



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.....	3
INTRODUCTION.....	4
PROBLEM DEFINITION	4
CUSTOMER REQUIREMENT SPECIFICATIONS.....	5
Hardware / Software requirement	7
TASK SHEET REVIEW 1	8
REVIEW 2	9
SITE MAP	27
ENTITY RELATIONSHIP DIAGRAM (ERD).....	28
TASK SHEET REVIEW 2	29
REVIEW 3	30
TASK SHEET REVIEW 3	48

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

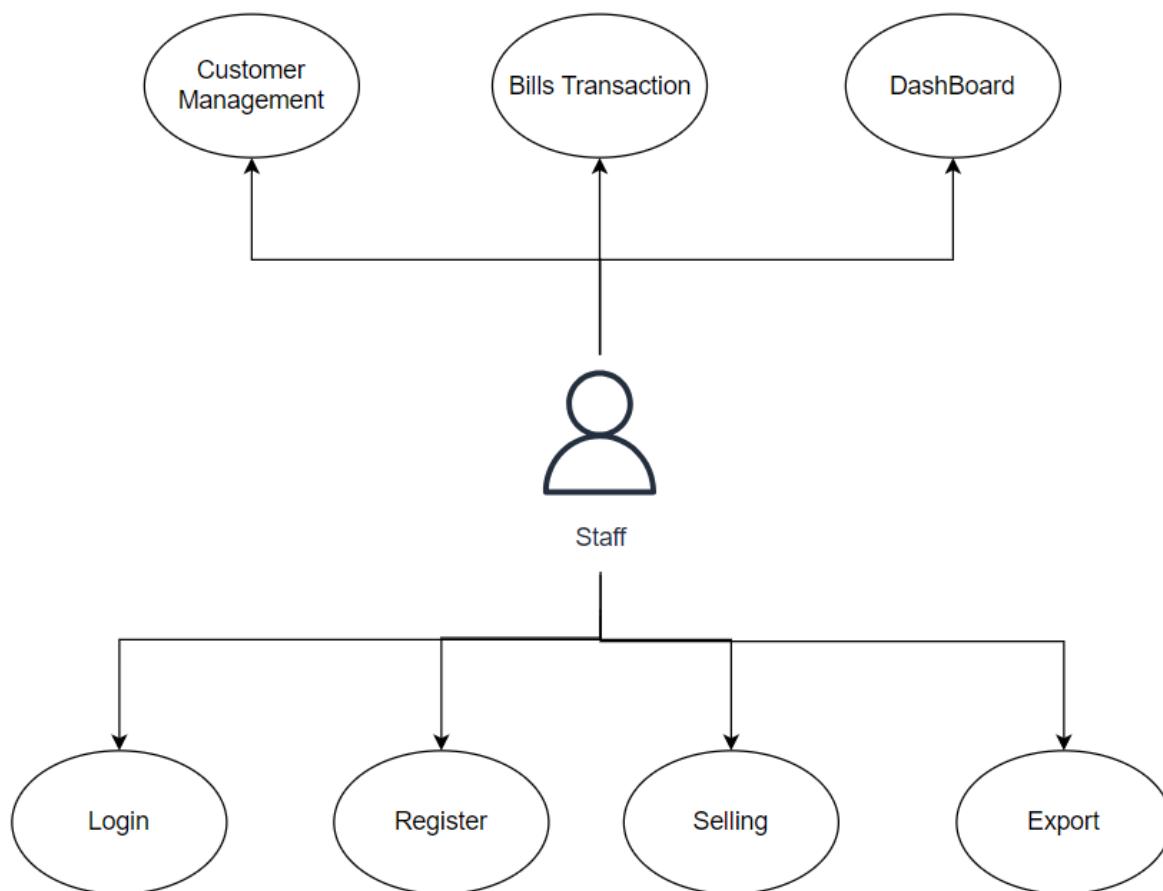
TASK SHEET REVIEW 1

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:
	<p>1. Actors click 'Login' button on Home Page</p> <p>3. Actors input Email and Password, then click the 'Login' button</p>	<p>2. System redirects to Login Page with the following controls:</p> <ul style="list-style-type: none"> - "Email" text field - "Password" text field - "Login" button <p>4. System checks the information</p> <p>5. System redirects to Home page</p>
Exceptions	Actor's actions:	System's responses:
	<p>1. Actors input invalid email or password.</p>	<p>2. System redirects to Login page with the following controls:</p> <ul style="list-style-type: none"> - "Email" text field - "Password" text field - "Login" button <p>3. System shows message: "Invalid email and password"</p>

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:
	<p>1. Actors click ‘Sign Up’ Button under the “Login” button</p> <p>3. Actors fill in the form and click the “Sign Up”</p>	<p>2.The systems show the form which force actors fill in like ID, Full name, Password, Confirm Password, Phone, Email, Address, Role (set Staff default)</p> <p>4. Systems check whether ID, Phone, Email exist.</p> <p>5. System inserts the status into database</p> <p>6. System shows success message and return to Login page</p>
Exceptions	Actor's actions:	System's responses:

Selling

Use case name	Selling	
Actors	Staff	
Description	Staff can sell product from 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
--	-----	-----

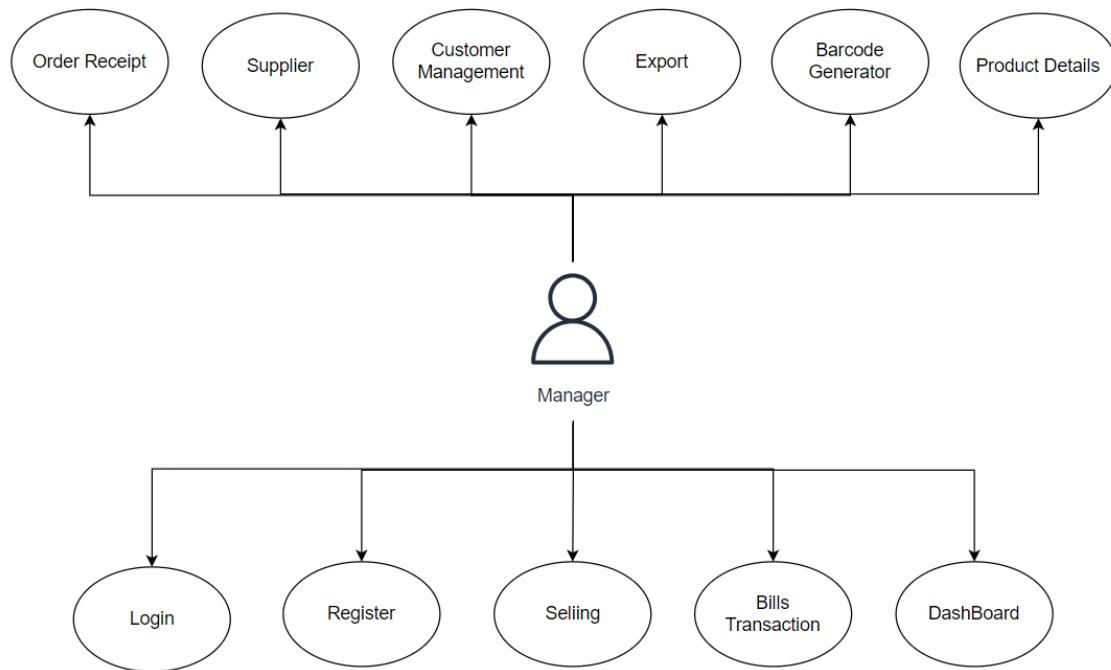
Bills Transaction

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	<p>Success: System show up bills(Receipt ID, Order Date, Receive Date, Supplier, Total, Status) and can delete</p> <p>Fail: N/A</p>	
Basic flow	Actor's actions:	System's responses:
	<p>1. Actors click 'Order Receipt' button</p> <p>3. Admin click "Add New" button and fill the form in right table and click "+" button</p>	<p>2. System show up data</p> <p>4. System will update to database</p>
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

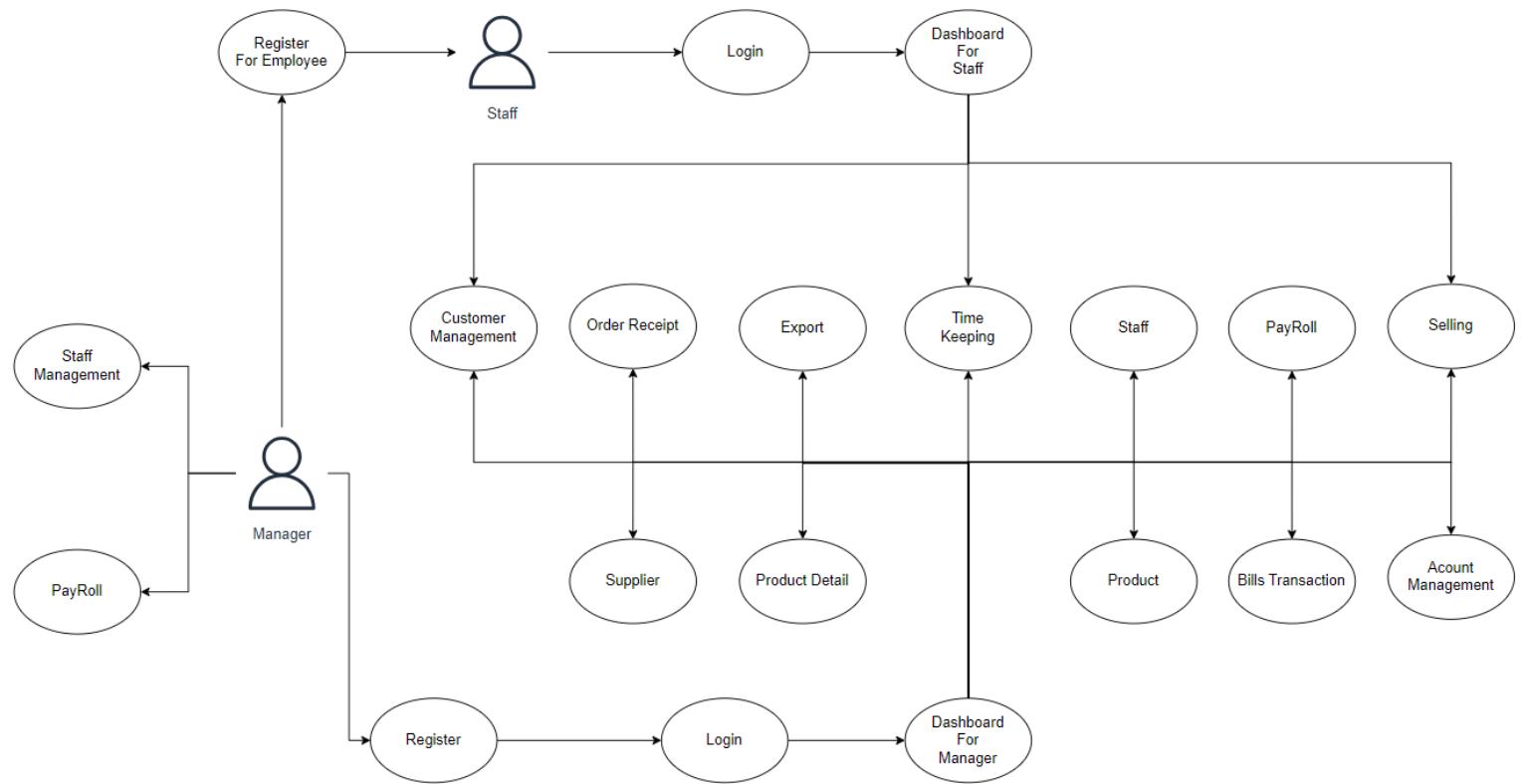
Use case name	Staff	
Actors	Admin	
Description	See all information about employees	
Requirements	N/A	
Pre-conditions	N/A	
Post-conditions	Success: Display all Staff information	
	Fail: N/A	
Basic flow	Actor's actions:	System's responses:
	1. Actors click "Staff" button 3. Actors fill in form under the 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

