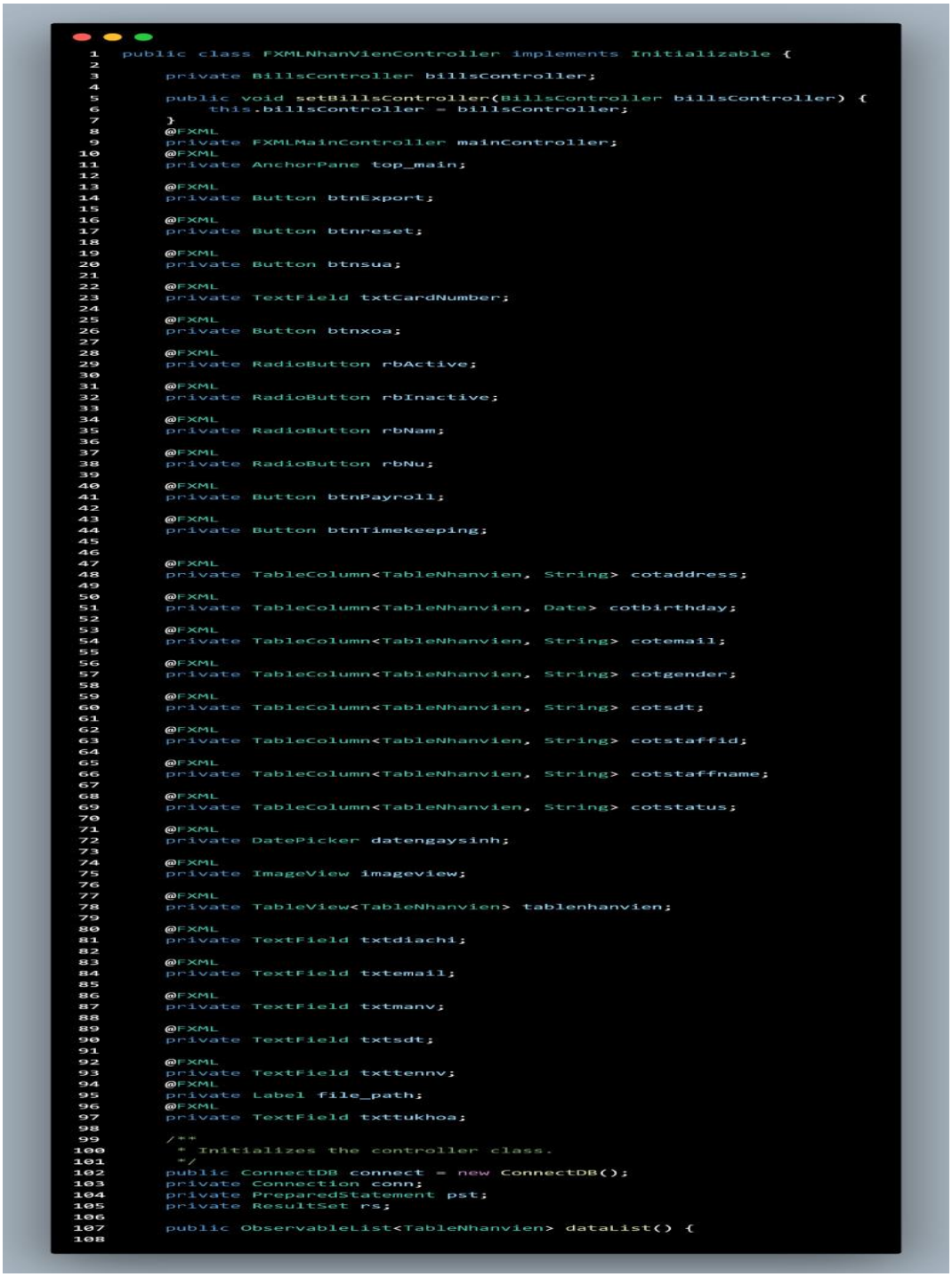
```
private boolean hasCheckedInToday(String staffID) {
        String sql = "SELECT * FROM tbltimekeeping WHERE StaffID = ? AND CAST(Day AS DATE) = CAST(GETDATE() AS DATE)";

        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);
            pst.setString(1, staffID);
            rs = pst.executeQuery();

            // Kiểm tra số lần đã checkin trong ngày hôm nay
            return rs.next(); // True nếu có kết quả từ câu truy vấn (đã checkin trong ngày hôm nay)

        } catch (SQLException ex) {
            System.out.println("Lỗi khi kiểm tra đã checkin: " + ex.getMessage());
            return false;
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (pst != null) {
                    pst.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException ex) {
                System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
            }
        }
    }

    private boolean checkValidStaff(String staffID, String staffname) {
        String sql = "SELECT * FROM tblStaff WHERE StaffID = ? AND Staffname = ?";
        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);
            pst.setString(1, staffID);
            pst.setString(2, staffname);
            rs = pst.executeQuery();
            return rs.next(); // True nếu có kết quả từ câu truy vấn

        } catch (SQLException ex) {
            System.out.println("Lỗi khi kiểm tra nhân viên: " + ex.getMessage());
            return false;
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (pst != null) {
                    pst.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException ex) {
                System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
            }
        }
    }

    private String getCardNumber(String staffID, String enteredCardNumber) {
        String sql = "SELECT CardNumber FROM tblStaff WHERE StaffID = ?";
        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);
            pst.setString(1, staffID);
            rs = pst.executeQuery();
            if (rs.next()) {
                String cardNumberFromDB = rs.getString("CardNumber"); // Loại bỏ khoảng trắng thừa từ cơ sở dữ liệu

                if (enteredCardNumber.equals(cardNumberFromDB.trim())) {
                    return cardNumberFromDB; // Trả về cardNumberFromDB nếu trùng khớp
                } else {
                    System.out.println("Error: CardNumber không trùng khớp. Input: " + enteredCardNumber + ", DB: " + cardNumberFromDB);
                    return null; // Không trùng khớp thì trả về null
                }
            }
            return null; // Không tìm thấy bản ghi có StaffID tương ứng
        } catch (SQLException ex) {
            System.out.println("Lỗi khi lấy CardNumber: " + ex.getMessage());
            return null;
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (pst != null) {
                    pst.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException ex) {
                System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
            }
        }
    }

    private void showAlert(Alert.AlertType type, String title, String content) {
        Alert alert = new Alert(type);
        alert.setTitle(title);
        alert.setContentText(content);
        alert.showAndWait();
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
}
```

```java
public class FXMLCheckinController implements Initializable {

    @FXML
    private Button btnCheckin;

    @FXML
    private TextField txtStaffID;

    @FXML
    private TextField txtStaffname;

    @FXML
    private TextField txtCheckin;

    @FXML
    private Button btncancel;

    private FXMLChamluongController parentController;

    public ConnectDB connect = new ConnectDB();
    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;

    public void setParentController(FXMLChamluongController parentController) {
        this.parentController = parentController;
    }

    @FXML
    public void handleCancel() {
        Stage stage = (Stage) btncancel.getScene().getWindow();
        stage.close();
    }

    @FXML
    public void handleCheckin() {
        String staffID = txtStaffID.getText();
        String staffname = txtStaffname.getText();
        String enteredCardNumber = txtCheckin.getText(); // Lấy giá trị CardNumber mà nhân viên nhập vào

        if (staffID.isEmpty() || staffname.isEmpty() || enteredCardNumber.isEmpty()) {
            showAlert(Alert.AlertType.ERROR, "Error", "Please enter Staff AND, Staff Name and Card Number.");
            return;
        }

        // Kiểm tra StaffID và Staffname có trong tblStaff không
        boolean validStaff = checkValidStaff(staffID, staffname);
        if (!validStaff) {
            showAlert(Alert.AlertType.ERROR, "Error", "Invalid StaffID or Staffname.");
            return;
        }

        // Lấy CardNumber từ tblStaff và kiểm tra tính hợp lệ của CardNumber
        String cardNumber = getCardNumber(staffID, enteredCardNumber);
        if (cardNumber == null) {
            showAlert(Alert.AlertType.ERROR, "Error", "CardNumber is invalid or does not match.");
            return;
        }

        // Kiểm tra đã checkin trong ngày hôm nay chưa
        boolean hasCheckedInToday = hasCheckedInToday(staffID);
        if (hasCheckedInToday) {
            showAlert(Alert.AlertType.ERROR, "Error", "Staff checked in today.");
            return;
        }

        Date currentDate = new Date();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        // Cập nhật dữ liệu vào tblTimekeeping
        String sql = "INSERT INTO tblTimekeeping (StaffID, Staffname, Day, Checkin) VALUES (?, ?, ?, ?)";

        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);
            pst.setString(1, staffID);
            pst.setString(2, staffname);
            pst.setString(3, dateFormat.format(currentDate));
            pst.setString(4, dateFormat.format(currentDate));
            pst.executeUpdate();

            // Đóng cửa sổ sau khi checkin thành công
            Stage stage = (Stage) btnCheckin.getScene().getWindow();
            stage.close();

            // Cập nhật lại TableView trong FXMLChamluongController
            if (parentController != null) {
                parentController.refreshTable();
            }

        } catch (SQLException ex) {
            showAlert(Alert.AlertType.ERROR, "Error", "Error when performing Checkin:" + ex.getMessage());
        } finally {
            try {
                if (pst != null) {
                    pst.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException ex) {
                System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
            }
        }
    }
```

18 Checkout

```java
public class FXMLCheckoutController implements Initializable {

    @FXML
    private Button btnCheckout;

    @FXML
    private Button btncancel;

    @FXML
    private TextField txtStaffID;

    @FXML
    private TextField txtStaffname;

    private FXMLChamluongController parentController;
    private TableChamluong selectedRecord;

    public ConnectDB connect = new ConnectDB();
    private Connection conn;
    private PreparedStatement pst;

    public void setParentController(FXMLChamluongController parentController) {
        this.parentController = parentController;
    }

    public void setSelectedRecord(TableChamluong selectedRecord) {
        this.selectedRecord = selectedRecord;
        // Kiểm tra xem selectedRecord có null hay không
        if (selectedRecord != null) {
            // Hiển thị dữ liệu lên các ô nhập
            txtStaffID.setText(selectedRecord.getStaffID());
            txtStaffname.setText(selectedRecord.getStaffname());
        } else {
            System.out.println("selectedRecord is null");
        }
    }

    @FXML
    public void handleCancel() {
        Stage stage = (Stage) btncancel.getScene().getWindow();
        stage.close();
    }

    @FXML
    public void handleCheckout() {
        Date currentDate = new Date();
        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

        // Cập nhật dữ liệu vào tblTimekeeping
        String sql = "UPDATE tblTimekeeping SET Checkout = ? WHERE ID = ?";

        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);
            pst.setString(1, dateFormat.format(currentDate));
            pst.setInt(2, selectedRecord.getID());
            pst.executeUpdate();

            // Đóng cửa sổ sau khi checkout thành công
            Stage stage = (Stage) btnCheckout.getScene().getWindow();
            stage.close();

            // Cập nhật lại TableView trong FXMLChamluongController
            if (parentController != null) {
                parentController.refreshTable();
            }

        } catch (SQLException ex) {
            showAlert(Alert.AlertType.ERROR, "Error", "Error when performing Checkout:" + ex.getMessage());
        } finally {
            try {
                if (pst != null) {
                    pst.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException ex) {
                System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
            }
        }
    }

    private void showAlert(Alert.AlertType type, String title, String content) {
        Alert alert = new Alert(type);
        alert.setTitle(title);
        alert.setContentText(content);
        alert.showAndWait();
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // TODO
    }
}

```

19 Staff



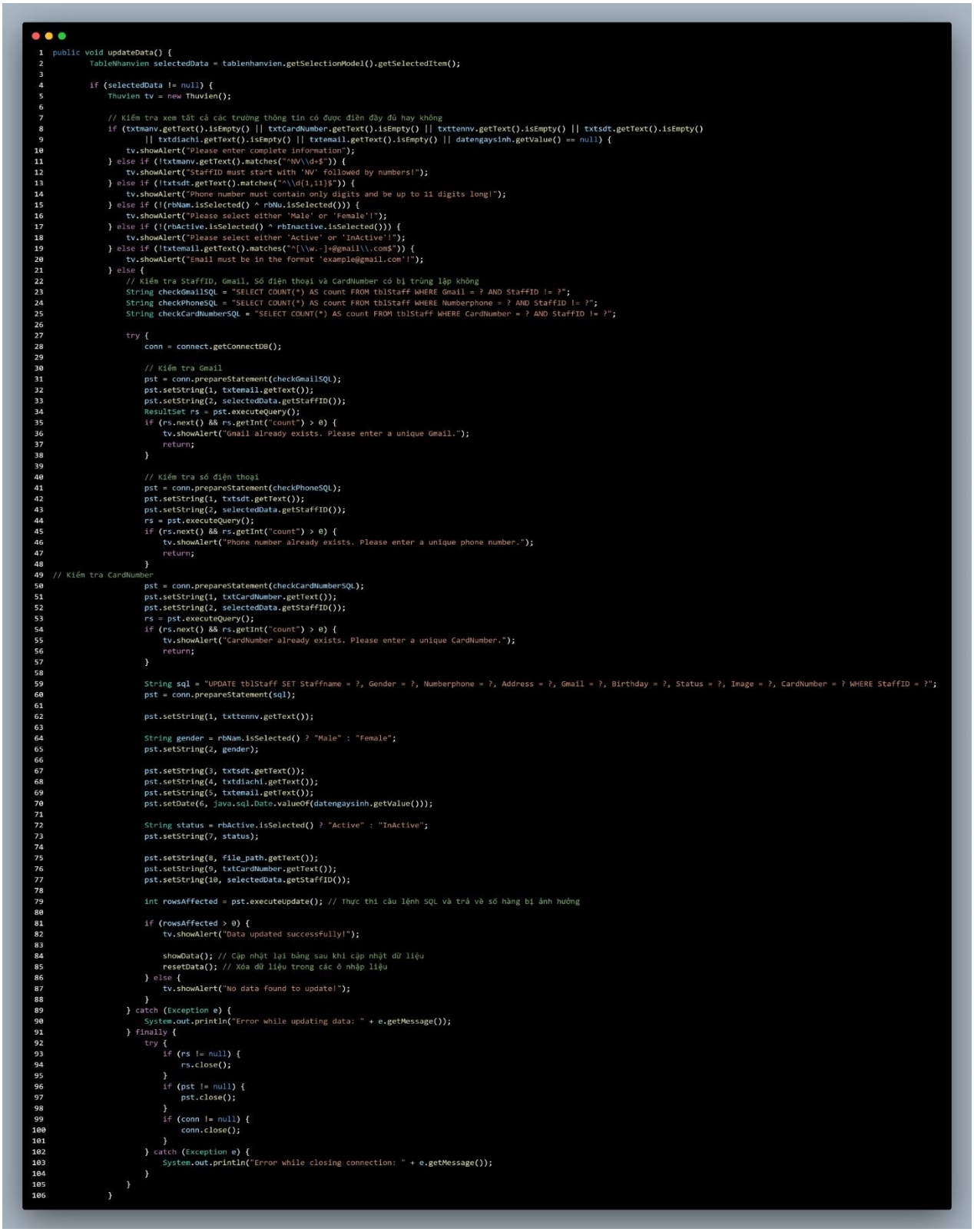
```
public class FXMLNhanVienController implements Initializable {

    private BillsController billsController;

    public void setBillsController(BillsController billsController) {
        this.billsController = billsController;
    }
    @FXML
    private FXMLMainController mainController;
    @FXML
    private AnchorPane top_main;

    @FXML
    private Button btnExport;

    @FXML
    private Button btnreset;

    @FXML
    private Button btnsua;

    @FXML
    private TextField txtCardNumber;

    @FXML
    private Button btnxoa;

    @FXML
    private RadioButton rbActive;

    @FXML
    private RadioButton rbInactive;

    @FXML
    private RadioButton rbNam;

    @FXML
    private RadioButton rbNu;

    @FXML
    private Button btnPayroll;

    @FXML
    private Button btnTimekeeping;


    @FXML
    private TableColumn<TableNhanvien, String> cotaddress;

    @FXML
    private TableColumn<TableNhanvien, Date> cotbirthday;

    @FXML
    private TableColumn<TableNhanvien, String> cotemail;

    @FXML
    private TableColumn<TableNhanvien, String> cotgender;

    @FXML
    private TableColumn<TableNhanvien, String> cotsdt;

    @FXML
    private TableColumn<TableNhanvien, String> cotstaffid;

    @FXML
    private TableColumn<TableNhanvien, String> cotstaffname;

    @FXML
    private TableColumn<TableNhanvien, String> cotstatus;

    @FXML
    private DatePicker datengaysinh;

    @FXML
    private ImageView imageview;

    @FXML
    private TableView<TableNhanvien> tablenhanvien;

    @FXML
    private TextField txtdiachi;

    @FXML
    private TextField txtemail;

    @FXML
    private TextField txtmanv;

    @FXML
    private TextField txtsdt;

    @FXML
    private TextField txttennv;
    @FXML
    private Label file_path;
    @FXML
    private TextField txttukhoa;

    /**
     * Initializes the controller class.
     */
    public ConnectDB connect = new ConnectDB();
    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;

    public ObservableList<TableNhanvien> dataList() {