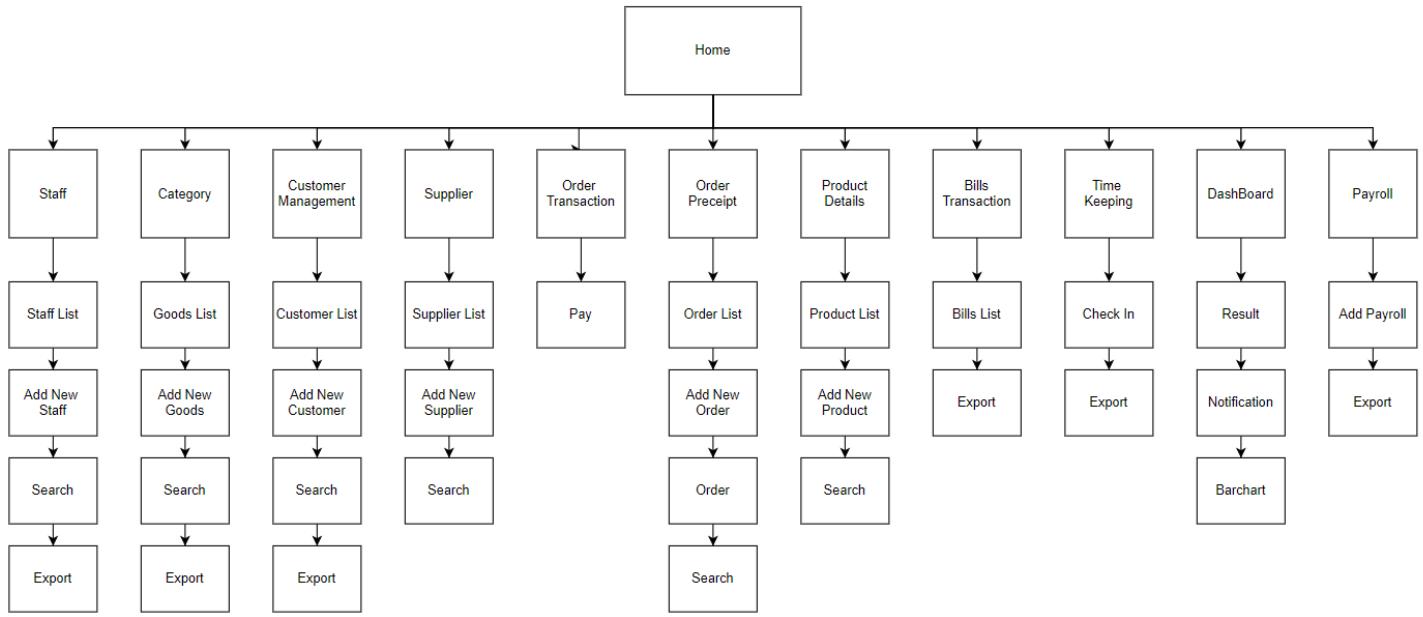
Time Keeping

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

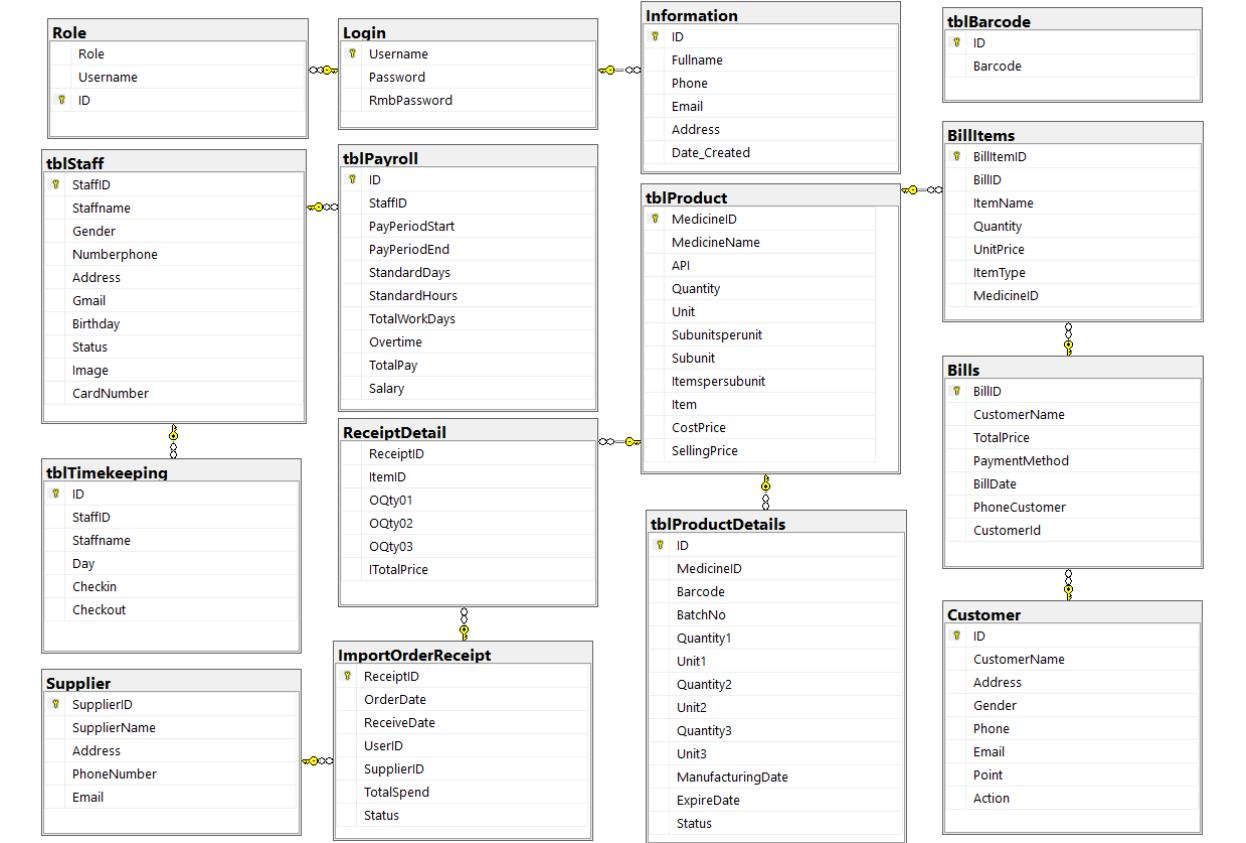
2. Data Flow Diagram (DFD)

SITE MAP

1. User



ENTITY RELATIONSHIP DIAGRAM (ERD)



TASK SHEET REVIEW 2

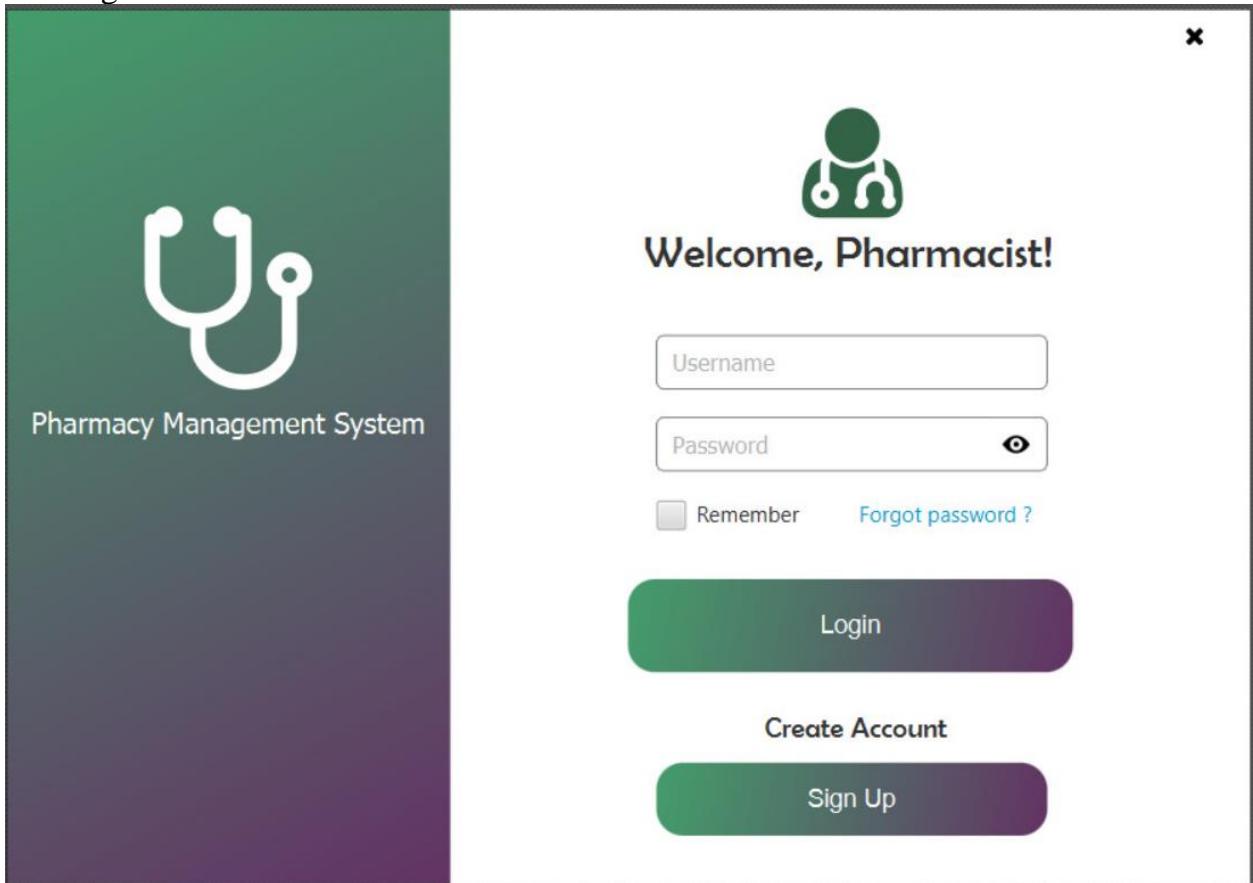
Project: Pharmacy			Date of preparation of activity plan:			
#	Task	Prepared by	Start date	Actuals Date	Team member name	status
1	Use case	Hoang Gia Huy	31/05/2024	07/06/2024	Hoang Gia Huy &	completed
2	Data Flow Diagram (DFD)	& Nguyen Anh Minh	31/05/2024	07/06/2024	Nguyen Anh Minh &	completed
3	Site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

REVIEW 3

GUI DESIGN

1. User side

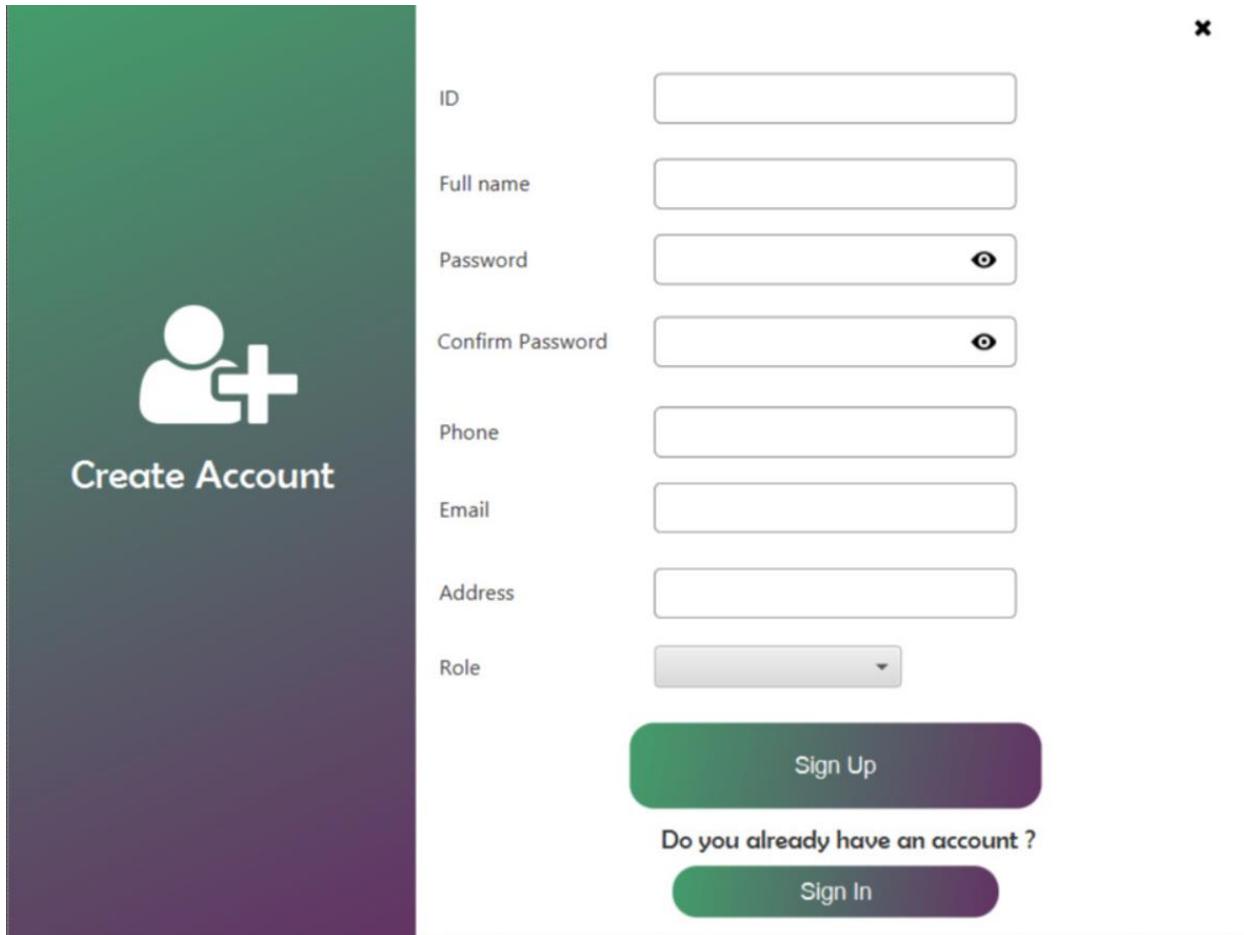
1.1. Login form



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



The image shows a mobile application's registration screen. On the left is a dark green gradient background featuring a white user icon with a plus sign and the text "Create Account". To the right is a white form area with input fields and buttons.

ID	<input type="text"/>
Full name	<input type="text"/>
Password	<input type="password"/> (eye icon)
Confirm Password	<input type="password"/> (eye icon)
Phone	<input type="text"/>
Email	<input type="text"/>
Address	<input type="text"/>
Role	<input type="button" value="▼"/>
Sign Up	
Do you already have an account ?	
Sign In	

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

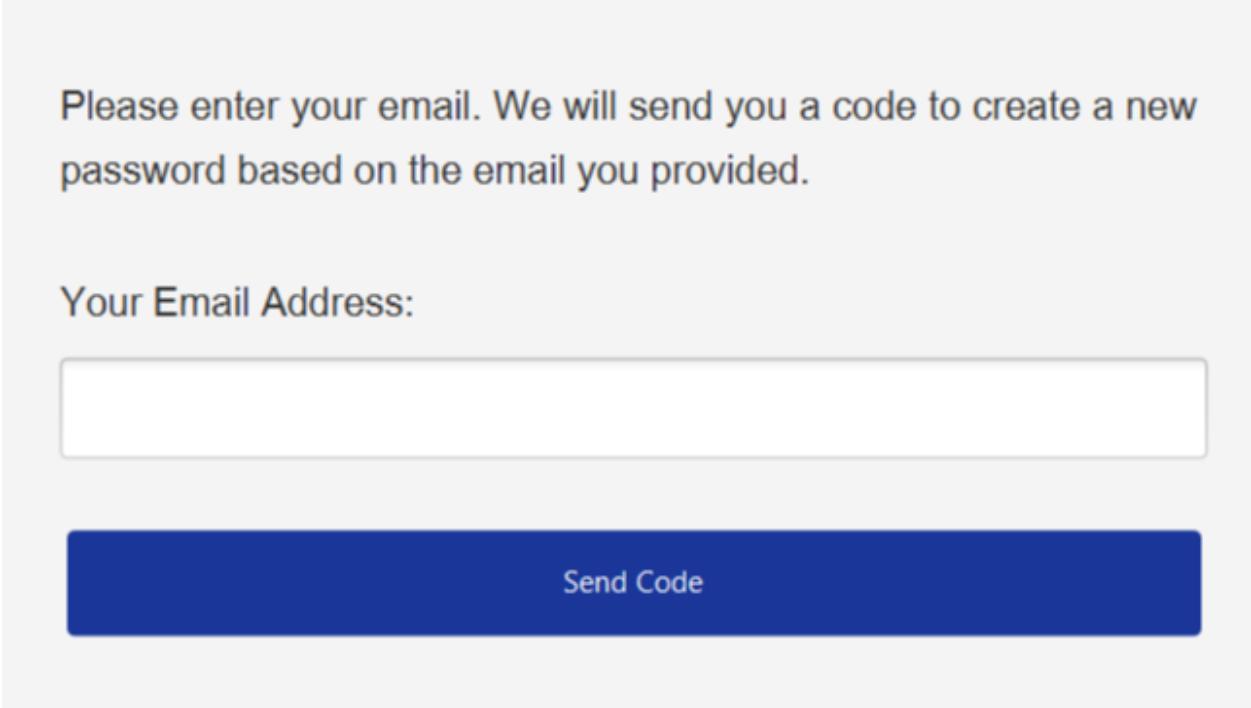
 Validate: “Password is required”, “Wrong password!”

+ 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



The image shows a user interface for forgot password. At the top, there is a message: "Please enter your email. We will send you a code to create a new password based on the email you provided." Below this is a text input field labeled "Your Email Address:" followed by a blue button labeled "Send Code".

Please enter your email. We will send you a code to create a new password based on the email you provided.

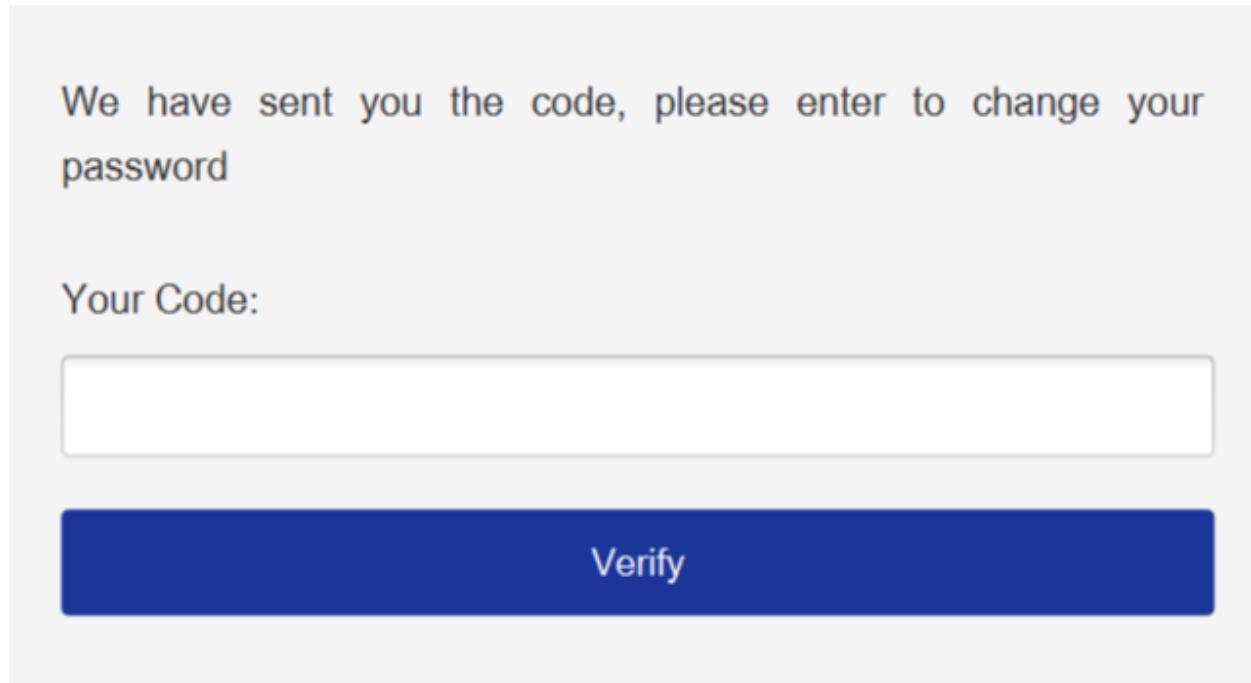
Your Email Address:

Send Code

+ 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

Enter your new password below:

New Password

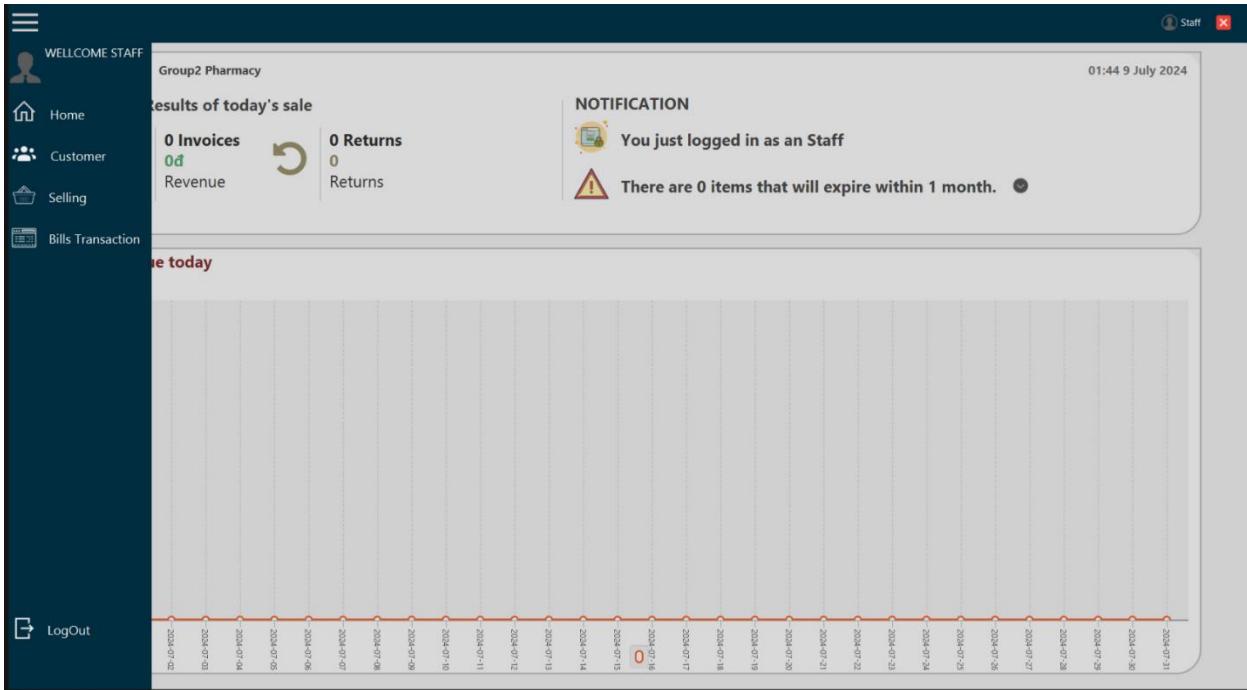
New Password (Confirmation)