```

19.1 Code

```
ObservableList<TableNhanvien> dataList = FXCollections.observableArrayList();

        String sql = "select * from tblStaff";

        try {
            conn = connect.getConnectDB();
            pst = conn.prepareStatement(sql);

            rs = pst.executeQuery();

            TableNhanvien data;

            while (rs.next()) {

                data = new TableNhanvien(rs.getString("StaffID"),
                        rs.getString("Staffname"),
                        rs.getString("Gender"),
                        rs.getString("Address"),
                        rs.getString("Numberphone"),
                        rs.getDate("Birthday"),
                        rs.getString("Gmail"),
                        rs.getString("Image"),
                        rs.getString("Status"),
                        rs.getInt("CardNumber"));
                dataList.add(data);

            }

        } catch (Exception e) {

            System.out.println(e.getMessage());

        }

        return dataList;
    }

    public void showData() {

        ObservableList<TableNhanvien> showList = dataList();

        cotstaffid.setCellValueFactory(new PropertyValueFactory<>("StaffID"));

        cotstaffname.setCellValueFactory(new PropertyValueFactory<>("Staffname"));

        cotgender.setCellValueFactory(new PropertyValueFactory<>("Gender"));

        cotsdt.setCellValueFactory(new PropertyValueFactory<>("Numberphone"));

        cotaddress.setCellValueFactory(new PropertyValueFactory<>("Address"));

        cotemail.setCellValueFactory(new PropertyValueFactory<>("Gmail"));

        cotbirthday.setCellValueFactory(new PropertyValueFactory<>("Birthday"));

        cotstatus.setCellValueFactory(new PropertyValueFactory<>("Status"));

        tablenhanvien.setItems(showList);

    }

    public void ThemStaff() {
        Thuvien tv = new Thuvien();

        // Kiểm tra xem tất cả các trường thông tin có được điền đầy đủ hay không
        if (txtmanv.getText().isEmpty() || txtCardNumber.getText().isEmpty() || txttennv.getText().isEmpty() || txtsdt.getText().isEmpty()
                || txtdiachi.getText().isEmpty() || txtemail.getText().isEmpty() || datengaysinh.getValue() == null) {
            tv.showAlert("Please enter complete information");
        } else if (!txtmanv.getText().matches("^NV\\d+$")) {
            tv.showAlert("StaffID must start with 'NV' followed by numbers!");
        } else if (!txtsdt.getText().matches("^\\d{1,11}$")) {
            tv.showAlert("Phone number must contain only digits and be up to 11 digits long!");
        } else if (!(rbNam.isSelected() ^ rbNu.isSelected())) {
            tv.showAlert("Please select either 'Male' or 'Female'!");
        } else if (!(rbActive.isSelected() ^ rbInactive.isSelected())) {
            tv.showAlert("Please select either 'Active' or 'InActive'!");
        } else if (!txtemail.getText().matches("^[\\w.-]+@gmail\\.com$")) {
            tv.showAlert("Email must be in the format 'example@gmail.com'!");
        } else {
            // Kiểm tra xem StaffID, Gmail, Số điện thoại và CardNumber có bị trùng lặp không
            String checkStaffIDSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE StaffID = ?";
            String checkGmailSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE Gmail = ?";
            String checkPhoneSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE Numberphone = ?";
            String checkCardNumberSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE CardNumber = ?";

            try {
                conn = connect.getConnectDB();

                // Kiểm tra StaffID
                pst = conn.prepareStatement(checkStaffIDSQL);
                pst.setString(1, txtmanv.getText());
                ResultSet rs = pst.executeQuery();
                if (rs.next() && rs.getInt("count") > 0) {
                    tv.showAlert("StaffID already exists. Please enter a unique StaffID.");
                    return;
                }

                // Kiểm tra Gmail
                pst = conn.prepareStatement(checkGmailSQL);
                pst.setString(1, txtemail.getText());
                rs = pst.executeQuery();
                if (rs.next() && rs.getInt("count") > 0) {
                    tv.showAlert("Gmail already exists. Please enter a unique Gmail.");
                    return;
                }

```

```
    // Kiểm tra số điện thoại
            pst = conn.prepareStatement(checkPhoneSQL);
            pst.setString(1, txtsdt.getText());
            rs = pst.executeQuery();
            if (rs.next() && rs.getInt("count") > 0) {
                tv.showAlert("Phone number already exists. Please enter a unique phone number.");
                return;
            }

            // Kiểm tra CardNumber
            pst = conn.prepareStatement(checkCardNumberSQL);
            pst.setString(1, txtCardNumber.getText());
            rs = pst.executeQuery();
            if (rs.next() && rs.getInt("count") > 0) {
                tv.showAlert("CardNumber already exists. Please enter a unique CardNumber.");
                return;
            }

        } catch (Exception e) {
            System.out.println("Error while checking data: " + e.getMessage());
            return;
        }

        // Nếu không có dữ liệu bị trùng lặp, tiếp tục thêm mới nhân viên
        String sql = "INSERT INTO tblStaff (StaffID, Staffname, Gender, Numberphone, Address, Gmail, Birthday, Status, Image, CardNumber) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
        try {
            pst = conn.prepareStatement(sql);

            pst.setString(1, txtmanv.getText()); // StaffID
            pst.setString(2, txttennv.getText()); // Staffname

            String gender = rbNam.isSelected() ? "Male" : "Female";
            pst.setString(3, gender); // Gender

            pst.setString(4, txtsdt.getText()); // Numberphone
            pst.setString(5, txtdiachi.getText()); // Address
            pst.setString(6, txtemail.getText()); // Gmail

            // Chuyển đổi giá trị DatePicker thành java.sql.Date
            pst.setDate(7, java.sql.Date.valueOf(datengaysinh.getValue())); // Birthday

            String status = rbActive.isSelected() ? "Active" : "InActive";
            pst.setString(8, status); // Status

            pst.setString(9, file_path.getText()); // Image, bạn có thể thay đổi giá trị này để lấy đường dẫn thực tế từ btnchonhinh

            pst.setString(10, txtCardNumber.getText()); // CardNumber

            pst.executeUpdate(); // Thực thi câu lệnh SQL
            tv.showAlert("Staff added successfully!");

            // Cập nhật lại bảng sau khi thêm mới nhân viên
            showData();
            // Reset các trường dữ liệu sau khi thêm nhân viên
            resetData();

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

public void insertImage() {
    FileChooser open = new FileChooser();
    Stage stage = (Stage) top_main.getScene().getWindow();
    File file = open.showOpenDialog(stage);
    if (file != null) {
        String path = file.getAbsolutePath();
        path = path.replace("\\", "\\\\");
        file_path.setText(path);
        Image image = new Image(file.toURI().toString(), 110, 110, false, true);
        imageview.setImage(image);

    } else {

        System.out.println("NO DATA EXIST!");
    }
}

//load du lieu len khi nhan vo tung dòng
public void selectData() {
    TableNhanvien data = tablenhanvien.getSelectionModel().getSelectedItem();
    int num = tablenhanvien.getSelectionModel().getSelectedIndex();
    if (num >= 0) {
        txtmanv.setText(data.getStaffID());
        txtmanv.setEditable(false); // Vô hiệu hóa chỉnh sửa
        txttennv.setText(data.getStaffname());
        txtdiachi.setText(data.getAddress());
        txtsdt.setText(data.getNumberphone());
        txtemail.setText(data.getGmail());
        // Set CardNumber
        txtCardNumber.setText(String.valueOf(data.getCardNumber()));

        // Set Gender RadioButtons
        if (data.getGender().equalsIgnoreCase("Male")) {
            rbNam.setSelected(true);
            rbNu.setSelected(false);
        } else if (data.getGender().equalsIgnoreCase("Female")) {
            rbNam.setSelected(false);
            rbNu.setSelected(true);
        }

        // Set DatePicker value without using toLocalDate()
        if (data.getBirthday() != null) {
            java.sql.Date sqlDate = (java.sql.Date) data.getBirthday();
            LocalDate localDate = LocalDate.of(sqlDate.getYear() + 1900, sqlDate.getMonth() + 1, sqlDate.getDate());
            datengaysinh.setValue(localDate);
        } else {
            datengaysinh.setValue(null);
        }