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

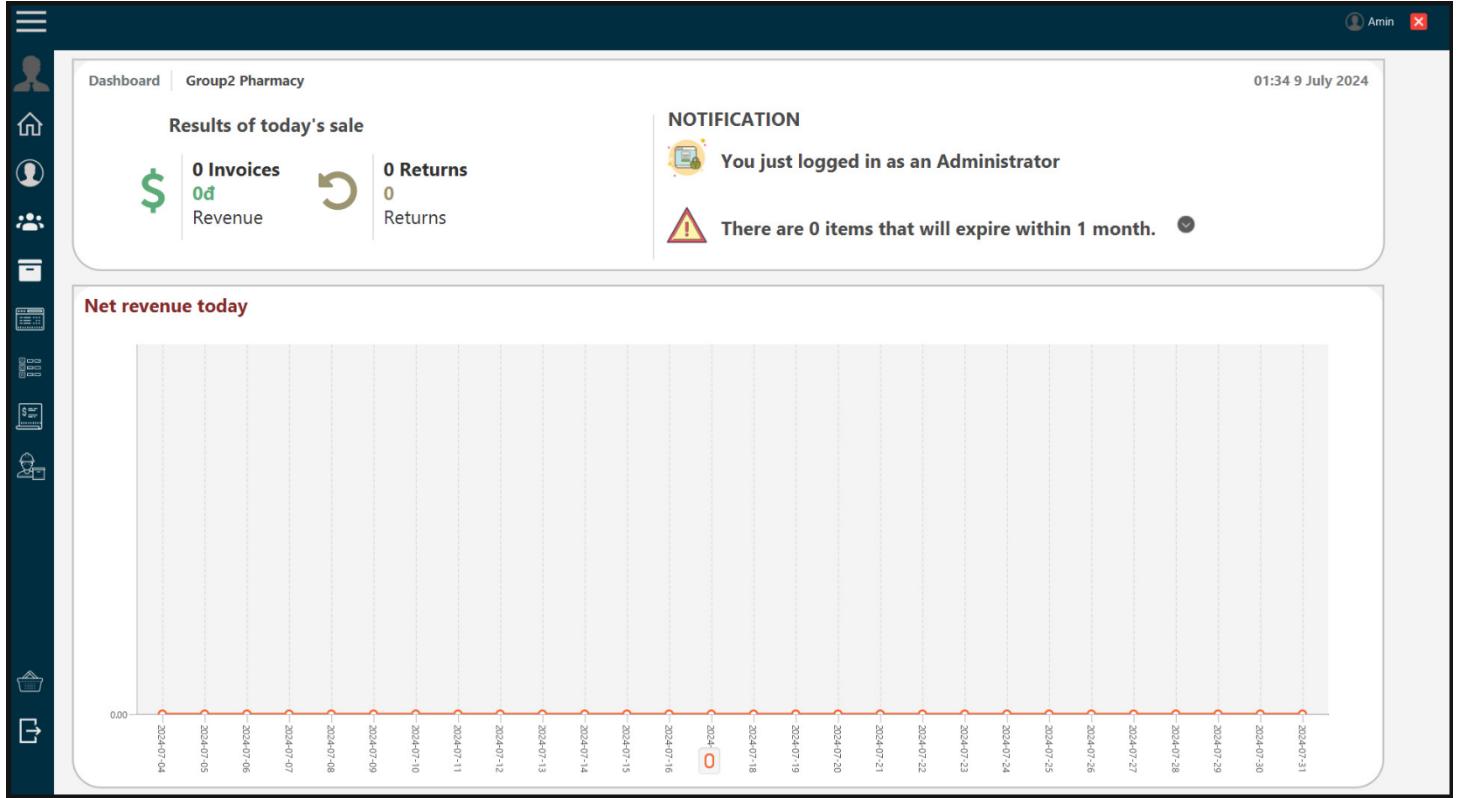
1.6.1. Dashboard for Staff:



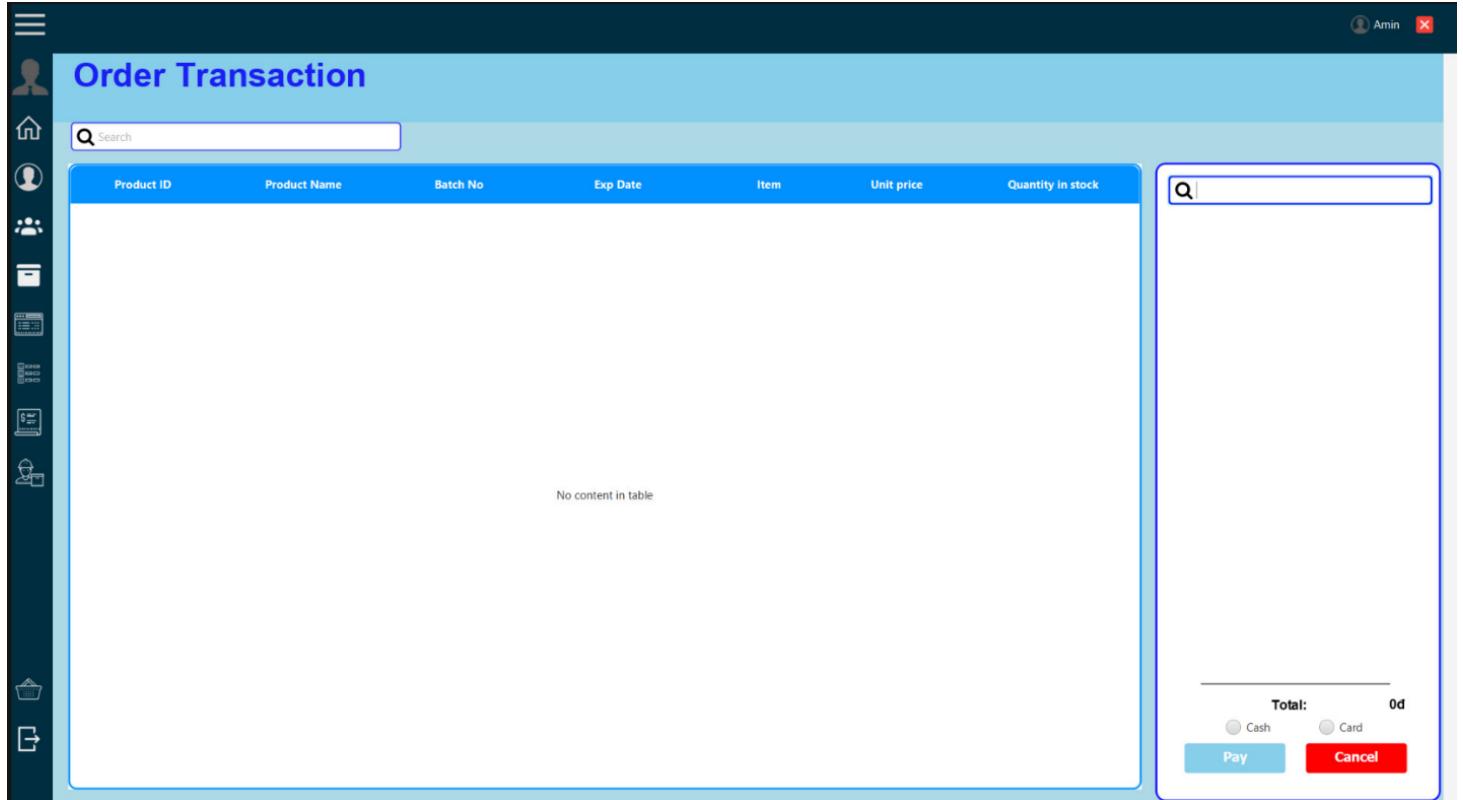
The dashboard page displays:

- 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:



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

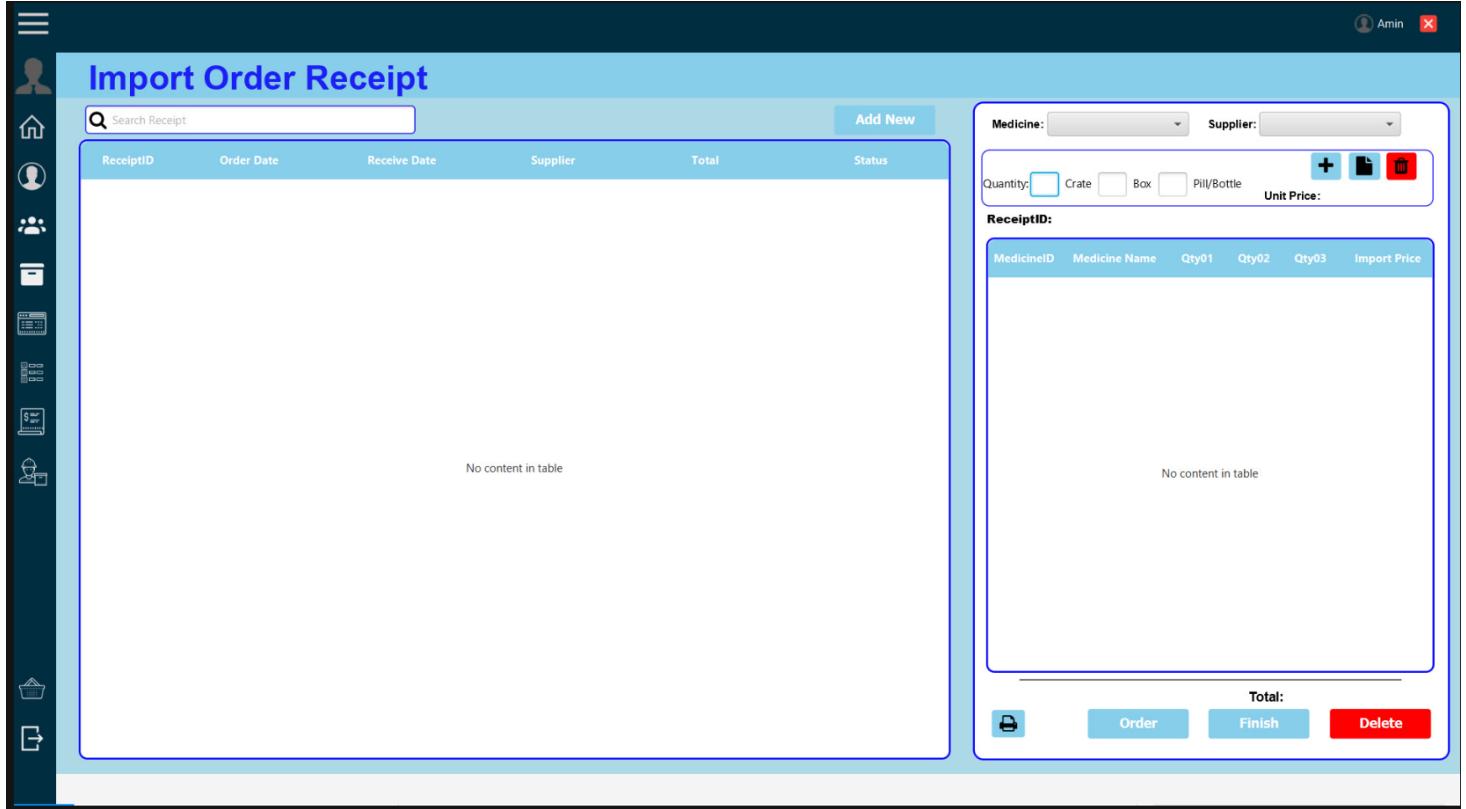
Integer Pill

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

The screenshot shows a software application window titled "Bills Transaction". On the left, there is a vertical toolbar with various icons. The main area has a search bar at the top and a table below it. The table has columns: Bills ID, Customer Name, Phone, Total Price, Payment Method, and Bills Date. A message "No content in table" is displayed in the center of the table area. To the right of the table is a panel titled "Bills Details :" which contains a "Total:" label. At the top right of the window, there is a "Admin" user icon.

Search: + Search bills 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

Integer Batch No with can have a device support to scan

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

3.1. Supplier

The screenshot shows a web application interface for managing suppliers. On the left is a vertical sidebar with icons for navigation. The main header says "Supplier" and includes a search bar and a "Add New Supplier" button. Below the header is a table with columns: ID, Supplier Name, Address, Phone, and Email. A message "No content in table" is displayed in the center of the table area. At the bottom are navigation buttons for pages 1 through 5.

Search:

Table shows up full information:

Search product by Name shown in table below

ID

Supplier Name

Address

Phone

Email

4.1. Category

The screenshot shows a web-based application for managing product categories. On the left is a vertical sidebar with icons for navigation. The main area is titled "Category" and contains a table with the following columns: ID, Medicine Name, API, Quantity, Unit, Subunits Per Unit, Subunit, Items per subunit, Item, Cost Price, Selling Price, and Action. At the top right of the table are buttons for "+ Add New Product" and "Export". A search bar is located at the top left. Below the table, it says "No content in table". At the bottom, there are navigation buttons (1/20) and a footer.

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

Drug Information

Drug Name	<input type="text"/>	Cost price	<input type="text"/> VND
API	<input type="text"/> Active Pharmaceutical Ingredient		
Quantity	<input type="text"/>	More Unit	<input type="text"/>
Unit	<input type="text"/>	<input type="button" value="≈"/>	<input type="text"/>
	<input type="text"/>	<input type="button" value="≈"/>	<input type="text"/>
	<input type="text"/>	<input type="button" value="≈"/>	<input type="text"/>

4.2. Customer Management

The screenshot shows a web-based application for managing customers. On the left, there is a vertical sidebar with various icons for navigation. The main header says "Customer". Below the header is a search bar labeled "SEARCH". To the right of the search bar are two buttons: "+ Add New Customer" and "Export". The main area contains a table with columns: ID, Customer Name, Address, Gender, Phone, Email, Point, and Action. A message "No content in table" is displayed in the center of the table area. At the bottom, there is a pagination control with buttons for page numbers 1 through 5 and a "next" button, followed by the text "1/20".