```

```java
// Set Status RadioButtons
        if (data.getStatus().equalsIgnoreCase("Active")) {
            rbActive.setSelected(true);
            rbInactive.setSelected(false);
        } else if (data.getStatus().equalsIgnoreCase("InActive")) {
            rbActive.setSelected(false);
            rbInactive.setSelected(true);
        }
        String picture = "file:" + data.getImage();
        Image image = new Image(picture, 110, 110, false, true);
        imageview.setImage(image);
        String path = data.getImage();
        file_path.setText(path);
    }
}

public void resetData() {
    txtmanv.clear();
    txttennv.clear();
    txtdiachi.clear();
    txtsdt.clear();
    txtemail.clear();
    txtCardNumber.clear();
    txtmanv.setEditable(true);
    datengaysinh.setValue(null);
    rbNam.setSelected(false);
    rbNu.setSelected(false);
    rbActive.setSelected(false);
    rbInactive.setSelected(false);
    imageview.setImage(null);
    file_path.setText("");
}

public void deleteData() {
    Thuvien tv = new Thuvien();
    // Lấy dữ liệu đã chọn từ bảng TableView
    TableNhanvien selectedData = tablenhanvien.getSelectionModel().getSelectedItem();
    if (selectedData != null) { // Kiểm tra xem có hàng nào được chọn hay không
        String staffID = selectedData.getStaffID(); // Lấy StaffID từ dữ liệu đã chọn

        // Hiển thị thông báo xác nhận
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Confirmation Dialog");
        alert.setHeaderText(null);
        alert.setContentText("Are you sure you want to delete this employee's information?");

        // Thêm các nút OK và Cancel vào thông báo
        ButtonType buttonTypeOK = new ButtonType("OK");
        ButtonType buttonTypeCancel = new ButtonType("Cancel", ButtonBar.ButtonData.CANCEL_CLOSE);
        alert.getButtonTypes().setAll(buttonTypeOK, buttonTypeCancel);

        // Lấy kết quả từ thông báo
        Optional<ButtonType> result = alert.showAndWait();
        if (result.isPresent() && result.get() == buttonTypeOK) {
            // Câu lệnh SQL để kiểm tra xem nhân viên có dữ liệu trong tblPayroll hay không
            String sqlCheck = "SELECT * FROM tblPayroll WHERE StaffID = ?";

            try {
                conn = connect.getConnectDB(); // Kết nối tới cơ sở dữ liệu
                pst = conn.prepareStatement(sqlCheck); // Chuẩn bị câu lệnh SQL
                pst.setString(1, staffID); // Thiết lập giá trị cho tham số đầu tiên trong câu lệnh SQL

                rs = pst.executeQuery(); // Thực thi câu lệnh SQL và nhận kết quả trả về

                if (rs.next()) { // Nếu có dữ liệu trong tblPayroll
                    tv.showAlert("Cannot delete staff. Please delete payroll data first."); // Thông báo không thể xóa vì chưa xóa bảng lương
                } else {
                    // Câu lệnh SQL để xóa dữ liệu từ bảng tblStaff dựa trên StaffID
                    String sqlDeleteStaff = "DELETE FROM tblStaff WHERE StaffID = ?";

                    pst = conn.prepareStatement(sqlDeleteStaff); // Chuẩn bị câu lệnh SQL để xóa nhân viên
                    pst.setString(1, staffID); // Thiết lập giá trị cho tham số đầu tiên trong câu lệnh SQL

                    int rowsAffected = pst.executeUpdate(); // Thực thi câu lệnh SQL và trả về số hàng bị ảnh hưởng

                    if (rowsAffected > 0) { // Kiểm tra xem có hàng nào bị xóa không
                        tv.showAlert("Data deleted successfully!"); // Thông báo xóa thành công

                        showData(); // Cập nhật lại bảng sau khi xóa
                        resetData(); // Xóa dữ liệu trong các ô nhập liệu
                    } else {
                        tv.showAlert("No data found to delete!"); // Thông báo không tìm thấy dữ liệu để xóa
                    }
                }
            } catch (Exception e) { // Bắt lỗi nếu có
                System.out.println("Error while deleting data: " + e.getMessage()); // Thông báo lỗi
            } finally {
                try {
                    if (rs != null) {
                        rs.close();
                    }
                    if (pst != null) {
                        pst.close();
                    }
                    if (conn != null) {
                        conn.close();
                    }
                } catch (Exception ex) {
                    System.out.println("Error while closing resources: " + ex.getMessage());
                }
            }
        } else {
            tv.showAlert("Delete operation cancelled."); // Thông báo nếu người dùng hủy bỏ thao tác xóa
        }
    } else {
        tv.showAlert("Please select a row to delete."); // Thông báo nếu không có hàng nào được chọn
    }
     txtmanv.setEditable(true);

}
```



```
public void updateData() {
        TableNhanvien selectedData = tablenhanvien.getSelectionModel().getSelectedItem();

        if (selectedData != null) {
            Thuvien tv = new Thuvien();

            // Kiểm tra xem tất cả các trường thông tin có được điền đầy đủ hay không
            if (txtmanv.getText().isEmpty() || txtCardNumber.getText().isEmpty() || txttennv.getText().isEmpty() || txtsdt.getText().isEmpty()
                    || txtdiachi.getText().isEmpty() || txtemail.getText().isEmpty() || datengaysinh.getValue() == null) {
                tv.showAlert("Please enter complete information");
            } else if (!txtmanv.getText().matches("^NV\\d+$")) {
                tv.showAlert("StaffID must start with 'NV' followed by numbers!");
            } else if (!txtsdt.getText().matches("^\\d{1,11}$")) {
                tv.showAlert("Phone number must contain only digits and be up to 11 digits long!");
            } else if (!(rbNam.isSelected() ^ rbNu.isSelected())) {
                tv.showAlert("Please select either 'Male' or 'Female'!");
            } else if (!(rbActive.isSelected() ^ rbInactive.isSelected())) {
                tv.showAlert("Please select either 'Active' or 'InActive'!");
            } else if (!txtemail.getText().matches("^[\\w.-]+@gmail\\.com$")) {
                tv.showAlert("Email must be in the format 'example@gmail.com'!");
            } else {
                // Kiểm tra StaffID, Gmail, Số điện thoại và CardNumber có bị trùng lặp không
                String checkGmailSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE Gmail = ? AND StaffID != ?";
                String checkPhoneSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE Numberphone = ? AND StaffID != ?";
                String checkCardNumberSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE CardNumber = ? AND StaffID != ?";

                try {
                    conn = connect.getConnectDB();

                    // Kiểm tra Gmail
                    pst = conn.prepareStatement(checkGmailSQL);
                    pst.setString(1, txtemail.getText());
                    pst.setString(2, selectedData.getStaffID());
                    ResultSet rs = pst.executeQuery();
                    if (rs.next() && rs.getInt("count") > 0) {
                        tv.showAlert("Gmail already exists. Please enter a unique Gmail.");
                        return;
                    }

                    // Kiểm tra số điện thoại
                    pst = conn.prepareStatement(checkPhoneSQL);
                    pst.setString(1, txtsdt.getText());
                    pst.setString(2, selectedData.getStaffID());
                    rs = pst.executeQuery();
                    if (rs.next() && rs.getInt("count") > 0) {
                        tv.showAlert("Phone number already exists. Please enter a unique phone number.");
                        return;
                    }
// Kiểm tra CardNumber
                    pst = conn.prepareStatement(checkCardNumberSQL);
                    pst.setString(1, txtCardNumber.getText());
                    pst.setString(2, selectedData.getStaffID());
                    rs = pst.executeQuery();
                    if (rs.next() && rs.getInt("count") > 0) {
                        tv.showAlert("CardNumber already exists. Please enter a unique CardNumber.");
                        return;
                    }

                    String sql = "UPDATE tblStaff SET Staffname = ?, Gender = ?, Numberphone = ?, Address = ?, Gmail = ?, Birthday = ?, Status = ?, Image = ?, CardNumber = ? WHERE StaffID = ?";
                    pst = conn.prepareStatement(sql);

                    pst.setString(1, txttennv.getText());

                    String gender = rbNam.isSelected() ? "Male" : "Female";
                    pst.setString(2, gender);

                    pst.setString(3, txtsdt.getText());
                    pst.setString(4, txtdiachi.getText());
                    pst.setString(5, txtemail.getText());
                    pst.setDate(6, java.sql.Date.valueOf(datengaysinh.getValue()));

                    String status = rbActive.isSelected() ? "Active" : "InActive";
                    pst.setString(7, status);

                    pst.setString(8, file_path.getText());
                    pst.setString(9, txtCardNumber.getText());
                    pst.setString(10, selectedData.getStaffID());

                    int rowsAffected = pst.executeUpdate(); // Thực thi câu lệnh SQL và trả về số hàng bị ảnh hưởng

                    if (rowsAffected > 0) {
                        tv.showAlert("Data updated successfully!");

                        showData(); // Cập nhật lại bảng sau khi cập nhật dữ liệu
                        resetData(); // Xóa dữ liệu trong các ô nhập liệu
                    } else {
                        tv.showAlert("No data found to update!");
                    }
                } catch (Exception e) {
                    System.out.println("Error while updating data: " + e.getMessage());
                } finally {
                    try {
                        if (rs != null) {
                            rs.close();
                        }
                        if (pst != null) {
                            pst.close();
                        }
                        if (conn != null) {
                            conn.close();
                        }
                    } catch (Exception e) {
                        System.out.println("Error while closing connection: " + e.getMessage());
                    }
                }
            }
```

```
    } else {
            Thuvien tv = new Thuvien();
            tv.showAlert("Please select a row to update.");
        }
    }

    public void searchStaff() {
        String keyword = txttukhoa.getText().toLowerCase();

        ObservableList<TableNhanvien> filteredList = FXCollections.observableArrayList();

        for (TableNhanvien staff : dataList()) {
            if (staff.getStaffID().toLowerCase().contains(keyword)
                    || staff.getStaffname().toLowerCase().contains(keyword)) {
                filteredList.add(staff);
            }
        }

        tablenhanvien.setItems(filteredList);
    }

    //chức năng xuất dữ liệu ra excel bằng nút Export
    public void exportToCSV() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save CSV File");
        fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "*.csv"));
        File selectedFile = fileChooser.showSaveDialog(btnExport.getScene().getWindow());

        if (selectedFile != null) {
            try {
                Writer writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(selectedFile), StandardCharsets.UTF_8));
                writer.write('\uFEFF'); // BOM for UTF-8

                // Ghi tiêu đề cột
                TableView<TableNhanvien> tableView = tablenhanvien;
                ObservableList<TableColumn<TableNhanvien, ?>> columns = tableView.getColumns();
                for (int i = 0; i < columns.size(); i++) {
                    TableColumn<TableNhanvien, ?> column = columns.get(i);
                    if (i > 0) {
                        writer.write(",");
                    }
                    writer.write(column.getText());
                }
                writer.write("\n");

                // Ghi dữ liệu từng dòng
                ObservableList<TableNhanvien> dataList = tableView.getItems();
                for (TableNhanvien staff : dataList) {
                    for (int i = 0; i < columns.size(); i++) {
                        TableColumn<TableNhanvien, ?> column = columns.get(i);
                        if (i > 0) {
                            writer.write(",");
                        }
                        // Lấy giá trị từ cell của bảng và ghi vào writer
                        Object cellValue = column.getCellData(staff);
                        writer.write(cellValue != null ? cellValue.toString() : "");
                    }
                    writer.write("\n");
                }
                writer.flush();
                writer.close();

                Thuvien tv = new Thuvien(); // Thêm dòng này nếu Thuvien cần được sử dụng
                tv.showAlert("Data exported to CSV successfully!");

                // Mở file CSV sau khi export
                Desktop.getDesktop().open(selectedFile);

            } catch (IOException e) {
                System.out.println("Error while exporting to CSV: " + e.getMessage());
            }
        }
    }

    public void setMainController(FXMLMainController mainController) {
        this.mainController = mainController;
    }

    //chuyen form bang luong
    @FXML
    private void goToChamluongForm() {
        if (mainController != null) {
            mainController.loadFXML("/demoproject1/FXMLChamluong.fxml");
        } else {
            System.err.println("FXMLNhanVienController: MainController is null, cannot load FXMLChamluong.fxml");
        }
    }

    @FXML
    private void goToBangluongForm() {
        if (mainController != null) {
            mainController.loadFXML("/demoproject1/FXMLBangluong.fxml");
        } else {
            System.err.println("FXMLNhanVienController: MainController is null, cannot load FXMLBangluong.fxml");
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        // Các phần khởi tạo khác
        btnreset.setOnAction(event -> resetData());
        btnxoa.setOnAction(event -> deleteData());
        btnsua.setOnAction(event -> updateData());
        btnExport.setOnAction(event -> exportToCSV()); //xuất ra excel
        showData();

        // Gọi phương thức tìm kiếm khi người dùng nhập từ khóa
        txttukhoa.textProperty().addListener((observable, oldValue, newValue) -> {
            searchStaff();
        });

    }
}
```

20 Add Payroll

```java
public class FXMLThemPayrollController implements Initializable {

    @FXML
    private Button btncancel;

    @FXML
    private Button btnAddpayroll;

    @FXML
    private DatePicker datePayend;

    @FXML
    private TextField txtsalary;

    @FXML
    private DatePicker datePaystart;

    @FXML
    private TextField txtStaffID;

    @FXML
    private TextField txtstandardday;

    @FXML
    private TextField txttotalworkday;

    @FXML
    private TextField txtovertime;

    /**
     * Initializes the controller class.
     */
    public ConnectDB connect = new ConnectDB();
    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;

    @FXML
    public void handleCancel() {
        // Lấy Stage hiện tại từ Button
        Stage stage = (Stage) btncancel.getScene().getWindow();

        // Đóng Stage (form) hiện tại
        stage.close();
    }

    @FXML
    public void handleAddPayroll() {
        Thuvien tv = new Thuvien();

        // Kiểm tra xem tất cả các trường thông tin có được điền đầy đủ hay không
        if (txtStaffID.getText().isEmpty() || datePaystart.getValue() == null || datePayend.getValue() == null
                || txtstandardday.getText().isEmpty() || txttotalworkday.getText().isEmpty() || txtovertime.getText().isEmpty() || txtsalary.getText().isEmpty()) {
            tv.showAlert("Please enter complete information");
        } else {
            try {
                // Thiết lập kết nối cơ sở dữ liệu trước khi thực hiện bất kỳ thao tác nào
                conn = connect.getConnectDB();
                if (conn == null) {
                    return;
                }

                // Kiểm tra xem StaffID có tồn tại trong tblStaff không
                String checkStaffSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE StaffID = ?";
                try (PreparedStatement checkStmt = conn.prepareStatement(checkStaffSQL)) {
                    checkStmt.setString(1, txtStaffID.getText());
                    ResultSet rs = checkStmt.executeQuery();
                    if (rs.next() && rs.getInt("count") == 0) {
                        tv.showAlert("StaffID does not exist in tblStaff. Please enter a valid StaffID.");
                        return; // Thoát phương thức nếu StaffID không tồn tại
                    }
                } catch (Exception e) {
                    System.out.println("Error while checking StaffID: " + e.getMessage());
                    return;
                }

                // Chuỗi truy vấn SQL để thêm bảng lương vào tblPayroll
                String sql = "INSERT INTO tblPayroll (StaffID, PayPeriodStart, PayPeriodEnd, StandardDays, StandardHours, TotalWorkDays, Overtime, TotalPay, Salary) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
                try {
                    pst = conn.prepareStatement(sql);

                    pst.setString(1, txtStaffID.getText()); // StaffID

                    // Chuyển đổi giá trị DatePicker thành java.sql.Date
                    pst.setDate(2, java.sql.Date.valueOf(datePaystart.getValue())); // PayPeriodStart
                    pst.setDate(3, java.sql.Date.valueOf(datePayend.getValue())); // PayPeriodEnd
```

20.1 Code

```
    pst.setInt(4, Integer.parseInt(txtstandardday.getText())); // StandardDays

                    // Thiết lập giá trị mặc định cho StandardHours là 8
                    int defaultStandardHours = 8;
                    pst.setInt(5, defaultStandardHours); // StandardHours

                    // Thiết lập giá trị TotalWorkDays từ người dùng nhập vào
                    int totalWorkDays = Integer.parseInt(txttotalworkday.getText());
                    pst.setInt(6, totalWorkDays); // TotalWorkDays

                    // Thiết lập giá trị Overtime từ người dùng nhập vào
                    BigDecimal overtime = new BigDecimal(txtovertime.getText());
                    pst.setBigDecimal(7, overtime); // Overtime

                    // Thiết lập giá trị Salary từ người dùng nhập vào
                    BigDecimal salary = new BigDecimal(txtsalary.getText());
                    pst.setBigDecimal(9, salary); // Salary

                    // Tính toán và thiết lập giá trị cho TotalPay
                    BigDecimal totalPay = calculateTotalPay(salary, Integer.parseInt(txtstandardday.getText()), defaultStandardHours, totalWorkDays, overtime);
                    pst.setBigDecimal(8, totalPay); // TotalPay

                    // Thực thi câu lệnh SQL
                    pst.executeUpdate();
                    tv.showAlert("Payroll added successfully!");

                    // Đóng cửa sổ hiện tại của FXMLThemPayrollController
                    Stage stage = (Stage) btnAddpayroll.getScene().getWindow();
                    stage.close();

                    // Cập nhật lại bảng sau khi thêm mới Payroll
                    FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLBangluong.fxml"));
                    Parent parent = loader.load();
                    FXMLBangluongController controller = loader.getController();
                    controller.showDataBangluong(); // Gọi phương thức cập nhật bảng

                } catch (Exception e) {
                    System.out.println("Error while adding payroll: " + e.getMessage());
                }
            } catch (Exception ex) {
                System.out.println("Error while establishing database connection: " + ex.getMessage());
            }
        }
    }

    private BigDecimal calculateTotalPay(BigDecimal salary, int standardDays, int standardHours, int totalWorkDays, BigDecimal overtimeHours) {
        // Tính số tiền làm được trong 1 giờ
        BigDecimal hourlyRate = salary.divide(BigDecimal.valueOf(standardDays), BigDecimal.ROUND_HALF_UP)
                .divide(BigDecimal.valueOf(standardHours), BigDecimal.ROUND_HALF_UP);

        // Tính tổng số giờ làm việc trong tháng
        int totalActualHours = totalWorkDays * standardHours;

        // Tính toán lương từ số giờ làm việc thực tế và số giờ tăng ca
        BigDecimal totalPay = hourlyRate.multiply(BigDecimal.valueOf(totalActualHours))
                .add(hourlyRate.multiply(overtimeHours));

        return totalPay;
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        btncancel.setOnAction(event -> handleCancel());
        btnAddpayroll.setOnAction(event -> handleAddPayroll());
    }
}
```