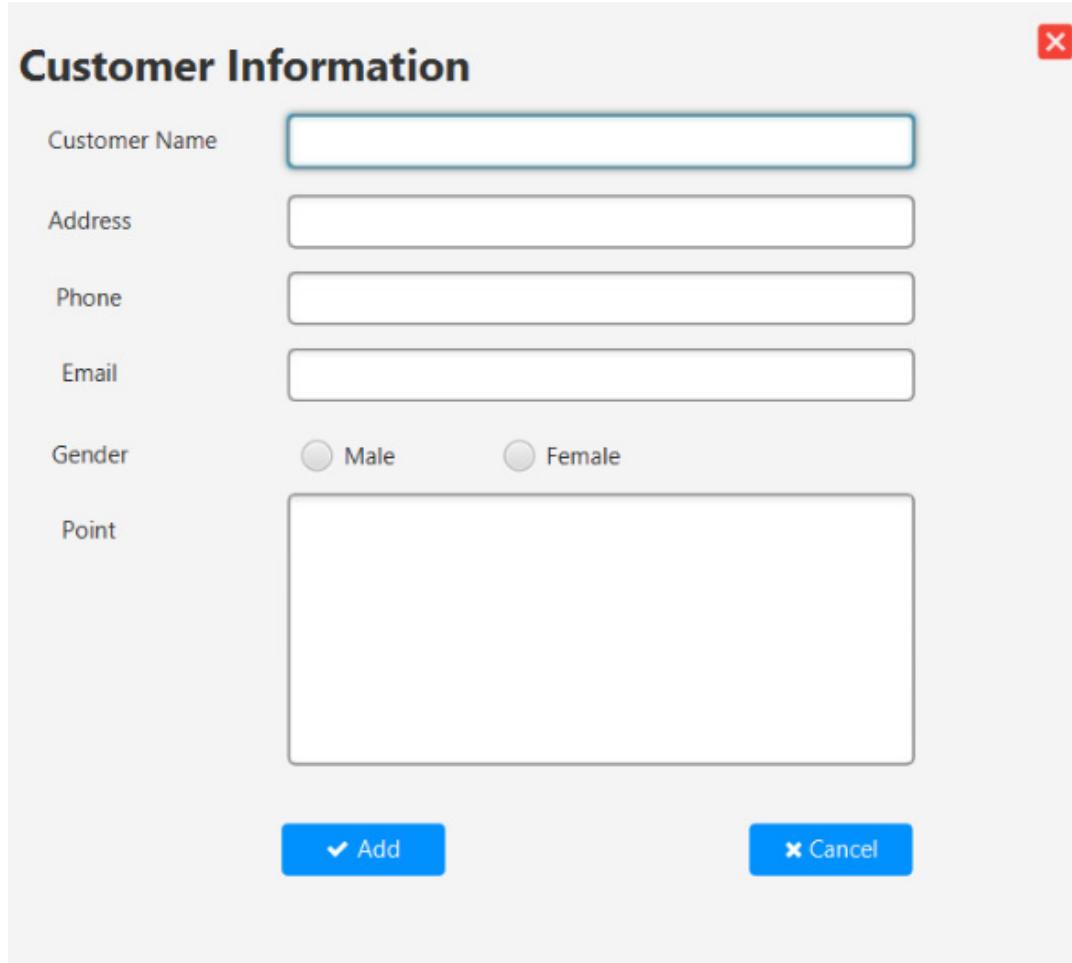
Search:

Add New Customer Button:

Search product by Name shown in table below

Click to go to Add new form

2.7.1.And new customer



The image shows a 'Customer Information' form with the following fields:

- Customer Name: A text input field.
- Address: A text input field.
- Phone: A text input field.
- Email: A text input field.
- Gender: A choice field with radio buttons for Male and Female.
- Note: A large text input field labeled 'Point'.

At the bottom are two buttons: a blue 'Add' button with a checkmark icon and a blue 'Cancel' button with a red 'X' icon.

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

The screenshot shows a web-based application for managing staff. On the left is a vertical sidebar with icons for home, user profile, staff, payroll, timekeeping, products, categories, and import/export. The main header says "Staff" and has a search bar and an "Export" button. Below the header is a table with columns: Staff ID, Staff Name, Gender, NumberPhone, Address, Gmail, Birthday, Role, and Status. A message "No content in table" is displayed. Underneath the table is a section titled "Staff Information:" containing input fields for Staff ID, Password, Address, Birthday, Staff Name, NumberPhone, Gmail, Role, Gender (Male/Female), Status (Active/Inactive), and CardNumber. At the bottom are buttons for "+ Add" (green), "Update" (blue), "Delete" (pink), "Reset" (grey), and "Select Image" (with a placeholder image box).

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:

Click to go Export page

Export Button:

Timekeeping Button: Click to go Timekeeping page

Timekeeping

Staff Button:

Click to go Staff page

Add Payroll information

Staff ID	<input type="text"/>
PayPeriod Start	<input type="text"/> <input type="button" value="Calendar"/>
PayPeriod End	<input type="text"/> <input type="button" value="Calendar"/>
Standard Days	<input type="text"/>
TotalWork Days	<input type="text"/>
Salary	<input type="text"/>
Overtime	<input type="text"/>

Add **Cancel**

4.5. Time Keeping Page

The screenshot shows a web-based application interface titled "Timekeeping". On the left, there is a vertical sidebar with various icons: Home, User, Staff, Payroll, Staff, Calendar, Print, and Log Out. The main area has a header with a search bar and two buttons: "Check in" and "Export". Below the header is a table with the following columns: ID, Staff ID, Staff Name, Day, Check in, and Check out. A message "No content in table" is centered in the table area.

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

Check in with Code ID

Staff ID :

Staff Name :

Code ID :

Check in

Cancel

4.6. Account Management

The screenshot shows a web-based application interface for managing accounts. On the left is a vertical sidebar with various icons for navigation. The main area has a dark header bar with the title "Account". Below the header is a search bar labeled "SEARCH". The main content is a table with the following columns: Full Name, Phone Number, Password, Address, Email, Role, Date Created, and Action. A single row of data is visible, representing a user named "nhattinh" with the following details: Phone Number 0968488430, Password 123456789, Address HCM, Email nhattinh@gmail.com, Role Manager, Date Created 2024-07-09T14:51:58.593, and Action buttons for Update and Delete. At the bottom of the table is a page navigation bar showing "1/1".

Full Name	Phone Number	Password	Address	Email	Role	Date Created	Action
nhattinh	0968488430	123456789	HCM	nhattinh@gmail.com	Manager	2024-07-09T14:51:58.593	<button>Update</button> <button>Delete</button>

Search:

Search product by Name/Phone/Email shown in table below

Update Button:

Click to Update data selected

Delete Button:

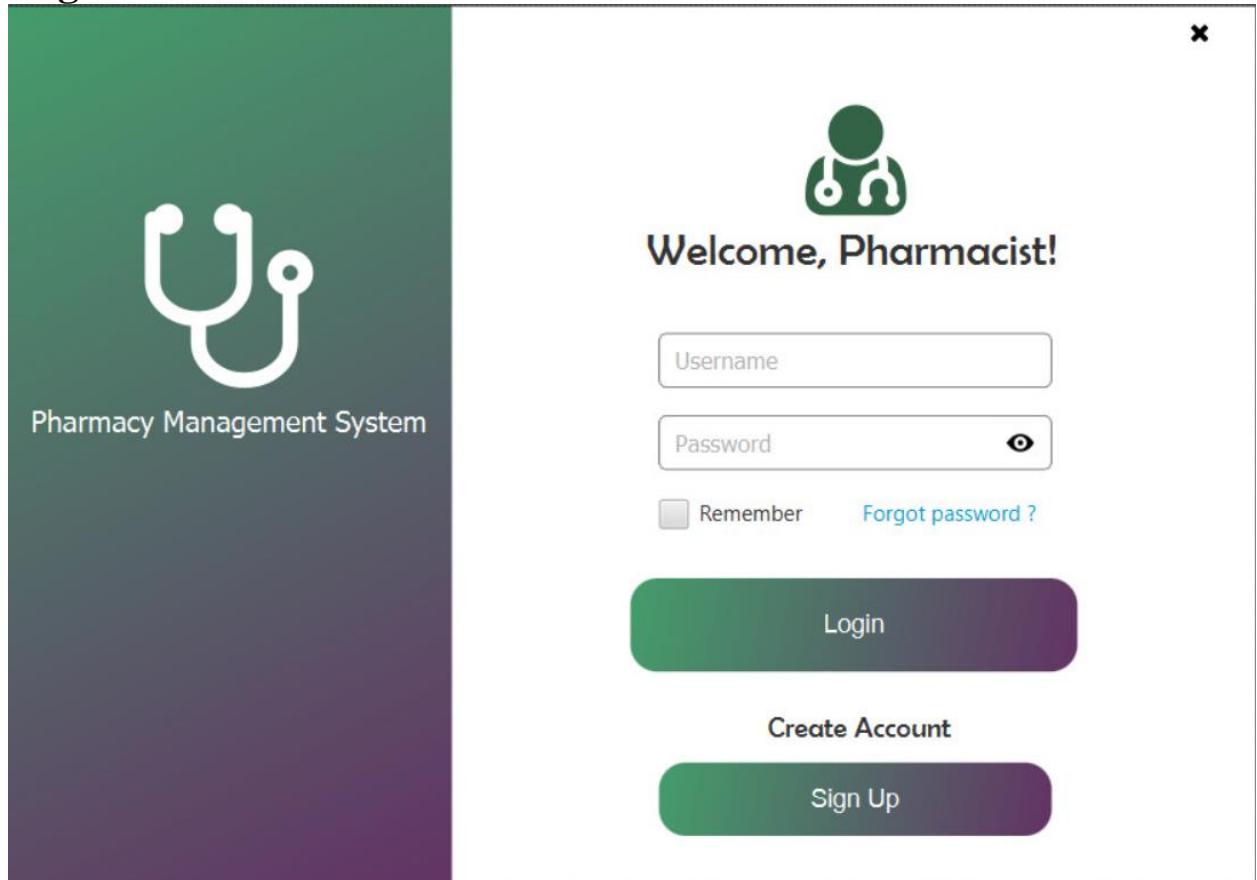
Click to delete data selected

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

```
1 public class LoginController implements Initializable {
2     private FXMLNhanVienController nhanvienController;
3     private FXMLMainController mainController;
4     private FXMLStaffController staffController;
5
6     public void setMainController(FXMLMainController mainController) {
7         this.mainController = mainController;
8     }
9     @FXML
10    private TextField username;
11
12    @FXML
13    private Button btnlogin;
14    @FXML
15    private Button btnsignup;
16
17    @FXML
18    private Button btnclose;
19    @FXML
20    private CheckBox ckremember;
21    @FXML
22    private ImageView EYE;
23
24    @FXML
25    private ImageView EYE_SLASH;
26    @FXML
27    private Button btnEYE;
28
29    @FXML
30    private Button btnEYE_SLASH;
31    @FXML
32    private PasswordField passwordField;
33
34    @FXML
35    private TextField showpasswordField;
36    @FXML
37    private Hyperlink forgotpassword;
38    private boolean isPasswordVisible = false;
39 //=====
40 // Connect Database
41 /**
42  *
43  */
44
45    private Connection conn;
46    private PreparedStatement prepare;
47    private ResultSet rs;
48
49 //=====
50    private static final String UPDATE_RMBPASSWORD_SQL = "UPDATE Login SET RmbPassword = ? WHERE Username = ?";
51    private static final String SELECT_RMBPASSWORD_SQL = "SELECT RmbPassword FROM Login WHERE Username = ?";
52
```

```
1 public LoginController() {
2     this.conn = DB.ConnectDB.getConnectDB();
3 }
4
5 @Override
6 public void initialize(URL url, ResourceBundle rb) {
7
8     btnclose.setOnAction(event -> btnclose());
9     //Show/Hide Password
10    btnEYE.setOnAction(event -> togglePasswordVisibility());
11    btnEYE_SLASH.setOnAction(event -> togglePasswordVisibility());
12    EYE.setOnMouseClicked(event -> togglePasswordVisibility());
13    EYE_SLASH.setOnMouseClicked(event -> togglePasswordVisibility());
14    updatePasswordVisibility();
15    forgotpassword.setOnAction(event -> ForgotPassword());
16
17    btnsignup.setOnAction(event -> CreateAdminAccount());
18    btnlogin.setOnAction(event -> {
19        try {
20            login();
21        } catch (IOException | SQLException ex) { // Print the exception for debugging purposes
22            // Print the exception for debugging purposes
23            AlertHelper.errorMessage("Error during login: " + ex.getMessage());
24        }
25    });
26
27    passwordField.addEventFilter(MouseEvent.MOUSE_CLICKED, event -> {
28        loadPassword();
29    });
30
31    passwordField.focusedProperty().addListener((observable, oldValue, newValue) -> {
32        if (newValue) { // Khi focus được nhận
33            loadPassword();
34            resetRemember();
35        }
36    });
37
38    showpasswordField.textProperty().addListener((ObservableValue<? extends String> observable, String oldValue, String newValue) -> {
39        if (passwordField.getText() == null || !passwordField.getText().equals(newValue)) {
40            passwordField.setText(newValue);
41        }
42    });
43 }
44 private String fullName = "";
```

```

1 public void login() throws IOException, SQLException {
2     try {
3         if (username.getText().isEmpty()) {
4             AlertHelper.errorMessage("Username cannot be empty");
5             return;
6         }
7
8         if (passwordField.getText().isEmpty()) {
9             AlertHelper.errorMessage("Password cannot be empty");
10            return;
11        }
12
13        // Verify username and password
14        String selectLogin = "SELECT Username FROM Login WHERE Username = ? AND Password = ?";
15        conn = DB.ConnectDB.getConnectDB();
16        prepare = conn.prepareStatement(selectLogin);
17        prepare.setString(1, username.getText());
18        prepare.setString(2, passwordField.getText());
19        rs = prepare.executeQuery();
20
21        if (rs.next()) {
22            // Username and password match, now fetch the role
23            String selectRole = "SELECT Role FROM Role WHERE Username = ?";
24            prepare = conn.prepareStatement(selectRole);
25            prepare.setString(1, username.getText());
26            rs = prepare.executeQuery();
27
28            if (rs.next()) {
29                String role = rs.getString("Role");
30                String fxmlFile = "";
31
32                if ("Manager".equals(role)) {
33                    if ("Staff".equals(role)) {
34                        fxmlFile = "/sam/FXMLStaff.fxml";
35                    } else {
36                        AlertHelper.errorMessage("Unknown role: " + role);
37                        return;
38                    }
39                } else {
40                    fxmlFile = "/sam/FXMLMain.fxml";
41                }
42
43                // lấy thông tin username
44                String selectUserInfo = "SELECT Fullname FROM Information WHERE Phone = ?";
45                prepare = conn.prepareStatement(selectUserInfo);
46                prepare.setString(1, username.getText());
47                rs = prepare.executeQuery();
48
49                if (rs.next()) {
50                    fullName = rs.getString("Fullname");
51                }
52
53                if (ckremember.isSelected()) {
54                    savePassword(username.getText(), passwordField.getText());
55                } else {
56                    clearRMPassword(username.getText());
57                }
58
59                FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));
60                Parent root = loader.load();
61                Scene scene = new Scene(root);
62                String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
63                scene.getStylesheets().add(cssPath);
64
65                Stage stage = (Stage) username.getScene().getWindow(); // Get the current stage
66
67                stage.setResizable(true);
68
69                if("Staff".equals(role)){
70                    FXMLStaffController staffController = loader.getController();
71                    staffController.setFullName(fullName);
72
73                }else{
74                    FXMLMainController mainController = loader.getController();
75                    mainController.setFullName(fullName);
76
77                Sam appInstance = new Sam();
78                appInstance.makeWindowDraggable(stage, root);
79
80                stage.setScene(scene);
81                stage.centerOnScreen();
82                stage.show();
83            } else {
84                AlertHelper.errorMessage("Role not found for the username");
85            }
86        } else {
87            AlertHelper.errorMessage("Incorrect username or password");
88        }
89    } catch (SQLException e) {
90        AlertHelper.errorMessage("Database error: " + e.getMessage());
91    } finally {
92        closeResources(rs, prepare, conn);
93    }
}

```

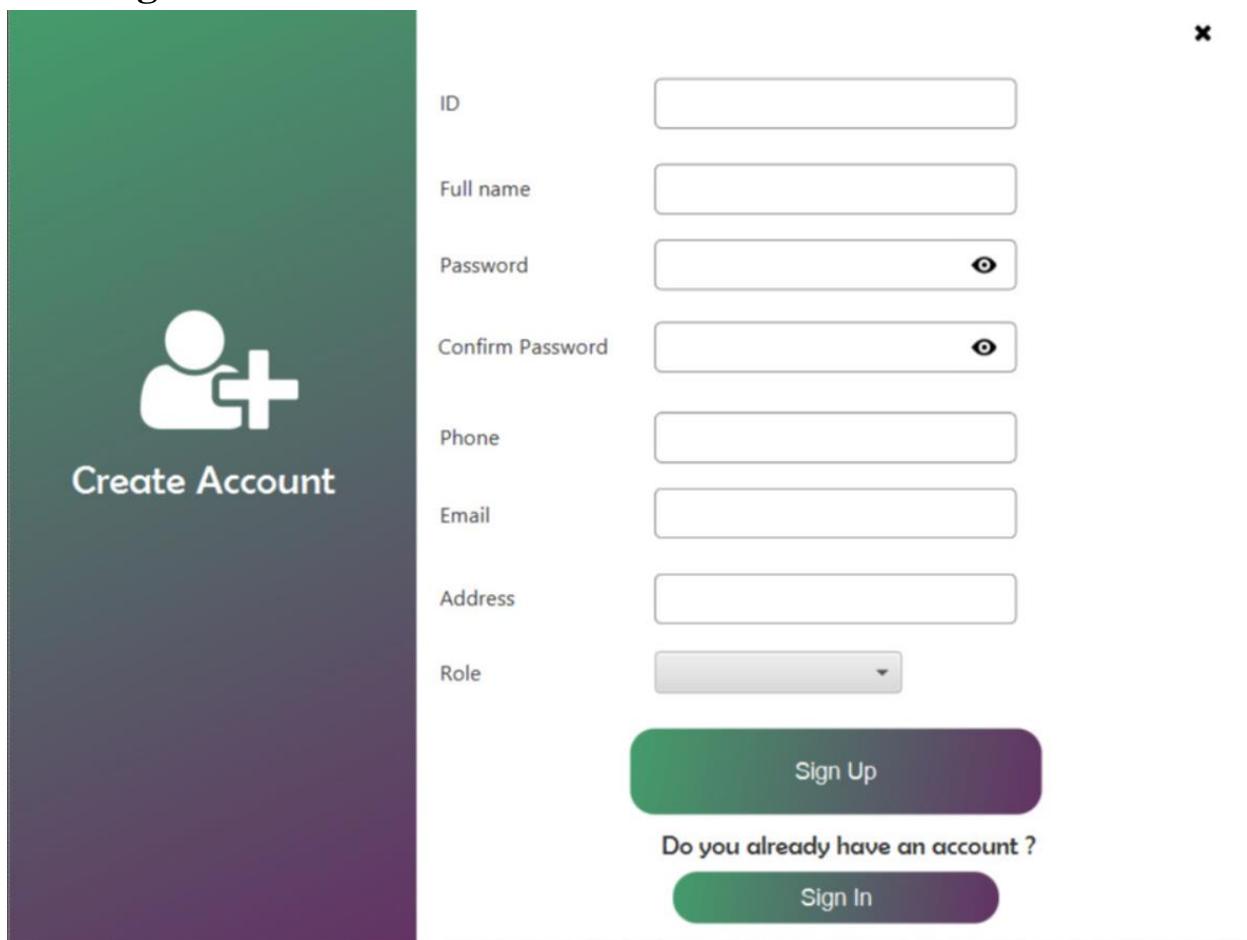
```
1  private void closeResources(ResultSet rs, PreparedStatement prepare, Connection conn) {
2      try {
3          if (rs != null) {
4              rs.close();
5          }
6          if (prepare != null) {
7              prepare.close();
8          }
9          if (conn != null) {
10              conn.close();
11          }
12      } catch (SQLException e) {
13      }
14  }
15
16 // RMP Function
17 //
18 private void resetRemember() {
19     if (isPasswordRemembered()) {
20         ckremember.setSelected(true);
21     } else {
22         ckremember.setSelected(false);
23     }
24 }
25
26 private boolean isPasswordRemembered() {
27     try (Connection conn = DB.ConnectDB.getConnectDB()) {
28         if (conn == null) {
29             System.out.println("Connection is null. Cannot check password.");
30             return false;
31         }
32
33         String usernameText = username.getText();
34         if (usernameText == null || usernameText.isEmpty()) {
35             System.out.println("Username field is empty.");
36             return false;
37         }
38
39         try (PreparedStatement selectStmt = conn.prepareStatement(SELECT_RMBPASSWORD_SQL)) {
40             selectStmt.setString(1, usernameText);
41             try (ResultSet rs = selectStmt.executeQuery()) {
42                 if (rs.next()) {
43                     String rmbPasswordText = rs.getString("RmbPassword");
44                     return rmbPasswordText != null && !rmbPasswordText.isEmpty();
45                 }
46             }
47         } catch (SQLException e) {
48             System.err.println("SQL error: " + e.getMessage());
49         }
50     }
51     return false;
52 }
53
54 private void savePassword(String username, String password) {
55     try (Connection conn = DB.ConnectDB.getConnectDB()) {
56         if (conn == null) {
57             System.out.println("Connection is null. Cannot save password.");
58             return;
59         }
60
61         String query = UPDATE_RMBPASSWORD_SQL;
62         try (PreparedStatement stmt = conn.prepareStatement(query)) {
63             stmt.setString(1, password);
64             stmt.setString(2, username);
65             int rowsUpdated = stmt.executeUpdate();
66             if (rowsUpdated > 0) {
67                 System.out.println("Password remembered successfully.");
68             } else {
69                 System.out.println("No user found with the specified username.");
70             }
71         } catch (SQLException e) {
72             System.out.println("SQL Exception: " + e.getMessage());
73         }
74     }
75 }
```

```
1  private void clearRMPassword(String username) {
2      try (Connection conn = DB.ConnectDB.getConnectDB()) {
3          if (conn == null) {
4              System.out.println("Connection is null. Cannot clear password.");
5              return;
6          }
7
8          String query = "UPDATE Login SET RmbPassword = NULL WHERE Username = ?";
9          try (PreparedStatement stmt = conn.prepareStatement(query)) {
10              stmt.setString(1, username);
11              int rowsUpdated = stmt.executeUpdate();
12              if (rowsUpdated > 0) {
13                  System.out.println("Password cleared successfully.");
14              } else {
15                  System.out.println("No user found with the specified username.");
16              }
17          }
18      } catch (SQLException e) {
19          System.out.println("SQL Exception: " + e.getMessage());
20      }
21  }
22
23  private void loadPassword() {
24      try (Connection conn = DB.ConnectDB.getConnectDB()) {
25          if (conn == null) {
26              System.out.println("Connection is null. Cannot load password.");
27              return;
28          }
29
30          String usernameText = username.getText();
31          if (usernameText == null || usernameText.isEmpty()) {
32              System.out.println("Username field is empty.");
33              return;
34          }
35
36          try (PreparedStatement selectStmt = conn.prepareStatement(SELECT_RMBPASSWORD_SQL)) {
37              selectStmt.setString(1, usernameText);
38              try (ResultSet rs = selectStmt.executeQuery()) {
39                  if (rs.next()) {
40                      String rmbPasswordText = rs.getString("rmbpassword");
41                      passwordField.setText(rmbPasswordText);
42                      showpasswordField.setText(rmbPasswordText);
43                      ckremember.setSelected(true);
44                  }
45              }
46          } catch (SQLException e) {
47              System.err.println("SQL error: " + e.getMessage());
48          }
49      }
50  }
51
52 // End RMP Function
53 // Show Password
54 @FXML
55 public void togglePasswordVisibility() {
56     isPasswordVisible = !isPasswordVisible;
57     updatePasswordVisibility();
58 }
59
60 public void updatePasswordVisibility() {
61     if (isPasswordVisible) {
62         // Show password
63         passwordField.setVisible(false);
64         showpasswordField.setVisible(true);
65         showpasswordField.setText(passwordField.getText());
66         btnEYE.setVisible(false);
67         btnEYE_SLASH.setVisible(true);
68     } else {
69         // Hide password
70         passwordField.setVisible(true);
71         showpasswordField.setVisible(false);
72         btnEYE.setVisible(true);
73         btnEYE_SLASH.setVisible(false);
74     }
75 }
76 }
```

```
1 //End Show Password
2     @FXML
3     public void ForgotPassword() {
4         try {
5             // Tạo một FXMLLoader để tải tệp FXML
6             FXMLLoader loader = new FXMLLoader(getClass().getResource("/ForgotPassword/ForgotPassword.fxml"));
7
8             // Tải tệp FXML và tạo một đối tượng Parent
9             Parent root = loader.load();
10
11            // Tạo một Scene mới từ đối tượng Parent
12            Scene scene = new Scene(root);
13
14
15            // Tạo một Stage mới
16            Stage Stageforgot = new Stage();
17            Stageforgot.setTitle("Forgot Password"); // Tiêu đề của cửa sổ mới
18            Stageforgot.setScene(scene);
19            Stageforgot.initStyle(StageStyle.UNDECORATED);
20
21            // Đặt kiểu Modality cho Stage mới nếu bạn muốn khóa giao diện cũ cho đến khi giao diện mới được đóng
22            Stageforgot.initModality(Modality.APPLICATION_MODAL);
23
24            // Hiển thị Stage mới
25            Stageforgot.show();
26        } catch (IOException e) {
27
28    }
29}
30
31     @FXML
32     public void CreateAdminAccount() {
33         try {
34             FXMLLoader loader;
35             loader = new FXMLLoader(getClass().getResource("/Register/Register.fxml"));
36             Parent root = loader.load();
37             Scene scene = new Scene(root);
38             Stage stage = (Stage) username.getScene().getWindow();
39             stage.setScene(scene);
40             stage.show();
41         } catch (IOException e) {
42
43    }
44}
45
46     public void btnclose() {
47         System.exit(0);
48    }
49
50 }
```

```
1  /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5  package Login;
6
7  import java.sql.Connection;
8  import java.sql.PreparedStatement;
9  import java.sql.ResultSet;
10 import java.sql.SQLException;
11
12 /**
13 *
14 * @author huypg
15 */
16 public class Validate {
17     // Phương thức để kiểm tra xem tài khoản có tồn tại không
18     public boolean checkAccountExists(String username) {
19         String sql = "SELECT COUNT(*) FROM users WHERE username = ?";
20         try {
21             Connection conn = DB.ConnectDB.getConnectDB();
22             PreparedStatement stmt = conn.prepareStatement(sql)
23         } {
24             stmt.setString(1, username);
25             ResultSet rs = stmt.executeQuery();
26             if (rs.next()) {
27                 int count = rs.getInt(1);
28                 return count > 0;
29             }
30         } catch (SQLException e) {
31         }
32         return false;
33     }
34
35     // Phương thức để kiểm tra tính chính xác của mật khẩu
36     public boolean checkPassword(String username, String password) {
37         String sql = "SELECT password FROM users WHERE username = ?";
38         try {
39             Connection conn = DB.ConnectDB.getConnectDB();
40             PreparedStatement stmt = conn.prepareStatement(sql)
41         } {
42             stmt.setString(1, username);
43             ResultSet rs = stmt.executeQuery();
44             if (rs.next()) {
45                 String storedPassword = rs.getString("password");
46                 return storedPassword.equals(password);
47             }
48         } catch (SQLException e) {
49         }
50         return false;
51     }
52 }
53
54
```

2 Register



The image shows a registration form titled "Create Account" on a green-to-purple gradient background. The form includes fields for ID, Full name, Password, Confirm Password, Phone, Email, Address, and Role. It features a "Sign Up" button and links for existing users to "Sign In".

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

```
1 @FXML  
2     private ImageView EYE;  
3  
4     @FXML  
5     private ImageView EYE_SLASH;  
6  
7     @FXML  
8     private Button btnEYE;  
9  
10    @FXML  
11    private Button btnEYE_SLASH;  
12  
13    @FXML  
14    private ImageView EYECF;  
15  
16    @FXML  
17    private ImageView EYE_SLASHCF;  
18  
19    @FXML  
20    private Button btnEYECF;  
21  
22    @FXML  
23    private Button btnEYE_SLASHCF;  
24  
25    @FXML  
26    private Button btnSignin;  
27  
28    @FXML  
29    private Button btnSignup;  
30  
31    @FXML  
32    private ComboBox<String> cbbRole;  
33  
34    @FXML  
35    private Button close;
```

```
1 @FXML  
2     private TextField txtFullscreen;  
3  
4     @FXML  
5     private TextField txtAddress;  
6  
7     @FXML  
8     private TextField txtEmail;  
9  
10    @FXML  
11    private PasswordField txtPassword;  
12  
13    @FXML  
14    private PasswordField txtCFPassword;  
15  
16    @FXML  
17    private TextField txtShowPassword;  
18  
19    @FXML  
20    private TextField txtshowCFPassword;  
21  
22    @FXML  
23    private TextField txtPhone;  
24  
25    @FXML  
26    private TextField txtID;  
27  
28    @FXML  
29  
30    private boolean isPasswordVisible = false;  
31    private boolean isCFPasswordVisible = false;  
32  
33    private Connection conn;  
34    private PreparedStatement prepare;
```

2.2 Code



The screenshot shows a Java code editor with a dark theme. The code is a Java class named `ValidateRegister`. It includes methods for initializing database connections, showing alerts, and validating user input. It also contains static methods for checking if a string is empty, if a password is confirmed, if it's numeric, and if an email is valid. The code uses annotations like `@Override`, `@FXML`, and `@FXML` for event handlers. The code is well-organized with proper indentation and comments.

```
1  @Override
2  public void initialize(URL url, ResourceBundle rb) {
3      conn = DB.ConnectDB.getConnectDB();
4      cbbRole.getItems().addAll("Staff", "Manager");
5      cbbRole.setValue("Staff");
6      btnSignup.setOnAction(event -> handleRegisterButton());
7      btnsignin.setOnAction(event -> ReturntoLogin());
8      close.setOnAction(event -> handleCloseButton());
9      //Password
10     btnEYE.setOnAction(event -> togglePasswordVisibility());
11     btnEYE_SLASH.setOnAction(event -> togglePasswordVisibility());
12     EVE.setOnMouseClicked(event -> togglePasswordVisibility());
13     EVE_SLASH.setOnMouseClicked(event -> togglePasswordVisibility());
14     updatePasswordVisibility();
15     //Confirm Password
16     btnEYECF.setOnAction(event -> toggleCFPVisibility());
17     btnEYE_SLASHCF.setOnAction(event -> toggleCFPVisibility());
18     EVECF.setOnMouseClicked(event -> toggleCFPVisibility());
19     EVE_SLASHCF.setOnMouseClicked(event -> toggleCFPVisibility());
20     updateCFPVisibility();
21 }
22
23 private void showAlert(String title, String message) {
24     Alert alert = new Alert(AlertType.INFORMATION);
25     alert.setTitle(title);
26     alert.setHeaderText(null);
27     alert.setContentText(message);
28     alert.showAndWait();
29 }
30
31 public class ValidateRegister {
32
33     public static boolean isEmpty(String input) {
34         return input != null && !input.trim().isEmpty();
35     }
36
37     public static boolean isPasswordConfirmed(String password, String confirmPassword) {
38         return isEmpty(password) && password.equals(confirmPassword);
39     }
40
41     public static boolean isNumeric(String input) {
42         return input != null && input.matches("\\d+");
43     }
44
45     public static boolean isValidEmail(String email) {
46         String emailRegex = "[A-Za-z0-9.%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$";
47         Pattern pat = Pattern.compile(emailRegex);
48         return email != null && pat.matcher(email).matches();
49     }
50 }
51
52 @FXML
53 private void handleRegisterButton() {
54     saveUserData();
55 }
56
57 @FXML
58 private void handleCloseButton() {
59     System.exit(0);
60 }
61
62 public void ReturntoLogin() {
63     try {
64         FXMLLoader loader = new FXMLLoader(getClass().getResource("/Login/Login.fxml"));
65         Parent root = loader.load();
66         Scene scene = new Scene(root);
67         Stage stage = (Stage) close.getScene().getWindow();
68         stage.setScene(scene);
69         stage.show();
70     } catch (IOException e) {
71     }
72 }
73 }
```

```
1 public void saveUserData() {
2     String password = txtPassword.getText();
3     String email = txtEmail.getText();
4     String phone = txtPhone.getText();
5     String address = txtAddress.getText();
6     String fullname = txtFullscreen.getText();
7     String role = cbbRole.getValue(); // Get the selected value from ComboBox
8
9
10    if (!ValidateRegister.isNotEmpty(txtFullscreen.getText())) {
11        showAlert("Validation Error", "Fullname cannot be empty");
12        return;
13    }
14
15    if (!ValidateRegister.isNotEmpty(txtPassword.getText())) {
16        showAlert("Validation Error", "Password cannot be empty");
17        return;
18    }
19
20    if (!ValidateRegister.isPasswordConfirmed(txtPassword.getText(), txtCFPassword.getText())) {
21        showAlert("Validation Error", "Confirm Password must match Password and cannot be empty");
22        return;
23    }
24
25    if (!ValidateRegister.isNotEmpty(txtPhone.getText())) {
26        showAlert("Validation Error", "Phone cannot be empty");
27        return;
28    }
29
30    if (!ValidateRegister.isNumeric(txtPhone.getText())) {
31        showAlert("Validation Error", "Phone must be a number");
32        return;
33    }
34    if (isPhoneAlreadyExists(txtPhone.getText())) {
35        showAlert("Validation Error", "Phone already exists in the database");
36        return;
37    }
38
39
40    if (!ValidateRegister.isValidEmail(txtEmail.getText())) {
41        showAlert("Validation Error", "Email is not valid");
42        return;
43    }
44
45
46    if (isEmailAlreadyExists(txtEmail.getText())) {
47        showAlert("Validation Error", "Email already exists in the database");
48        return;
49    }
50    if (!ValidateRegister.isNotEmpty(txtAddress.getText())) {
51        showAlert("Validation Error", "Address cannot be empty");
52        return;
53    }
```



```

1 // Additional input validation can be added here
2     Timestamp dateCreated = Timestamp.from(Instant.now());
3
4     try {
5         conn = DB.ConnectDB.getConnectDB();
6         // Insert into Login table
7         String sqlLogin = "INSERT INTO Login (Username, Password) VALUES (?, ?)";
8         prepare = conn.prepareStatement(sqlLogin);
9         prepare.setString(1, phone);
10        prepare.setString(2, password);
11        prepare.executeUpdate();
12
13        // Insert into Information table
14        String sqlInformation = "INSERT INTO Information ( Fullname, Phone, Email, Address, Date_Created) VALUES ( ?, ?, ?, ?, ?)";
15        prepare = conn.prepareStatement(sqlInformation);
16        prepare.setString(1, fullname);
17        prepare.setString(2, phone);
18        prepare.setString(3, email);
19        prepare.setString(4, address);
20        prepare.setTimestamp(5, dateCreated);
21        prepare.executeUpdate();
22
23        // Insert into Role table
24        String sqlRole = "INSERT INTO Role (Role, Username) VALUES (?, ?)";
25        prepare = conn.prepareStatement(sqlRole);
26        prepare.setString(1, role);
27        prepare.setString(2, phone);
28        prepare.executeUpdate();
29
30
31        System.out.println("Data saved successfully!");
32
33        // Optionally switch to another scene or give feedback to the user
34        showAlert("Successfully", "Congratulation!!");
35        ReturntoLogin(); // Example function to return to login screen
36
37    } catch (SQLException e) {
38        // Log detailed SQL exception for troubleshooting
39        // Handle SQL exception properly in your application (e.g., show error message to user)
40        System.out.println(""+e.getMessage());
41    } finally {
42        try {
43            if (prepare != null) {
44                prepare.close();
45            }
46            if (conn != null) {
47                conn.close();
48            }
49        } catch (SQLException ex) {
50            // Log detailed exception for resource closing issues
51            // Handle closing exception properly in your application
52        }
53    }
54}
55
56
57 private boolean isIdAlreadyExists(String id) {
58     try (Connection conn = DB.ConnectDB.getConnectDB()) {
59         String sqlCheckId = "SELECT COUNT(*) FROM Information WHERE ID = ?";
60         try (PreparedStatement stmt = conn.prepareStatement(sqlCheckId)) {
61             stmt.setString(1, id);
62             try (ResultSet rs = stmt.executeQuery()) {
63                 if (rs.next()) {
64                     int count = rs.getInt(1);
65                     return count > 0;
66                 }
67             }
68         } catch (SQLException e) {
69             e.printStackTrace();
70         }
71     }
72     return false;
73 }
74
75 private boolean isPhoneAlreadyExists(String phone) {
76     try (Connection conn = DB.ConnectDB.getConnectDB()) {
77         String sqlCheckPhone = "SELECT COUNT(*) FROM Information WHERE Phone = ?";
78         try (PreparedStatement stmt = conn.prepareStatement(sqlCheckPhone)) {
79             stmt.setString(1, phone);
80             try (ResultSet rs = stmt.executeQuery()) {
81                 if (rs.next()) {
82                     int count = rs.getInt(1);
83                     return count > 0;
84                 }
85             }
86         } catch (SQLException e) {
87             e.printStackTrace();
88         }
89     }
90     return false;
91 }
92
93 }
```

```
1  private boolean isEmailAlreadyExists(String email) {
2      try (Connection conn = DB.ConnectDB.getConnectDB()) {
3          String sqlCheckEmail = "SELECT COUNT(*) FROM Information WHERE BINARY Email = ?";
4          try (PreparedStatement stmt = conn.prepareStatement(sqlCheckEmail)) {
5              stmt.setString(1, email);
6              try (ResultSet rs = stmt.executeQuery()) {
7                  if (rs.next()) {
8                      int count = rs.getInt(1);
9                      return count > 0;
10                 }
11             }
12         } catch (SQLException e) {
13     }
14     return false;
15 }
16
17 // Show Password
18 @FXML
19 public void togglePasswordVisibility() {
20     isPasswordVisible = !isPasswordVisible;
21     updatePasswordVisibility();
22 }
23
24
25 public void updatePasswordVisibility() {
26     if (isPasswordVisible) {
27         // Show password
28         txtPassword.setVisible(false);
29         txtShowPassword.setVisible(true);
30         txtShowPassword.setText(txtPassword.getText());
31         btnEYE.setVisible(false);
32         btnEYE_SLASH.setVisible(true);
33     } else {
34         // Hide password
35         txtPassword.setVisible(true);
36         txtShowPassword.setVisible(false);
37         btnEYE.setVisible(true);
38         btnEYE_SLASH.setVisible(false);
39     }
40 }
41
42 //End Show Password
43 // Show Confirm Password
44 @FXML
45 public void toggleCFPVisibility() {
46     isCFPasswordVisible = !isCFPasswordVisible;
47     updateCFPVisibility();
48 }
49
50 public void updateCFPVisibility() {
51     if (isCFPasswordVisible) {
52         // Show password
53         txtCFPassword.setVisible(false);
54         txtshowCFPassword.setVisible(true);
55         txtshowCFPassword.setText(txtCFPassword.getText());
56         btnEYECF.setVisible(false);
57         btnEYE_SLASHCF.setVisible(true);
58     } else {
59         // Hide password
60         txtCFPassword.setVisible(true);
61         txtshowCFPassword.setVisible(false);
62         btnEYECF.setVisible(true);
63         btnEYE_SLASHCF.setVisible(false);
64     }
65 }
66
67 //End Show Confirm Password
68 }
69
```