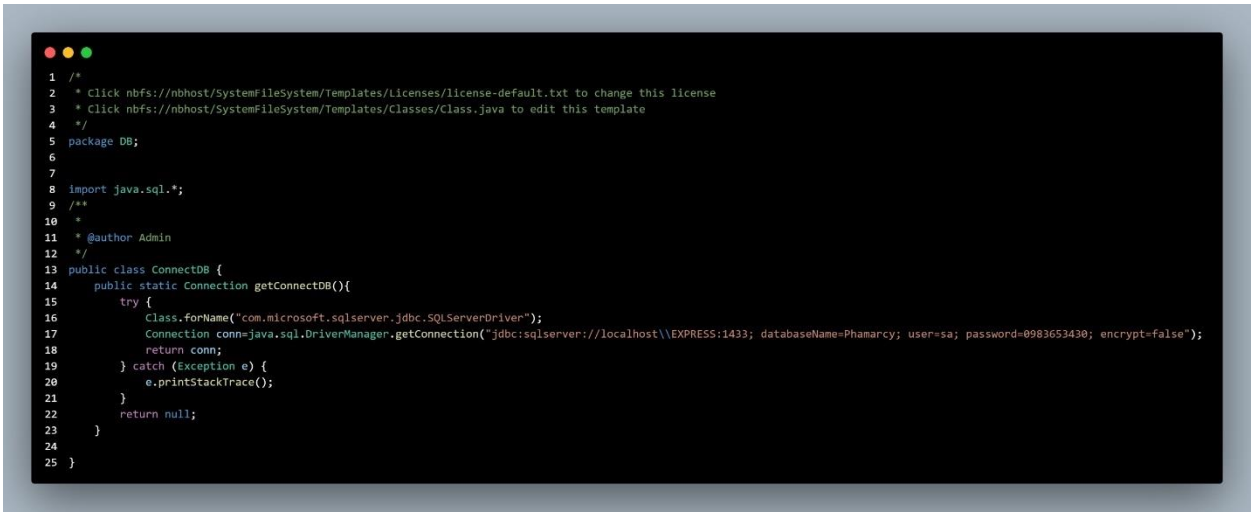
21 Update Payroll

```java
public class FXMLUpdatePayrollController implements initializable {

    @FXML
    private Button btnUpdatepayroll; // Nút để cập nhật thông tin lương

    @FXML
    private Button btncancel; // Nút để hủy bỏ và đóng form

    @FXML
    private DatePicker datePayend; // DatePicker cho ngày kết thúc kỳ trả lương

    @FXML
    private DatePicker datePaystart; // DatePicker cho ngày bắt đầu kỳ trả lương

    @FXML
    private TextField txtStaffID; // TextField cho ID nhân viên

    @FXML
    private TextField txtstandardday; // TextField cho số ngày tiêu chuẩn

    @FXML
    private TextField txttotalworkday;

    @FXML
    private TextField txtsalary;

    @FXML
    private TextField txtovertime;

    public ConnectDB connect = new ConnectDB(); // Kết nối cơ sở dữ liệu
    private Connection conn; // Kết nối SQL
    private PreparedStatement pst; // Câu lệnh SQL đã chuẩn bị
    private ResultSet rs; // Kết quả trả về từ câu lệnh SQL

    @FXML
    public void handleCancel() {
        Stage stage = (Stage) btncancel.getScene().getWindow();
        stage.close();
    }

    private TableBangluong payrollData; // Lưu trữ dữ liệu từ bảng lương

    private FXMLBangluongController parentController; // Tham chiếu đến controller cha để làm mới bảng

    public void setParentController(FXMLBangluongController parentController) {
        this.parentController = parentController;
    }

    public void setUpdatePayroll(TableBangluong data) {
        payrollData = data; // Lưu dữ liệu từ dòng được chọn
        txtStaffID.setText(data.getStaffID());
        txtStaffID.setEditable(false); // Không cho phép chỉnh sửa trường Staff ID

        java.util.Date payPeriodStartDate = new java.util.Date(data.getPayPeriodStart().getTime());
        java.util.Date payPeriodEndDate = new java.util.Date(data.getPayPeriodEnd().getTime());

        datePaystart.setValue(payPeriodStartDate.toInstant().atZone(ZoneId.systemDefault()).toLocalDate());
        datePayend.setValue(payPeriodEndDate.toInstant().atZone(ZoneId.systemDefault()).toLocalDate());

        txtstandardday.setText(String.valueOf(data.getStandardDays()));
        txttotalworkday.setText(String.valueOf(data.getTotalWorkDays()));
        txtsalary.setText(String.valueOf(data.getSalary()));
        txtovertime.setText(String.valueOf(data.getOvertime()));
    }

    @FXML
    public void handleUpdate() {
        if (txtStaffID.getText().isEmpty() || datePaystart.getValue() == null || datePayend.getValue() == null
                || txtstandardday.getText().isEmpty() || txttotalworkday.getText().isEmpty() || txtovertime.getText().isEmpty() || txtsalary.getText().isEmpty()) {
            showAlert("Please enter complete information");
        } else {
            try {
                conn = connect.getConnectDB();
                if (conn == null) {
                    return;
                }

                String sql = "UPDATE tblPayroll SET PayPeriodStart = ?, PayPeriodEnd = ?, StandardDays = ?, StandardHours = ?, TotalWorkDays = ?, Overtime = ?, TotalPay = ?, Salary = ? WHERE StaffID = ?";

                pst = conn.prepareStatement(sql);

                pst.setDate(1, java.sql.Date.valueOf(datePaystart.getValue())); // PayPeriodStart
                pst.setDate(2, java.sql.Date.valueOf(datePayend.getValue())); // PayPeriodEnd

                int standardDays = Integer.parseInt(txtstandardday.getText());
                pst.setInt(3, standardDays); // StandardDays

                int defaultStandardHours = 8;
                pst.setInt(4, defaultStandardHours); // StandardHours

                BigDecimal totalWorkDays = new BigDecimal(txttotalworkday.getText());
                pst.setBigDecimal(5, totalWorkDays); // TotalWorkDays

                BigDecimal overtime = new BigDecimal(txtovertime.getText());
                pst.setBigDecimal(6, overtime); // Overtime

```

21.1 Code

```
                BigDecimal salary = new BigDecimal(txtsalary.getText());
                pst.setBigDecimal(8, salary); // Salary

                BigDecimal totalPay = calculateTotalPay(salary, standardDays, defaultStandardHours, totalWorkDays, overtime);
                pst.setBigDecimal(7, totalPay); // TotalPay

                pst.setString(9, txtStaffID.getText()); // StaffID

                int rowsAffected = pst.executeUpdate();
                if (rowsAffected > 0) {
                    showAlert("Payroll data updated successfully.");

                    payrollData.setPayPeriodStart(java.sql.Date.valueOf(datePaystart.getValue()));
                    payrollData.setPayPeriodEnd(java.sql.Date.valueOf(datePayend.getValue()));
                    payrollData.setStandardDays(standardDays);
                    payrollData.setTotalWorkDays(totalWorkDays);
                    payrollData.setOvertime(overtime);
                    payrollData.setSalary(salary);
                    payrollData.setTotalPay(totalPay);

                    if (parentController != null) {
                        parentController.showDataBangluong();
                    }

                    Stage stage = (Stage) btnUpdatepayroll.getScene().getWindow();
                    stage.close();
                }
            } catch (SQLException e) {
                System.out.println("Error while updating payroll: " + e.getMessage());
            } finally {
                closeResources();
            }
        }
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        btnUpdatepayroll.setOnAction(event -> handleUpdate());
        btncancel.setOnAction(event -> handleCancel());
    }

    private void showAlert(String message) {
        Alert alert = new Alert(Alert.AlertType.INFORMATION);
        alert.setTitle("Information");
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

    private BigDecimal calculateTotalPay(BigDecimal salary, int standardDays, int standardHours, BigDecimal totalWorkDays, BigDecimal overtimeHours) {
        // Số tiền làm được trong 1 giờ
        BigDecimal hourlyRate = salary.divide(BigDecimal.valueOf(standardDays), BigDecimal.ROUND_HALF_UP)
                .divide(BigDecimal.valueOf(standardHours), BigDecimal.ROUND_HALF_UP);

        // Tổng số giờ làm việc trong tháng
        BigDecimal totalActualHours = totalWorkDays.multiply(BigDecimal.valueOf(standardHours));

        // Tính toán lương từ số giờ làm việc thực tế và số giờ tăng ca
        BigDecimal totalPay = hourlyRate.multiply(totalActualHours)
                .add(hourlyRate.multiply(overtimeHours));

        return totalPay;
    }

    private void closeResources() {
        try {
            if (pst != null) {
                pst.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            System.out.println("Error while closing connection: " + e.getMessage());
        }
    }
}
```

22 Connect Database



```java
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package DB;


import java.sql.*;
/**
 *
 * @author Admin
 */
public class ConnectDB {
    public static Connection getConnectDB(){
        try {
            Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
            Connection conn=java.sql.DriverManager.getConnection("jdbc:sqlserver://localhost\\EXPRESS:1433; databaseName=Phamarcy; user=sa; password=0983653430; encrypt=false");
            return conn;
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

23 Account

```java
public class AccountInfoController implements Initializable {

        @FXML
    private TableColumn<AccountData, Void> Acc_Col_AC;

    @FXML
    private TableColumn<AccountData, String> Acc_Col_Address;

    @FXML
    private TableColumn<AccountData, LocalDateTime> Acc_Col_DateCreated;

    @FXML
    private TableColumn<AccountData, String> Acc_Col_Email;

    @FXML
    private TableColumn<AccountData, String> Acc_Col_FullName;

    @FXML
    private TableColumn<AccountData, String> Acc_Col_Password;

    @FXML
    private TableColumn<AccountData, String> Acc_Col_Phone;

    @FXML
    private TableColumn<AccountData, String> Acc_Col_Role;

    @FXML
    private Pagination pgtAL;

    @FXML
    private TextField searchAL;

    @FXML
    private TableView<AccountData> tableaccoutlist;

    private Connection conn;
    private PreparedStatement ps;
    private ResultSet rs;
    Message alert = new Message();

    public ObservableList<AccountData> DataList() {
        ObservableList<AccountData> Datalist = FXCollections.observableArrayList();
        String sql = "SELECT i.ID, i.Fullname, i.Phone, i.Email, i.Address, i.Date_Created, l.Password, r.Role " +
                "FROM Information i " +
                "JOIN Login l ON i.Phone = l.username " +
                "JOIN Role r ON l.username = r.UserName";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            rs = ps.executeQuery();
            AccountData data;
            while (rs.next()) {
                Timestamp timestamp = rs.getTimestamp("Date_Created");
                LocalDateTime datecreated = timestamp.toLocalDateTime();
                data = new AccountData(rs.getInt("ID"),
                        rs.getString("FullName"),
                        rs.getString("Email"),
                        rs.getString("Phone"),
                        rs.getString("Address"),
                        rs.getString("Password"),
                        rs.getString("Role"),
                        datecreated

                );
                Datalist.add(data);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                System.out.println("Error closing resources: " + e.getMessage());
            }

        }
        return Datalist;
    }

    public void showData() {
        ObservableList<AccountData> showList = DataList();

        Acc_Col_Address.setCellValueFactory(new PropertyValueFactory<>("Address"));
        Acc_Col_DateCreated.setCellValueFactory(new PropertyValueFactory<>("DateCreated"));
        Acc_Col_Email.setCellValueFactory(new PropertyValueFactory<>("Email"));
        Acc_Col_FullName.setCellValueFactory(new PropertyValueFactory<>("FullName"));
        Acc_Col_Password.setCellValueFactory(new PropertyValueFactory<>("Password"));
        Acc_Col_Phone.setCellValueFactory(new PropertyValueFactory<>("Phone"));
        Acc_Col_Role.setCellValueFactory(new PropertyValueFactory<>("Role"));
        Acc_Col_AC.setCellFactory(new Callback<TableColumn<AccountData, Void>, TableCell<AccountData, Void>>() {
            @Override
            public TableCell<AccountData, Void> call(final TableColumn<AccountData, Void> param) {
                final TableCell<AccountData, Void> cell = new TableCell<AccountData, Void>() {

                    private final Button btnUpdate = new Button("Update");
                    private final Button btnDelete = new Button("Delete");

```

23.1 Code

```
                    {
                        btnUpdate.getStyleClass().add("btn_update");
                        btnDelete.getStyleClass().add("btn_delete");
                        btnUpdate.setOnAction((ActionEvent event) -> {
                            AccountData pdata = getTableView().getItems().get(getIndex());
                            openUpdateForm(pdata);
                        });

                        btnDelete.setOnAction((ActionEvent event) -> {
                            AccountData pdata = getTableView().getItems().get(getIndex());
                            if (alert.confirmMessage("Are you sure want to delete User: " + pdata.getPhone() + "?")) {
                                deleteData(pdata.getPhone());
                                showData(); // Refresh table
                                Pagination();
                            }
                        });
                        btnUpdate.setAlignment(Pos.CENTER);
                        btnDelete.setAlignment(Pos.CENTER);
                    }

                    @Override
                    public void updateItem(Void item, boolean empty) {
                        super.updateItem(item, empty);
                        if (empty) {
                            setGraphic(null);
                        } else {
                            HBox hbox = new HBox(5);
                            hbox.getChildren().addAll(btnUpdate, btnDelete);
                            hbox.setAlignment(Pos.CENTER);
                            setGraphic(hbox);
                        }
                    }
                };
                return cell;
            }
        });

        tableaccoutlist.setItems(showList);
    }

    private void deleteData(String username) {
    String sqlDeleteRole = "DELETE FROM Role WHERE Username = ?";
    String sqlDeleteInformation = "DELETE FROM Information WHERE Phone = ?";
    String sqlDeleteLogin = "DELETE FROM Login WHERE Username = ?";

    try {
        conn = DB.ConnectDB.getConnectDB();

        // Bắt đầu giao dịch
        conn.setAutoCommit(false);

        // Xóa từ bảng Role
        ps = conn.prepareStatement(sqlDeleteRole);
        ps.setString(1, username);
        ps.executeUpdate();
        ps.close();

        // Xóa từ bảng Infomation
        ps = conn.prepareStatement(sqlDeleteInformation);
        ps.setString(1, username);
        ps.executeUpdate();
        ps.close();

        // Xóa từ bảng Login
        ps = conn.prepareStatement(sqlDeleteLogin);
        ps.setString(1, username);
        ps.executeUpdate();

        // Hoàn thành giao dịch
        conn.commit();

        alert.successMessage("Deleted Successfully");
    } catch (SQLException e) {
        e.printStackTrace();
        try {
            // Rollback giao dịch nếu có lỗi
            if (conn != null) {
                conn.rollback();
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    } finally {
        try {
            if (ps != null) {
                ps.close();
            }
            if (conn != null) {
                conn.close();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```java
private void openUpdateForm(AccountData pdata) {
        try {
            Data.al_ID = pdata.getID();
            Data.al_email = pdata.getEmail();
            Data.al_datecreated = pdata.getDateCreated();
            Data.al_role = pdata.getRole();
            Data.al_address = pdata.getAddress();
            Data.al_password = pdata.getPassword();
            Data.al_phone = pdata.getPhone();
            Data.al_fullname = pdata.getFullName();

            FXMLLoader loader = new FXMLLoader(getClass().getResource("/AccountInfo/UpdateInfo.fxml"));

            Parent root = loader.load();

            Stage stage = new Stage();
            stage.initStyle(StageStyle.UNDECORATED);
            stage.setScene(new Scene(root));
            stage.show();
        } catch (Exception e) {
            e.printStackTrace();
        }

    }

    private void filterData(String keyword) {
        ObservableList<AccountData> filteredList = FXCollections.observableArrayList();
        String sql = "SELECT i.ID, i.Fullname, i.Email, i.Phone, i.Address, l.Password, r.Role, i.Date_Created " +
                "FROM Information i " +
                "INNER JOIN Login l ON i.Phone = l.Username " + // Thêm khoảng trắng sau "l.Username"
                "INNER JOIN Role r ON l.Username = r.Username " + // Thêm khoảng trắng sau "r.Username"
                "WHERE i.Fullname LIKE ? OR i.Phone LIKE ? OR i.Email LIKE ?";
        try {
            conn = DB.ConnectDB.getConnectDB();
            ps = conn.prepareStatement(sql);
            String searchKeyword = "%" + keyword + "%";
            ps.setString(1, searchKeyword);
            ps.setString(2, searchKeyword);
            ps.setString(3, searchKeyword);
            rs = ps.executeQuery();

            while (rs.next()) {
                Timestamp timestamp = rs.getTimestamp("Date_Created");
                LocalDateTime datecreated = timestamp.toLocalDateTime();
                AccountData data = new AccountData(
                        rs.getInt("ID"),
                        rs.getString("FullName"),
                        rs.getString("Email"),
                        rs.getString("Phone"),
                        rs.getString("Address"),
                        rs.getString("Password"),
                        rs.getString("Role"),
                        datecreated
                );
                filteredList.add(data);
            }
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            try {
                if (rs != null) {
                    rs.close();
                }
                if (ps != null) {
                    ps.close();
                }
                if (conn != null) {
                    conn.close();
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        tableaccoutlist.setItems(filteredList);
    }

    public void Search() {
        searchAL.textProperty().addListener((observable, oldValue, newValue) -> {
            filterData(newValue);
        });
    }

    private static final int itemsperpage = 23;

    private int getPageCount() {
        return (int) Math.ceil((double) DataList().size() / itemsperpage);
    }

    private void updateTable(int pageIndex) {
        int fromIndex = pageIndex * itemsperpage;
        int toIndex = Math.min(fromIndex + itemsperpage, DataList().size());
        tableaccoutlist.setItems(FXCollections.observableArrayList(DataList().subList(fromIndex, toIndex)));
    }

    private Node createPage(int pageIndex) {
        updateTable(pageIndex);
        return tableaccoutlist;
    }

    public void Pagination() {
        pgtAL.setPageCount(getPageCount());
        pgtAL.setPageFactory(this::createPage);
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        ControllerHolder.getInstance().setAccountinfoController(this);
        showData();
        Pagination();
        Search();
    }

}
```

24 Update Account

```
public class UpdateInfoController implements Initializable {

      @FXML
    private Button btnCancelUpdate;

    @FXML
    private Button btnUpdate;

    @FXML
    private Button btnUpdateClose;

    @FXML
    private ComboBox<String> cbRole;

    @FXML
    private TextField txtAddress;

    @FXML
    private TextField txtFullName;

    @FXML
    private TextField txtPassword;

    @FXML
    private TextField txtPhone;

      @FXML
    private TextField txtEmail;

    @FXML
    void Cancelbtn(ActionEvent event) {
        updateDruglistForm();
        Stage stage = (Stage) btnCancelUpdate.getScene().getWindow();
        stage.close();
    }

    @FXML
    void Closebtn(ActionEvent event) {
        updateDruglistForm();
        Stage stage = (Stage) btnUpdateClose.getScene().getWindow();
        stage.close();
    }



    private Connection conn;
    private PreparedStatement ps;
    private ResultSet rs;
    Message alert = new Message();

    public void setField() {
        txtFullName.setText(Data.al_fullname);
        txtAddress.setText(Data.al_address);
        txtPhone.setText(Data.al_phone);
        txtPassword.setText(Data.al_password);
        cbRole.setValue(Data.al_role);
        txtEmail.setText(Data.al_email);
    }

    public class ValidateRegister {

        public static boolean isNotEmpty(String input) {
            return input != null && !input.trim().isEmpty();
        }


        public static boolean isNumeric(String input) {
            return input != null && input.matches("\\d+");
        }

        public static boolean isValidEmail(String email) {
            String emailRegex = "^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$";
            Pattern pat = Pattern.compile(emailRegex);
            return email != null && pat.matcher(email).matches();
        }
    }

    private boolean isPhoneAlreadyExists(String phone, int currentUserId) {
    try (Connection conn = DB.ConnectDB.getConnectDB()) {
        String sqlCheckPhone = "SELECT COUNT(*) FROM Information WHERE Phone = ? AND ID <> ?";
        try (PreparedStatement stmt = conn.prepareStatement(sqlCheckPhone)) {
            stmt.setString(1, phone);
            stmt.setInt(2, currentUserId);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    int count = rs.getInt(1);
                    return count > 0;
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}

    private boolean isEmailAlreadyExists(String email, int currentUserId) {
    try (Connection conn = DB.ConnectDB.getConnectDB()) {
        String sqlCheckEmail = "SELECT COUNT(*) FROM Information WHERE Email = ? AND ID <> ?";
        try (PreparedStatement stmt = conn.prepareStatement(sqlCheckEmail)) {
            stmt.setString(1, email);
            stmt.setInt(2, currentUserId);
            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    int count = rs.getInt(1);
                    return count > 0;
                }
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return false;
}
```

```java
public void updatebtn() {
        Integer ID = Data.al_ID;
        if (txtFullName.getText().isEmpty()
                || txtAddress.getText().isEmpty()
                || txtPhone.getText().isEmpty()
                || txtPassword.getText().isEmpty()
                || cbRole.getValue() == null
                || txtEmail.getText().isEmpty()) {
            alert.errorMessage("Please fill all blank fields");
        }
        else if (!ValidateRegister.isNumeric(txtPhone.getText())) {
            alert.errorMessage( "Phone must be a number");

        }
        if (isPhoneAlreadyExists(txtPhone.getText(), ID)) {
            alert.errorMessage( "Phone already exists in the database");

        }
         if (!ValidateRegister.isValidEmail(txtEmail.getText())) {
            alert.errorMessage( "Email is not valid");

        }

        if (isEmailAlreadyExists(txtEmail.getText(), ID)) {
            alert.errorMessage( "Email already exists in the database");

        }
            String fullname = txtFullName.getText();
            String address = txtAddress.getText();
            String email = txtEmail.getText();
            String password = txtPassword.getText();
            String phone = txtPhone.getText();
            String role = cbRole.getValue();

            try {
                if (alert.confirmMessage("Are you sure want to update User: " + phone + "?")) {
                    conn = DB.ConnectDB.getConnectDB();
                    String currentRmbPassword = getRmbPassword(phone);

            if (currentRmbPassword != null && !currentRmbPassword.trim().isEmpty()) {
                String sqlUpdateLogin = "UPDATE Login SET Password = ?, Username = ?, RmbPassword = ? WHERE Username = ?";
                try (PreparedStatement psLogin = conn.prepareStatement(sqlUpdateLogin)) {
                    psLogin.setString(1, password);
                    psLogin.setString(2, phone);
                    psLogin.setString(3, password);
                    psLogin.setString(4, Data.al_phone);
                    psLogin.executeUpdate();
                }
            } else {
                String sqlUpdateLogin = "UPDATE Login SET Password = ?, Username = ? WHERE Username = ?";
                try (PreparedStatement psLogin = conn.prepareStatement(sqlUpdateLogin)) {
                    psLogin.setString(1, password);
                    psLogin.setString(2, phone);
                    psLogin.setString(3, Data.al_phone);
                    psLogin.executeUpdate();
                }
            }
                    String sqlUpdateRole = "UPDATE Role SET Role = ?, Username=?  WHERE Username = ?";
            try (PreparedStatement psRole = conn.prepareStatement(sqlUpdateRole)) {
                psRole.setString(1, role);
                psRole.setString(2, phone);
                psRole.setString(3, Data.al_phone);
psRole.executeUpdate();
            }String sqlUpdateInformation = "UPDATE Information SET FullName = ?, Address = ?, Email = ?, Phone = ? WHERE Phone = ?";

                try(PreparedStatement psInfo = conn.prepareStatement(sqlUpdateInformation)){

                    ps = conn.prepareStatement(sqlUpdateInformation);
                    ps.setString(1, fullname);
                    ps.setString(2, address);
                    ps.setString(3, email);

                    ps.setString(4, phone);
                    ps.setString(5, Data.al_phone);


                    ps.executeUpdate();
                }


                    alert.successMessage("Update successfully");
                    Stage stage = (Stage) btnUpdate.getScene().getWindow();
                    stage.close();
                    updateDruglistForm();
                } else {
                    alert.errorMessage("Failed to update drug");
                }

            } catch (Exception e) {
                e.printStackTrace();
                alert.errorMessage("Error adding drug: " + e.getMessage());
            }
        }

    private String getRmbPassword(String username) {
    String rmbPassword = null;
    String sqlFetchRmbPassword = "SELECT RmbPassword FROM Login WHERE Username = ?";
    try (Connection conn = DB.ConnectDB.getConnectDB();
         PreparedStatement ps = conn.prepareStatement(sqlFetchRmbPassword)) {
        ps.setString(1, username);
        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                rmbPassword = rs.getString("rmbPassword");
            }
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return rmbPassword;
}

    private void updateDruglistForm() {

        AccountInfoController accountinfoController = ControllerHolder.getInstance().getAccountinfoController();
        if (accountinfoController != null) {
            accountinfoController.showData();
            accountinfoController.Pagination();
        }
    }

    public void RoleList() {
        List<String> ListS = new ArrayList<>();

        for (String data : Data.role) {
            ListS.add(data);
        }
        ObservableList ListData = FXCollections.observableArrayList(ListS);
        cbRole.setItems(ListData);
    }
    @Override
    public void initialize(URL url, ResourceBundle rb) {
        RoleList();
        setField();
    }

}

```

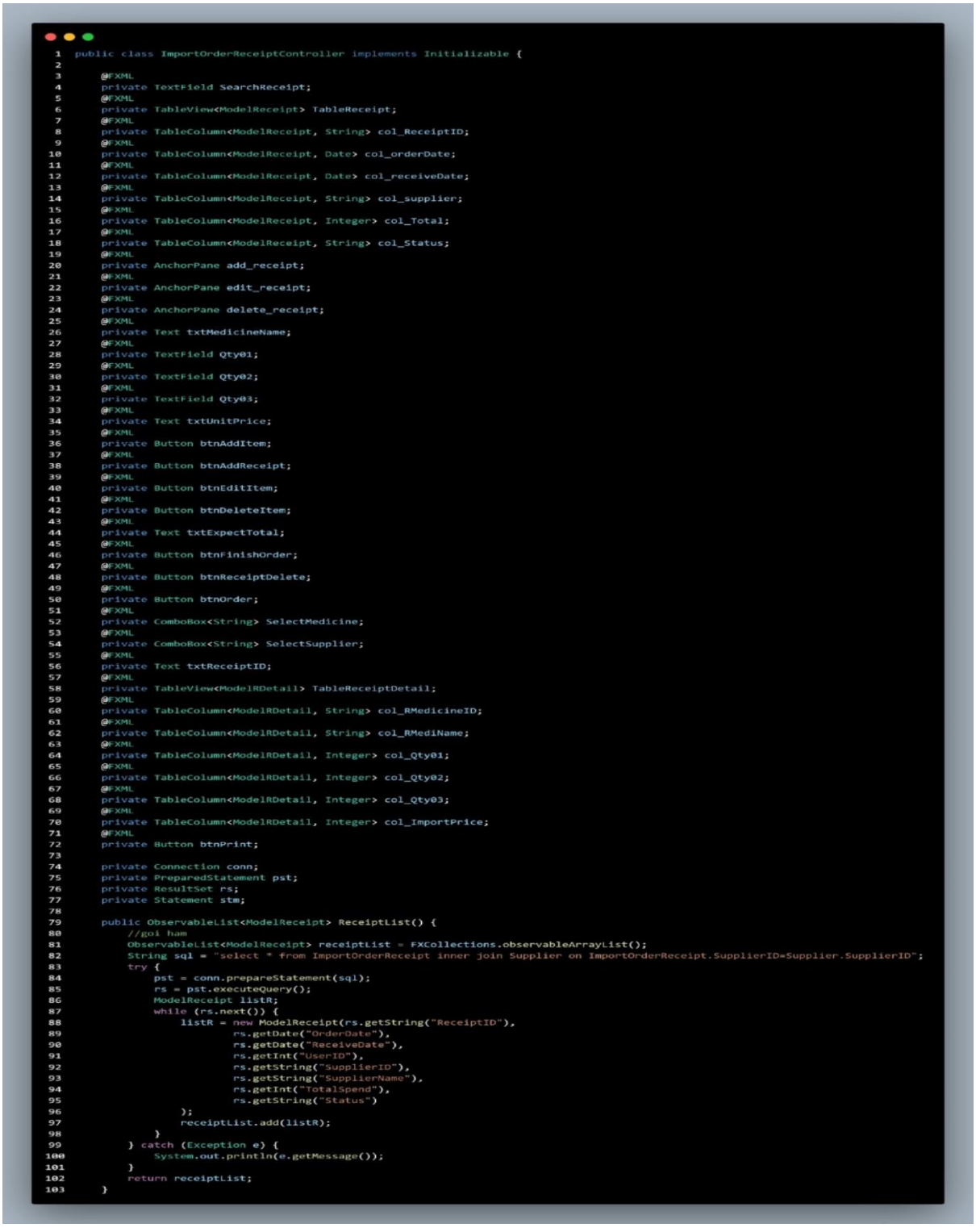
25 Add Supplier

```
public class AddsupplierController implements Initializable {

    @FXML
    private TextField txtSupplierName;
    @FXML
    private Button BtnAdd;
    @FXML
    private Button BtnCancle;
    @FXML
    private TextField txtPhone;
    @FXML
    private TextField txtAddress;
    @FXML
    private TextField txtEmail;
    @FXML
    private TextField txtSupplierID;

    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;
    private Statement stm;

    private SupplierController supplierController;

    private String SuppliertID;
    public String getHighestSupplierID() {
        String highestID = "S0000"; // Default value if there are no records
        String sql = "SELECT MAX(SupplierID) AS HighestID FROM Supplier";
        try {
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();
            if (rs.next()) {
                highestID = rs.getString("HighestID");
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return highestID;
    }

    public String generateNextSupplierID() {
    String highestID = getHighestSupplierID();
    int number = 1; // Mặc định bắt đầu từ số 1 nếu không có bản ghi nào tồn tại
    if (highestID != null && highestID.length() > 1) {
        number = Integer.parseInt(highestID.substring(1)) + 1;
    }
    return String.format("S%04d", number);
}

    private boolean validatePhone(String phone) {
        // Số điện thoại hợp lệ phải có 10 chữ số và bắt đầu bằng số 0
        return phone.matches("0[0-9]{9}");
    }

    private boolean validateEmail(String email) {
        // Email hợp lệ theo định dạng phổ biến
        return email.matches("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}$");
    }

    private boolean isPhoneExists(String phone) {
        String sql = "SELECT COUNT(*) FROM Supplier WHERE PhoneNumber = ?";
        try {
            pst = conn.prepareStatement(sql);
            pst.setString(1, phone);
            rs = pst.executeQuery();
            if (rs.next() && rs.getInt(1) > 0) {
                return true;
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return false;
    }

    private boolean isEmailExists(String email) {
        String sql = "SELECT COUNT(*) FROM Supplier WHERE Email = ?";
        try {
            pst = conn.prepareStatement(sql);
            pst.setString(1, email);
            rs = pst.executeQuery();
            if (rs.next() && rs.getInt(1) > 0) {
                return true;
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return false;
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        conn=DB.ConnectDB.getConnectDB();
        String newSupplierID = generateNextSupplierID();
        txtSupplierID.setText(newSupplierID);
        txtSupplierID.setEditable(false);
    }

    public void setSupplierController(SupplierController supplierController) {
        this.supplierController = supplierController;
    }

    @FXML
    private void btn_add(ActionEvent event) {
        if (!validatePhone(txtPhone.getText())) {
            showAlert("Invalid Phone Number", "Phone number must be 10 digits and start with 0.");
            return;
        }
```

codesnap.dev

```
if (!validateEmail(txtEmail.getText())) {
            showAlert("Invalid Email Address", "Please enter a valid email address.");
            return;
        }

        if (isPhoneExists(txtPhone.getText())) {
            showAlert("Phone Number Exists", "The phone number already exists. Please enter a different phone number.");
            return;
        }

        if (isEmailExists(txtEmail.getText())) {
            showAlert("Email Address Exists", "The email address already exists. Please enter a different email address.");
            return;
        }

        String sql = "INSERT INTO Supplier (SupplierID, SupplierName, PhoneNumber, Address, Email) VALUES (?, ?, ?, ?, ?)";
        try {
            pst = conn.prepareStatement(sql);
            pst.setString(1, txtSupplierID.getText());
            pst.setString(2, txtSupplierName.getText());
            pst.setString(3, txtPhone.getText());
            pst.setString(4, txtAddress.getText());
            pst.setString(5, txtEmail.getText());
            pst.executeUpdate();
            System.out.println("Supplier added successfully!");

            if (supplierController != null) {
                supplierController.showSList();;
            }

            Stage stage = (Stage) BtnCancle.getScene().getWindow();
            stage.close();

            } catch (Exception e) {
                System.out.println(e.getMessage());
            }
        }

        private void showAlert(String title, String message) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle(title);
            alert.setHeaderText(null);
            alert.setContentText(message);
            alert.showAndWait();
    }

    @FXML
    private void btn_cancle(ActionEvent event) {
        Stage stage = (Stage) BtnCancle.getScene().getWindow();
        stage.close();
    }

}

```

26 Import Order



```java
public class ImportOrderReceiptController implements Initializable {

    @FXML
    private TextField SearchReceipt;
    @FXML
    private TableView<ModelReceipt> TableReceipt;
    @FXML
    private TableColumn<ModelReceipt, String> col_ReceiptID;
    @FXML
    private TableColumn<ModelReceipt, Date> col_orderDate;
    @FXML
    private TableColumn<ModelReceipt, Date> col_receiveDate;
    @FXML
    private TableColumn<ModelReceipt, String> col_supplier;
    @FXML
    private TableColumn<ModelReceipt, Integer> col_Total;
    @FXML
    private TableColumn<ModelReceipt, String> col_Status;
    @FXML
    private AnchorPane add_receipt;
    @FXML
    private AnchorPane edit_receipt;
    @FXML
    private AnchorPane delete_receipt;
    @FXML
    private Text txtMedicineName;
    @FXML
    private TextField Qty01;
    @FXML
    private TextField Qty02;
    @FXML
    private TextField Qty03;
    @FXML
    private Text txtUnitPrice;
    @FXML
    private Button btnAddItem;
    @FXML
    private Button btnAddReceipt;
    @FXML
    private Button btnEditItem;
    @FXML
    private Button btnDeleteItem;
    @FXML
    private Text txtExpectTotal;
    @FXML
    private Button btnFinishOrder;
    @FXML
    private Button btnReceiptDelete;
    @FXML
    private Button btnOrder;
    @FXML
    private ComboBox<String> SelectMedicine;
    @FXML
    private ComboBox<String> SelectSupplier;
    @FXML
    private Text txtReceiptID;
    @FXML
    private TableView<ModelRDetail> TableReceiptDetail;
    @FXML
    private TableColumn<ModelRDetail, String> col_RMedicineID;
    @FXML
    private TableColumn<ModelRDetail, String> col_RMediName;
    @FXML
    private TableColumn<ModelRDetail, Integer> col_Qty01;
    @FXML
    private TableColumn<ModelRDetail, Integer> col_Qty02;
    @FXML
    private TableColumn<ModelRDetail, Integer> col_Qty03;
    @FXML
    private TableColumn<ModelRDetail, Integer> col_ImportPrice;
    @FXML
    private Button btnPrint;

    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;
    private Statement stm;

    public ObservableList<ModelReceipt> ReceiptList() {
        //goi ham
        ObservableList<ModelReceipt> receiptList = FXCollections.observableArrayList();
        String sql = "select * from ImportOrderReceipt inner join Supplier on ImportOrderReceipt.SupplierID=Supplier.SupplierID";
        try {
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();
            ModelReceipt listR;
            while (rs.next()) {
                listR = new ModelReceipt(rs.getString("ReceiptID"),
                        rs.getDate("OrderDate"),
                        rs.getDate("ReceiveDate"),
                        rs.getInt("UserID"),
                        rs.getString("SupplierID"),
                        rs.getString("SupplierName"),
                        rs.getInt("TotalSpend"),
                        rs.getString("Status")
                );
                receiptList.add(listR);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return receiptList;
    }
```

26.1 Code

```java
public void showRList() {
        ObservableList<ModelReceipt> showRList = ReceiptList();
        col_ReceiptID.setCellValueFactory(new PropertyValueFactory<>("ReceiptID"));
        col_orderDate.setCellValueFactory(new PropertyValueFactory<>("OrderDate"));
        col_receiveDate.setCellValueFactory(new PropertyValueFactory<>("ReceiveDate"));
        col_supplier.setCellValueFactory(new PropertyValueFactory<>("SupplierName"));
        col_Total.setCellValueFactory(new PropertyValueFactory<>("TotalSpend"));
        col_Status.setCellValueFactory(new PropertyValueFactory<>("Status"));
        TableReceipt.setItems(showRList);
    }

    public ObservableList<ModelRDetail> DetailList(String selectReceiptID) {
        //goi ham
        ObservableList<ModelRDetail> detailtList = FXCollections.observableArrayList();
        String sql = "select * from ReceiptDetail inner join tblProduct on ReceiptDetail.ItemID=tblProduct.MedicineID where ReceiptDetail.ReceiptID= ? ";
        try {
            pst = conn.prepareStatement(sql);
            pst.setString(1, selectReceiptID);
            rs = pst.executeQuery();
            ModelRDetail data;
            while (rs.next()) {
                data = new ModelRDetail(rs.getString("ReceiptID"),
                        rs.getString("MedicineID"),
                        rs.getString("MedicineName"),
                        rs.getInt("OQty01"),
                        rs.getInt("OQty02"),
                        rs.getInt("OQty03"),
                        rs.getInt("ITotalPrice"),
                        rs.getInt("CostPrice"),
                        rs.getInt("Quantity"),
                        rs.getInt("Subunitsperunit"),
                        rs.getInt("Itemspersubunit")
                );
                detailtList.add(data);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return detailtList;
    }

    public ObservableList<ModelRDetail> DetailList() {
        //goi ham
        ObservableList<ModelRDetail> detailtList = FXCollections.observableArrayList();
        String sql = "select * from ReceiptDetail inner join tblProduct on ReceiptDetail.ItemID=tblProduct.MedicineID ";
        try {
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();
            ModelRDetail data;
            while (rs.next()) {
                data = new ModelRDetail(rs.getString("ReceiptID"),
                        rs.getString("MedicineID"),
                        rs.getString("MedicineName"),
                        rs.getInt("OQty01"),
                        rs.getInt("OQty02"),
                        rs.getInt("OQty03"),
                        rs.getInt("ITotalPrice"),
                        rs.getInt("CostPrice"),
                        rs.getInt("Quantity"),
                        rs.getInt("Subunitsperunit"),
                        rs.getInt("Itemspersubunit")
                );
                detailtList.add(data);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return detailtList;
    }

    public void showRDetail(String selectReceiptID) {
        ObservableList<ModelRDetail> showRDetal = DetailList(selectReceiptID);
        col_RMedicineID.setCellValueFactory(new PropertyValueFactory<>("MedicineID"));
        col_RMediName.setCellValueFactory(new PropertyValueFactory<>("MedicineName"));
        col_Qty01.setCellValueFactory(new PropertyValueFactory<>("OQty01"));
        col_Qty02.setCellValueFactory(new PropertyValueFactory<>("OQty02"));
        col_Qty03.setCellValueFactory(new PropertyValueFactory<>("OQty03"));
        col_ImportPrice.setCellValueFactory(new PropertyValueFactory<>("ITotalPrice"));
        TableReceiptDetail.setItems(showRDetal);

        purchaseDisplayTotal();
    }

    public void selectMedicine() {
        ObservableList<String> listMedicine = FXCollections.observableArrayList();
        String sql = "select MedicineName from tblProduct";
        try {
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();
            while (rs.next()) {
                listMedicine.add(rs.getString("MedicineName"));
            }
            SelectMedicine.setItems(listMedicine);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public void selectSupplier() {
        ObservableList<String> listSupplier = FXCollections.observableArrayList();
        String sql = "select SupplierName from Supplier";
        try {
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();
            while (rs.next()) {
                listSupplier.add(rs.getString("SupplierName"));
            }
            SelectSupplier.setItems(listSupplier);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
```

```java
@Override
    public void initialize(URL url, ResourceBundle rb) {
        conn = DB.ConnectDB.getConnectDB();
        showRList();
        selectMedicine();
        selectSupplier();
        System.out.println("ok");
    }

    public ObservableList<ModelReceipt> SearchReceiptList(String keyword) {
        ObservableList<ModelReceipt> receiptList = FXCollections.observableArrayList();
        String sql = "select * from ImportOrderReceipt inner join Supplier on ImportOrderReceipt.SupplierID=Supplier.SupplierID";
        if (keyword != null && !keyword.trim().isEmpty()) {
            sql += " where lower(ReceiptID) like ? or lower(SupplierName) like ? or lower(Status) like ?";
        }
        try {
            pst = conn.prepareStatement(sql);
            if (keyword != null && !keyword.trim().isEmpty()) {
                String searchPattern = "%" + keyword.toLowerCase() + "%";
                pst.setString(1, searchPattern);
                pst.setString(2, searchPattern);
                pst.setString(3, searchPattern);
            }
            rs = pst.executeQuery();
            ModelReceipt listR;
            while (rs.next()) {
                listR = new ModelReceipt(rs.getString("ReceiptID"),
                        rs.getDate("OrderDate"),
                        rs.getDate("ReceiveDate"),
                        rs.getInt("UserID"),
                        rs.getString("SupplierID"),
                        rs.getString("SupplierName"),
                        rs.getInt("TotalSpend"),
                        rs.getString("Status")
                );
                receiptList.add(listR);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return receiptList;
    }

    public void showsearchRList(String keyword) {
        ObservableList<ModelReceipt> showsearchRList;
        if (keyword == null || keyword.trim().isEmpty()) {
            showsearchRList = ReceiptList(); // Lấy danh sách đầy đủ nếu không có từ khóa
        } else {
            showsearchRList = SearchReceiptList(keyword); // Lấy danh sách dựa trên từ khóa
        }

        col_ReceiptID.setCellValueFactory(new PropertyValueFactory<>("ReceiptID"));
        col_orderDate.setCellValueFactory(new PropertyValueFactory<>("OrderDate"));
        col_receiveDate.setCellValueFactory(new PropertyValueFactory<>("ReceiveDate"));
        col_supplier.setCellValueFactory(new PropertyValueFactory<>("SupplierName"));
        col_Total.setCellValueFactory(new PropertyValueFactory<>("TotalSpend"));
        col_Status.setCellValueFactory(new PropertyValueFactory<>("Status"));

        TableReceipt.setItems(showsearchRList);
    }

    @FXML
    public void search_receipt(ActionEvent event) {
        SearchReceipt.textProperty().addListener((observable, oldValue, newValue) -> {
            showsearchRList(newValue); // Gọi lại showRList với từ khóa tìm kiếm mới
        });
    }

    @FXML
    void selectReceipt(MouseEvent event) {
        if (event.getClickCount() == 2) { // Chỉ xử lý khi người dùng chọn một lần
            ModelReceipt data = TableReceipt.getSelectionModel().getSelectedItem();
            if (data != null) {
                txtReceiptID.setText(data.getReceiptID());
                showRDetail(data.getReceiptID());
            }
        }
    }

    @FXML
    void selectItem(MouseEvent event) {
        if (event.getClickCount() == 2) { // Chỉ xử lý khi người dùng chọn một lần
            ModelRDetail data = TableReceiptDetail.getSelectionModel().getSelectedItem();
            if (data != null) {
                txtMedicineName.setText(data.getMedicineName());
                txtUnitPrice.setText(data.getCostPrice().toString() + " đ");
                Qty01.setText(data.getOQty01().toString());
                Qty02.setText(data.getOQty02().toString());
                Qty03.setText(data.getOQty03().toString());
            }
        }
    }

    @FXML
    public void search(ActionEvent event) {
    }

    private String ReceiptID;

    public String getHighestReceiptID() {
        String highestID = "R0000"; // Default value if there are no records
        String sql = "SELECT MAX(ReceiptID) AS HighestID FROM ImportOrderReceipt";
        try {
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();
            if (rs.next()) {
                highestID = rs.getString("HighestID");
                if(highestID == null){
                    highestID = "O0000";
                }
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
        return highestID;
    }
```

```
public String generateNextReceiptID() {
        String highestID = getHighestReceiptID();

        // Kiểm tra nếu highestID là null hoặc không phải độ dài mong muốn
        int number = Integer.parseInt(highestID.substring(1));
        number++;
        return String.format("O%04d", number);
    }

    @FXML
    void btn_add_receipt(ActionEvent event) {
        txtMedicineName.setText(null);
        txtUnitPrice.setText(null);
        Qty01.setText(null);
        Qty02.setText(null);
        Qty03.setText(null);
        TableReceiptDetail.getItems().clear();
        String newReceiptID = generateNextReceiptID();
        txtReceiptID.setText(newReceiptID);
    }

    private int totalPriceD;

    public void purchaseDisplayTotal() {

        String sql = "SELECT SUM(ITotalPrice) as total FROM ReceiptDetail WHERE ReceiptID = '"  + txtReceiptID.getText() + "'";

        try {
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();

            if (rs.next()) {
                totalPriceD = rs.getInt("total");
            }
            txtExpectTotal.setText(String.valueOf(totalPriceD));

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

    }

    private String ItemID;
    private String SupplierID;
    private int ITotal;

    @FXML
    public void btn_add_item(ActionEvent event) {
        String sql = "INSERT INTO ReceiptDetail (ReceiptID, ItemID, OQty01, OQty02, OQty03, ITotalPrice) "
                + "VALUES(?,?,?,?,?,?)";
        String sql2 = "INSERT INTO ImportOrderReceipt (ReceiptID,SupplierID) "
                + "VALUES(?,?)";
        try {

            Alert alert;

            if (SelectMedicine.getSelectionModel().getSelectedItem() == null
                    || Qty01.getText().isEmpty()
                    || Qty02.getText().isEmpty()
                    || Qty03.getText().isEmpty()
                    || SelectSupplier.getSelectionModel().getSelectedItem() == null) {
                alert = new Alert(AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("Please fill all blank fields");
                alert.showAndWait();
            } else {
                String checkReceipt = "SELECT ReceiptID FROM ImportOrderReceipt WHERE ReceiptID = '"
                        + txtReceiptID.getText() + "'";
                stm = conn.createStatement();
                rs = stm.executeQuery(checkReceipt);

                if (!rs.next()) {
                    pst = conn.prepareStatement(sql2);
                    pst.setString(1, txtReceiptID.getText());
                    String checkISupplier = "SELECT SupplierID FROM Supplier WHERE SupplierName = '"
                            + SelectSupplier.getSelectionModel().getSelectedItem() + "'";

                    stm = conn.createStatement();
                    rs = stm.executeQuery(checkISupplier);
                    if (rs.next()) {
                        SupplierID = rs.getString("SupplierID");
                    }
                    pst.setString(2, String.valueOf(SupplierID));
                    pst.executeUpdate();
                }

                pst = conn.prepareStatement(sql);
                pst.setString(1, txtReceiptID.getText());
                String checkIName = "SELECT MedicineID FROM tblProduct WHERE MedicineName = '"
                        + SelectMedicine.getSelectionModel().getSelectedItem() + "'";

                stm = conn.createStatement();
                rs = stm.executeQuery(checkIName);
                if (rs.next()) {
                    ItemID = rs.getString("MedicineID");
                }
                pst.setString(2, String.valueOf(ItemID));
                pst.setString(3, Qty01.getText());
                pst.setString(4, Qty02.getText());
                pst.setString(5, Qty03.getText());
                String checkData = "SELECT * FROM tblProduct WHERE MedicineName = '"
                        + SelectMedicine.getSelectionModel().getSelectedItem() + "'";

                stm = conn.createStatement();
                rs = stm.executeQuery(checkData);
                int priceD = 0;
                int RQty02 = 0;
                int RQty03 = 0;
                int OQty01 = Integer.parseInt(Qty01.getText());
                int OQty02 = Integer.parseInt(Qty02.getText());
                int OQty03 = Integer.parseInt(Qty03.getText());
                if (rs.next()) {
                    priceD = rs.getInt("CostPrice");
                    RQty02 = rs.getInt("Subunitsperunit");
                    RQty03 = rs.getInt("Itemspersubunit");
                }
                if (OQty01 == 0) {
                    OQty01 = 1;
                    RQty02 = 1;
                }
                if (OQty02 == 0) {
                    OQty02 = 1;
                    RQty03 = 1;
                }
                if (OQty03 == 0) {
                    OQty03 = 1;
                }
                ITotal = (priceD * RQty02 * RQty03 * OQty01 * OQty02 * OQty03);

                pst.setString(6, String.valueOf(ITotal));

                pst.executeUpdate();

                showRList();
                showRDetail(txtReceiptID.getText());
                purchaseDisplayTotal();
            }

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

```java
@FXML
    public void btn_edit_item(ActionEvent event) {

        try {
            Alert alert;

            if (Qty01.getText().isEmpty() || Qty02.getText().isEmpty() || Qty03.getText().isEmpty()) {
                alert = new Alert(AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("Please fill all blank fields");
                alert.showAndWait();
                return;
            }

            String checkIName = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
            try (PreparedStatement pst = conn.prepareStatement(checkIName)) {
                pst.setString(1, txtMedicineName.getText());
                try (ResultSet rs = pst.executeQuery()) {
                    if (rs.next()) {
                        ItemID = rs.getString("MedicineID");
                    } else {
                        alert = new Alert(AlertType.ERROR);
                        alert.setTitle("Error Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Item not found");
                        alert.showAndWait();
                        return;
                    }
                }
            }

            String checkData = "SELECT * FROM tblProduct WHERE MedicinesName = ?";
            int priceD = 0;
            int RQty02 = 0;
            int RQty03 = 0;
            int OQty01 = Integer.parseInt(Qty01.getText());
            int OQty02 = Integer.parseInt(Qty02.getText());
            int OQty03 = Integer.parseInt(Qty03.getText());

            try (PreparedStatement pst = conn.prepareStatement(checkData)) {
                pst.setString(1, txtMedicineName.getText());
                try (ResultSet rs = pst.executeQuery()) {
                    if (rs.next()) {
                        priceD = rs.getInt("CostPrice");
                        RQty02 = rs.getInt("Subunitsperunit");
                        RQty03 = rs.getInt("Itemspersubunit");
                    }
                }
            }

            if (OQty01 == 0) {
                OQty01 = 1;
                RQty02 = 1;
            }
            if (OQty02 == 0) {
                OQty02 = 1;
                RQty03 = 1;
            }
            if (OQty03 == 0) {
                OQty03 = 1;
            }

            ITotal = priceD * RQty02 * RQty03 * OQty01 * OQty02 * OQty03;

            alert = new Alert(AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Message");
            alert.setHeaderText(null);
            alert.setContentText("Are you sure you want to UPDATE Medicine: " + txtMedicineName.getText() + "?");
            Optional<ButtonType> option = alert.showAndWait();

            if (option.isPresent() && option.get() == ButtonType.OK) {
                String sql = "UPDATE ReceiptDetail SET OQty01 = ?, OQty02 = ?, OQty03 = ?, ITotalPrice = ? WHERE ReceiptID = ? AND ItemID = ?";
                try (PreparedStatement pst = conn.prepareStatement(sql)) {
                    pst.setString(1, Qty01.getText());
                    pst.setString(2, Qty02.getText());
                    pst.setString(3, Qty03.getText());
                    pst.setInt(4, ITotal);
                    pst.setString(5, txtReceiptID.getText());
                    pst.setString(6, ItemID);

                    int affectedRows = pst.executeUpdate();
                    if (affectedRows > 0) {
                        alert = new Alert(AlertType.INFORMATION);
                        alert.setTitle("Information Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Successfully Updated!");
                        alert.showAndWait();

                        showRList();
                        showRDetail(txtReceiptID.getText());
                        purchaseDisplayTotal();
                    } else {
                        alert = new Alert(AlertType.ERROR);
                        alert.setTitle("Error Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Update failed, no rows affected");
                        alert.showAndWait();
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

```

```java
@FXML
    void btn_delete_item(ActionEvent event) {
        try {
            Alert alert;

            if (txtReceiptID.getText().isEmpty()) {
                alert = new Alert(AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("Please select an item and ensure the ReceiptID is filled");
                alert.showAndWait();
                return;
            }

            String checkIName = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
            try (PreparedStatement pst = conn.prepareStatement(checkIName)) {
                pst.setString(1, txtMedicineName.getText());
                try (ResultSet rs = pst.executeQuery()) {
                    if (rs.next()) {
                        ItemID = rs.getString("MedicineID");
                    } else {
                        alert = new Alert(AlertType.ERROR);
                        alert.setTitle("Error Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Item not found");
                        alert.showAndWait();
                        return;
                    }
                }
            }

            alert = new Alert(AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Message");
            alert.setHeaderText(null);
            alert.setContentText("Are you sure you want to DELETE the selected item?");
            Optional<ButtonType> option = alert.showAndWait();

            if (option.isPresent() && option.get() == ButtonType.OK) {
                String sql = "DELETE FROM ReceiptDetail WHERE ReceiptID = ? AND ItemID = ?";
                try (PreparedStatement pst = conn.prepareStatement(sql)) {
                    pst.setString(1, txtReceiptID.getText());
                    pst.setString(2, ItemID);

                    int affectedRows = pst.executeUpdate();
                    if (affectedRows > 0) {
                        alert = new Alert(AlertType.INFORMATION);
                        alert.setTitle("Information Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Successfully Deleted!");
                        alert.showAndWait();

                        showRList();
                        showRDetail(txtReceiptID.getText());
                        purchaseDisplayTotal();
                    } else {
                        alert = new Alert(AlertType.ERROR);
                        alert.setTitle("Error Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Delete failed, no rows affected");
                        alert.showAndWait();
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @FXML
    void showMedicine(MouseEvent event) {
        if (event.getClickCount() == 1) { // Chỉ xử lý khi người dùng chọn một lần
            SelectMedicine.getSelectionModel().selectedItemProperty().addListener((options, oldValue, newValue) -> {
                if (newValue != null) {
                    String sql = "SELECT * FROM tblProduct WHERE MedicineName = ?";
                    try {
                        pst = conn.prepareStatement(sql);
                        pst.setString(1, newValue);
                        rs = pst.executeQuery();
                        if (rs.next()) {
                            String ItemName = rs.getString("MedicineName");
                            txtMedicineName.setText(ItemName);
                            String ImportPrice = rs.getString("CostPrice");
                            txtUnitPrice.setText(ImportPrice.toString() + " đ");
                        }
                        Qty01.setText("0");
                        Qty02.setText("0");
                        Qty03.setText("0");
                    } catch (Exception e) {
                        System.out.println(e.getMessage());
                    }
                }
            });
        }
    }
```

```
@FXML
    void showSupplier(ActionEvent event) {

    }

    @FXML
    public void btn_finish(ActionEvent event) {
        try {
            Alert alert;

            if (TableReceiptDetail.getItems().isEmpty()) {
                alert = new Alert(AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("This order item first");
                alert.showAndWait();
                return;
            }

            String checkStatus = "SELECT Status FROM ImportOrderReceipt WHERE ReceiptID = ?";
            try (PreparedStatement pst = conn.prepareStatement(checkStatus)) {
                pst.setString(1, txtReceiptID.getText());
                try (ResultSet rs = pst.executeQuery()) {
                    if (rs.next()) {
                        String status = rs.getString("Status");
                        if ("Finishing".equals(status)) {
                            alert = new Alert(Alert.AlertType.ERROR);
                            alert.setTitle("Error Message");
                            alert.setHeaderText(null);
                            alert.setContentText("This Order is already finished");
                            alert.showAndWait();
                            return;
                        }
                    }
                }
            } catch (SQLException e) {
                e.printStackTrace();
                alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("An error occurred: " + e.getMessage());
                alert.showAndWait();
            }

            alert = new Alert(AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Message");
            alert.setHeaderText(null);
            alert.setContentText("Are you sure you want to finish this ORDER: " + txtReceiptID.getText() + "?");
            Optional<ButtonType> option = alert.showAndWait();

            if (option.isPresent() && option.get() == ButtonType.OK) {
                String sql = "UPDATE ImportOrderReceipt SET ReceiveDate = ?, TotalSpend = ?, Status= ? WHERE ReceiptID = ?";
                try (PreparedStatement pst = conn.prepareStatement(sql)) {
                    LocalDate currentDate = LocalDate.now();
                    Date sqlDate = Date.valueOf(currentDate);
                    pst.setDate(1, sqlDate);
                    pst.setString(2, txtExpectTotal.getText());
                    pst.setString(3, "Finishing");
                    pst.setString(4, txtReceiptID.getText());

                    int affectedRows = pst.executeUpdate();
                    if (affectedRows > 0) {
                        alert = new Alert(AlertType.INFORMATION);
                        alert.setTitle("Information Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Successfully Updated!");
                        alert.showAndWait();

                        showRList();
                        showRDetail(txtReceiptID.getText());
                        purchaseDisplayTotal();
                    } else {
                        alert = new Alert(AlertType.ERROR);
                        alert.setTitle("Error Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Update failed, no rows affected");
                        alert.showAndWait();
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @FXML
    public void btn_Rdelete(ActionEvent event) {
        try {
            Alert alert;

            String checkStatus = "SELECT Status FROM ImportOrderReceipt WHERE ReceiptID = ?";
            try (PreparedStatement pst = conn.prepareStatement(checkStatus)) {
                pst.setString(1, txtReceiptID.getText());
                try (ResultSet rs = pst.executeQuery()) {
                    if (rs.next()) {
                        String status = rs.getString("Status");
                        if ("Finishing".equals(status)) {
                            alert = new Alert(Alert.AlertType.ERROR);
                            alert.setTitle("Error Message");
                            alert.setHeaderText(null);
                            alert.setContentText("This Order is already finished");
                            alert.showAndWait();
                            return;
                        }
                    }
                }
            } catch (SQLException e) {
                e.printStackTrace();
                alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("An error occurred: " + e.getMessage());
                alert.showAndWait();
            }

            alert = new Alert(AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Message");
            alert.setHeaderText(null);
            alert.setContentText("Are you sure you want to delete this ORDER: " + txtReceiptID.getText() + "?");
            Optional<ButtonType> option = alert.showAndWait();

```

```
    if (option.isPresent() && option.get() == ButtonType.OK) {
                String sql1 = "DELETE FROM ReceiptDetail WHERE ReceiptID = ?";
                String sql2 = "DELETE FROM ImportOrderReceipt WHERE ReceiptID = ?";

                try (PreparedStatement pst1 = conn.prepareStatement(sql1); PreparedStatement pst2 = conn.prepareStatement(sql2)) {

                    conn.setAutoCommit(false);  // Start transaction

                    String receiptID = txtReceiptID.getText();
                    pst1.setString(1, receiptID);
                    pst2.setString(1, receiptID);

                    int affectedRows1 = pst1.executeUpdate();
                    int affectedRows2 = pst2.executeUpdate();
                    if (affectedRows1 > 0 && affectedRows2 > 0) {
                        conn.commit();
                        alert = new Alert(AlertType.INFORMATION);
                        alert.setTitle("Information Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Successfully Delete!");
                        alert.showAndWait();

                        showRList();
                        showRDetail(txtReceiptID.getText());
                        purchaseDisplayTotal();
                    } else {
                        alert = new Alert(AlertType.ERROR);
                        alert.setTitle("Error Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Delete failed, no rows affected");
                        alert.showAndWait();
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @FXML
    public void btn_order(ActionEvent event) {
        try {
            Alert alert;

            if (TableReceiptDetail.getItems().isEmpty() || SelectSupplier.getSelectionModel().getSelectedItem() == null) {
                alert = new Alert(AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("Please add Item or Supplier");
                alert.showAndWait();
                return;
            }

            String checkISupplier = "SELECT SupplierID FROM Supplier WHERE SupplierName = ?";
            try (PreparedStatement pst = conn.prepareStatement(checkISupplier)) {
                pst.setString(1, SelectSupplier.getSelectionModel().getSelectedItem());
                try (ResultSet rs = pst.executeQuery()) {
                    if (rs.next()) {
                        SupplierID = rs.getString("SupplierID");
                    } else {
                        alert = new Alert(AlertType.ERROR);
                        alert.setTitle("Error Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Supplier not found");
                        alert.showAndWait();
                        return;
                    }
                }
            }

            String checkStatus = "SELECT Status FROM ImportOrderReceipt WHERE ReceiptID = ?";
            try (PreparedStatement pst = conn.prepareStatement(checkStatus)) {
                pst.setString(1, txtReceiptID.getText());
                try (ResultSet rs = pst.executeQuery()) {
                    if (rs.next()) {
                        String status = rs.getString("Status");
                        if ("Finishing".equals(status)) {
                            alert = new Alert(Alert.AlertType.ERROR);
                            alert.setTitle("Error Message");
                            alert.setHeaderText(null);
                            alert.setContentText("This Order is already finished");
                            alert.showAndWait();
                            return;
                        }
                    }
                }
            } catch (SQLException e) {
                e.printStackTrace();
                alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Error Message");
                alert.setHeaderText(null);
                alert.setContentText("An error occurred: " + e.getMessage());
                alert.showAndWait();
            }

            alert = new Alert(AlertType.CONFIRMATION);
            alert.setTitle("Confirmation Message");
            alert.setHeaderText(null);
            alert.setContentText("Are you sure you want to ORDER: " + txtReceiptID.getText() + "?");
            Optional<ButtonType> option = alert.showAndWait();

            if (option.isPresent() && option.get() == ButtonType.OK) {
                String sql = "UPDATE ImportOrderReceipt SET OrderDate = ?, SupplierID = ?, TotalSpend = ?, Status= ? WHERE ReceiptID = ?";
                try (PreparedStatement pst = conn.prepareStatement(sql)) {
                    LocalDate currentDate = LocalDate.now();
                    Date sqlDate = Date.valueOf(currentDate);
                    pst.setDate(1, sqlDate);
                    pst.setString(2, SupplierID);
                    pst.setString(3, txtExpectTotal.getText());
                    pst.setString(4, "Ordering");
                    pst.setString(5, txtReceiptID.getText());

                    int affectedRows = pst.executeUpdate();
                    if (affectedRows > 0) {
                        alert = new Alert(AlertType.INFORMATION);
                        alert.setTitle("Information Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Successfully Updated!");
                        alert.showAndWait();

                        showRList();
                        showRDetail(txtReceiptID.getText());
                        purchaseDisplayTotal();
                    } else {
                        alert = new Alert(AlertType.ERROR);
                        alert.setTitle("Error Message");
                        alert.setHeaderText(null);
                        alert.setContentText("Update failed, no rows affected");
                        alert.showAndWait();
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    @FXML
    public void btn_print(ActionEvent event) {
    }

}
```

27 Add Supplier

```java
public class SupplierController implements Initializable {

    @FXML
    private TextField SearchSupplier;
    @FXML
    private Button btnAddSupplier;
    @FXML
    private TableView<ModelSupplier> TableSupplier;
    @FXML
    private TableColumn<ModelSupplier, String> col_SupplierID;
    @FXML
    private TableColumn<ModelSupplier, String> col_SupplierName;
    @FXML
    private TableColumn<ModelSupplier, String> col_Address;
    @FXML
    private TableColumn<ModelSupplier, String> col_Phone;
    @FXML
    private TableColumn<ModelSupplier, String> col_Email;

    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;
    private Statement stm;

    public ObservableList<ModelSupplier> SuppliertList(){
        //goi ham
        ObservableList<ModelSupplier> suppliertList = FXCollections.observableArrayList();
        String sql = "select * from Supplier";
        try{
            pst = conn.prepareStatement(sql);
            rs = pst.executeQuery();
            ModelSupplier listS;
            while(rs.next()){
                listS = new ModelSupplier(rs.getString("SupplierID"),
                        rs.getString("SupplierName"),
                        rs.getString("Address"),
                        rs.getString("PhoneNumber"),
                        rs.getString("Email")
                );
                suppliertList.add(listS);
            }
        }catch(Exception e){
            System.out.println(e.getMessage());
        }
        return suppliertList;
    }


    public void showSList(){
        ObservableList<ModelSupplier> showSList = SuppliertList();
        col_SupplierID.setCellValueFactory(new PropertyValueFactory<>("SupplierID"));
        col_SupplierName.setCellValueFactory(new PropertyValueFactory<>("SupplierName"));
        col_Address.setCellValueFactory(new PropertyValueFactory<>("Address"));
        col_Phone.setCellValueFactory(new PropertyValueFactory<>("PhoneNumber"));
        col_Email.setCellValueFactory(new PropertyValueFactory<>("Email"));

        TableSupplier.setItems(showSList);
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        conn=DB.ConnectDB.getConnectDB();
       showSList();
    }
```

```java

    @FXML
    private void btn_add_supplier(MouseEvent event) throws IOException {
        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("Addsupplier.fxml"));
            Parent root = loader.load();

            AddsupplierController controller = loader.getController();
            controller.setSupplierController(this);

            Stage stage = new Stage();
            stage.initModality(Modality.APPLICATION_MODAL);
            stage.setTitle("Add Supplier");
            stage.setScene(new Scene(root));
            stage.showAndWait();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    @FXML
    private void sellectCustomer(MouseEvent event) {
         if (event.getClickCount() == 2) {
            openUpdateSupplierWindow();
        }
    }

    @FXML
    private void openUpdateSupplierWindow() {
        ModelSupplier selectedSupplier = TableSupplier.getSelectionModel().getSelectedItem();

        try {
            FXMLLoader loader = new FXMLLoader(getClass().getResource("UpdateSupplier.fxml"));
            Parent root = loader.load();

            UpdateSupplierController controller = loader.getController();
            controller.setSupplierController(this);
            controller.setSupplierData(selectedSupplier.getSupplierID(),
                                       selectedSupplier.getSupplierName(),
                                       selectedSupplier.getPhoneNumber(),
                                       selectedSupplier.getAddress(),
                                       selectedSupplier.getEmail());

            Stage stage = new Stage();
            stage.initModality(Modality.APPLICATION_MODAL);
            stage.setTitle("Update Supplier");
            stage.setScene(new Scene(root));
            stage.showAndWait();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

```

28 Update Supplier

```
public class UpdateSupplierController implements Initializable {

    @FXML
    private TextField txtSupplierName;
    @FXML
    private Button btnCancle;
    @FXML
    private TextField txtPhone;
    @FXML
    private TextField txtAddress;
    @FXML
    private TextField txtEmail;
    @FXML
    private Button btnUpdate;
    @FXML
    private Button btnDelete;
    @FXML
    private TextField txtSupplierID;

    private Connection conn;
    private PreparedStatement pst;
    private ResultSet rs;
    private SupplierController supplierController;
    private String supplierID;

    public void setSupplierController(SupplierController supplierController) {
        this.supplierController = supplierController;
    }

    public void setSupplierData(String supplierID, String supplierName, String phone, String address, String email) {
        this.supplierID = supplierID;
        txtSupplierID.setText(supplierID);
        txtSupplierID.setEditable(false);
        txtSupplierName.setText(supplierName);
        txtPhone.setText(phone);
        txtAddress.setText(address);
        txtEmail.setText(email);
    }

    @Override
    public void initialize(URL url, ResourceBundle rb) {
        conn=DB.ConnectDB.getConnectDB();
    }

    @FXML
    private void btn_Cancle(ActionEvent event) {
        Stage stage = (Stage) btnCancle.getScene().getWindow();
        stage.close();
    }

    private boolean validatePhone(String phone) {
        // Số điện thoại hợp lệ phải có 10 chữ số và bắt đầu bằng số 0
        return phone.matches("0[0-9]{9}");
    }

    private boolean validateEmail(String email) {
        // Email hợp lệ theo định dạng phổ biến
        return email.matches("^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}$");
    }

    private void showAlert(String title, String message) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle(title);
        alert.setHeaderText(null);
        alert.setContentText(message);
        alert.showAndWait();
    }

    @FXML
    private void btn_update(ActionEvent event) {
        String supplierName = txtSupplierName.getText();
        String phone = txtPhone.getText();
        String address = txtAddress.getText();
        String email = txtEmail.getText();

        if (!validatePhone(phone)) {
            showAlert("Invalid Phone Number", "Phone number must be 10 digits and start with 0.");
            return;
        }

        if (!validateEmail(email)) {
            showAlert("Invalid Email Address", "Please enter a valid email address.");
            return;
        }

        String sql = "UPDATE Supplier SET SupplierName=?, PhoneNumber=?, Address=?, Email=? WHERE SupplierID=?";
        try {
            pst = conn.prepareStatement(sql);
            pst.setString(1, supplierName);
            pst.setString(2, phone);
            pst.setString(3, address);
            pst.setString(4, email);
            pst.setString(5, supplierID);
            pst.executeUpdate();
            System.out.println("Supplier updated successfully!");

            // Cập nhật bảng trong SupplierController
            if (supplierController != null) {
                supplierController.showSList();
            }

            // Đóng cửa sổ hiện tại
            Stage stage = (Stage) btnUpdate.getScene().getWindow();
            stage.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    @FXML
    private void btn_delete(ActionEvent event) {
        String sql = "DELETE FROM Supplier WHERE SupplierID=?";
        try {
            pst = conn.prepareStatement(sql);
            pst.setString(1, supplierID);
            pst.executeUpdate();
            System.out.println("Supplier deleted successfully!");

            // Cập nhật bảng trong SupplierController
            if (supplierController != null) {
                supplierController.showSList();
            }

            // Đóng cửa sổ hiện tại
            Stage stage = (Stage) btnDelete.getScene().getWindow();
            stage.close();
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

}
```

TASK SHEET CODING REVIEW 1, 2, 3

<table>
<tr><td colspan="3">Project: Pharmacy</td><td>Date of preparation of activity plan:</td></tr>
<tr><td>#</td><td>Task</td><td>Prepared by</td><td>Status</td></tr>
<tr><td>1</td><td>Coding Review 1</td><td>Hoang Gia Huy</td><td rowspan="7">Completed</td></tr>
<tr><td>2</td><td>Coding Review 2</td><td>Hoang Gia Huy</td></tr>
<tr><td>3</td><td>Coding Review 3</td><td>Hoang Gia Huy</td></tr>
<tr><td>4</td><td>Document</td><td>Hoang Gia Huy
&
Tran Nhat Linh</td></tr>
<tr><td>5</td><td>PowerPoint</td><td>Nguyen Anh Minh</td></tr>
<tr><td>6</td><td>Video</td><td>Nguyen Anh Quan</td></tr>
<tr><td colspan="3">Review</td><td>Signature of instructor</td></tr>
<tr><td colspan="3"></td><td></td></tr>
<tr><td colspan="3"></td><td>Mr. Pham Cong Danh</td></tr>
</table>

CODE PART

Project: Pharmacy		**Date of preparation of activity plan:**		
#	**Code Part**	**Main Coder**	**Supporter**	**status**
1	Login	Hoang Gia Huy	N/A	Completed
2	Register	Hoang Gia Huy	N/A	Completed
3	Forgot Password	Hoang Gia Huy	N/A	Completed
4	Email	Hoang Gia Huy	Nguyen Anh Minh	Completed
5	Admin Dashboard	Tran Nhat Linh	N/A	Completed
6	Staff Dashboard	Tran Nhat Linh	N/A	Completed
7	Order Transaction	Tran Nhat Linh	N/A	Completed
8	Category	Nguyen Anh Minh	N/A	Completed
9	Import Medicine Details	Nguyen Anh Minh	N/A	Completed
10	Import Order Receipt	Doan Duc Do	N/A	Completed

11	Supplier	Doan Duc Do	N/A	Completed
12	Account	Nguyen Anh Minh	N/A	Completed
13	Export PDF	Nguyen Anh Minh	N/A	Completed
14	Staff	Nguyen Anh Quan	N/A	Completed
15	Payroll	Nguyen Anh Quan	N/A	Completed
16	Timekeeping	Nguyen Anh Quan	N/A	Completed
17	Check in-out	Nguyen Anh Quan	Nguyen Anh Minh	Completed
18	Search	Nguyen Anh Quan & Nguyen Anh Minh & Tran Nhat Linh	N/A	Completed
19	Customer	Tran Nhat Linh	N/A	Completed
Review		Signature of instructor		
		Mr. Pham Cong Danh		