3 Forgot Password

```
1  public class ForgotPasswordController implements Initializable {
2
3      @FXML
4      private StackPane Fogot;
5
6      @FXML
7      private Button exit;
8      @FXML
9      private Button btnSendCode;
10
11     @FXML
12     private Button btnVerify;
13
14     @FXML
15     private AnchorPane forgotpane;
16
17     @FXML
18     private TextField hide;
19
20     @FXML
21     private TextField txtCode;
22
23     @FXML
24     private TextField txtemail;
25
26     @FXML
27     private AnchorPane verifypane;
28
29     @FXML
30     private AnchorPane resetpane;
31
32     @FXML
33     private ImageView EYE;
34
35     @FXML
36     private ImageView EYECF;
37
38     @FXML
39     private ImageView EYE_SLASH;
40
41     @FXML
42     private ImageView EYE_SLASHCF;
43
44     @FXML
45     private PasswordField NewPassword;
46
47     @FXML
48     private PasswordField ConfirmNewPassword;
49
50     @FXML
51     private TextField ShowConfirmNewPassword;
52
53     @FXML
54     private TextField Shownewpassword;
55
56     @FXML
57     private Button btnEYE;
58
59     @FXML
60     private Button btnEYECF;
61
62     @FXML
63     private Button btnEYE_SLASH;
64
65     @FXML
66     private Button btnEYE_SLASHCF;
67
68     @FXML
69     private Button btnResetPassword;
70
71     private boolean isPasswordVisible = false;
72     private boolean isCFPasswordVisible = false;
73 }
```

3.1 Code

```
1  @Override
2      public void initialize(URL url, ResourceBundle rb) {
3          btnResetPassword.setOnAction(event -> resetPassword());
4          btnEYE.setOnAction(event -> togglePasswordVisibility());
5          btnEYE_SLASH.setOnAction(event -> togglePasswordVisibility());
6          EYE.setOnMouseClicked(event -> togglePasswordVisibility());
7          EYE_SLASH.setOnMouseClicked(event -> togglePasswordVisibility());
8          updatePasswordVisibility();
9          btnEYECF.setOnAction(event -> toggleCFPasswordVisibility());
10         btnEYE_SLASHCF.setOnAction(event -> toggleCFPasswordVisibility());
11         EYECF.setOnMouseClicked(event -> toggleCFPasswordVisibility());
12         EYE_SLASHCF.setOnMouseClicked(event -> toggleCFPasswordVisibility());
13         updateCFPasswordVisibility();
14
15         exit.setOnMouseClicked(event -> {
16             Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
17             stage.close();
18         });
19     }
20
21     @FXML
22     private void Sendcode(ActionEvent event) throws Exception {
23         if (event.getSource() == btnSendCode) {
24
25             forgotpane.setVisible(false);
26             verifypane.setVisible(true);
27             resetpane.setVisible(false);
28             sendOTP();
29
30         }
31     }
32
33     @FXML
34     private void VerifyCode(ActionEvent event) throws Exception {
35         if (event.getSource() == btnVerify) {
36
37             String otp = hide.getText();
38             String enteredOtp = txtCode.getText().trim();
39
40             if (otp.equals(enteredOtp)) {
41                 // OTP is correct, load the Staff Dashboard
42                 forgotpane.setVisible(false);
43                 verifypane.setVisible(false);
44                 resetpane.setVisible(true);
45             } else {
46                 JOptionPane.showMessageDialog(null, "Invalid OTP. Please try again.");
47             }
48         }
49     }
50
51     private void showAlert(String title, String message) {
52         Alert alert = new Alert(Alert.AlertType.ERROR);
53         alert.setTitle(title);
54         alert.setHeaderText(null);
55         alert.setContentText(message);
56         alert.showAndWait();
57     }
58 }
```

```
1 @FXML
2     private void resetPassword() {
3         String newPassword = NewPassword.getText().trim();
4         String confirmPassword = ConfirmNewPassword.getText().trim();
5
6         if (newPassword.isEmpty() || confirmPassword.isEmpty()) {
7             // Hiển thị thông báo lỗi: Cả hai trường phải được điền
8             showAlert("Error", "Both fields must be filled.");
9             return;
10        }
11
12        if (!newPassword.equals(confirmPassword)) {
13            // Hiển thị thông báo lỗi: Mật khẩu không khớp
14            showAlert("Error", "Passwords do not match.");
15            return;
16        }
17
18        try (Connection connection = DB.ConnectDB.getConnectDB()) {
19            // Bước 1: Lấy Phone từ bảng Information dựa trên email
20            String phoneQuery = "SELECT Phone FROM Information WHERE Email = ?";
21            try (PreparedStatement phoneStatement = connection.prepareStatement(phoneQuery)) {
22                phoneStatement.setString(1, txtemail.getText());
23                ResultSet resultSet = phoneStatement.executeQuery();
24                if (resultSet.next()) {
25                    String phone = resultSet.getString("Phone");
26
27                    // Đặt username bằng giá trị phone
28                    String username = phone;
29
30                    // Bước 3: Cập nhật mật khẩu mới trong bảng Login
31                    String updatePasswordQuery = "UPDATE Login SET Password = ? WHERE Username = ?";
32                    try (PreparedStatement updatePasswordStatement = connection.prepareStatement(updatePasswordQuery)) {
33                        updatePasswordStatement.setString(1, newPassword);
34                        updatePasswordStatement.setString(2, username);
35                        int rowsUpdated = updatePasswordStatement.executeUpdate();
36                        if (rowsUpdated > 0) {
37                            System.out.println("Password updated successfully for user: " + username);
38                            try {
39                                FXMLLoader loader = new FXMLLoader(getClass().getResource("/Login/Login.fxml"));
40                                Parent root = loader.load();
41
42                                Stage stage = new Stage();
43                                stage.initStyle(StageStyle.UNDECORATED);
44                                stage.setScene(new Scene(root));
45                                stage.show();
46                                clearRMPassword(username);
47
48                            } catch (IOException e) {
49                                JOptionPane.showMessageDialog(null, "Error loading verification screen: " + e.getMessage());
50                            }
51                        } else {
52                            System.out.println("Failed to update password for user: " + username);
53                        }
54                    }
55                } else {
56                    System.out.println("No user found with phone: ");
57                    // Xử lý khi không tìm thấy Username
58                }
59            }
60        } catch (SQLException e) {
61            // Xử lý ngoại lệ SQL
62            e.printStackTrace();
63        }
64    }
```

```
1  @FXML
2  private void clearRMPassword(String username) {
3      try (Connection conn = DB.ConnectDB.getConnectDB()) {
4          if (conn == null) {
5              System.out.println("Connection is null. Cannot clear password.");
6              return;
7          }
8
9          String query = "UPDATE Login SET RmbPassword = NULL WHERE Username = ?";
10         try (PreparedStatement stmt = conn.prepareStatement(query)) {
11             stmt.setString(1, username);
12             int rowsUpdated = stmt.executeUpdate();
13             if (rowsUpdated > 0) {
14                 System.out.println("Password cleared successfully.");
15             } else {
16                 System.out.println("No user found with the specified username.");
17             }
18         }
19         } catch (SQLException e) {
20             System.out.println("SQL Exception: " + e.getMessage());
21         }
22     }
23
24 // Toggle visibility of NewPassword field
25 @FXML
26 private void togglePasswordVisibility() {
27     isPasswordVisible = !isPasswordVisible;
28     updatePasswordVisibility();
29 }
30
31 private void updatePasswordVisibility() {
32     if (isPasswordVisible) {
33         NewPassword.setVisible(false);
34         Shownewpassword.setVisible(true);
35         Shownewpassword.setText(NewPassword.getText());
36         btnEYE.setVisible(false);
37         btnEYE_SLASH.setVisible(true);
38     } else {
39         NewPassword.setVisible(true);
40         Shownewpassword.setVisible(false);
41         btnEYE.setVisible(true);
42         btnEYE_SLASH.setVisible(false);
43     }
44 }
45
46 // Toggle visibility of ConfirmNewPassword field
47 @FXML
48 private void toggleCFPasswordVisibility() {
49     isCFPasswordVisible = !isCFPasswordVisible;
50     updateCFPasswordVisibility();
51 }
52
53 private void updateCFPasswordVisibility() {
54     if (isCFPasswordVisible) {
55         ConfirmNewPassword.setVisible(false);
56         ShowConfirmNewPassword.setVisible(true);
57         ShowConfirmNewPassword.setText(ConfirmNewPassword.getText());
58         btnEYECF.setVisible(false);
59         btnEYE_SLASHCF.setVisible(true);
60     } else {
61         ConfirmNewPassword.setVisible(true);
62         ShowConfirmNewPassword.setVisible(false);
63         btnEYECF.setVisible(true);
64         btnEYE_SLASHCF.setVisible(false);
65     }
66 }
```

```
1  private String generateOTP(int len) {
2      // All possible characters of my OTP
3      String str = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" + "0123456789";
4      int n = str.length();
5
6      // String to hold my OTP
7      StringBuilde otp = new StringBuilde();
8
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

```
1  public class FXMLMainController implements Initializable {
2
3      private LoginController loginController;
4      private BillsController billsController;
5
6      public void setBillsController(BillsController billsController) {
7          this.billsController = billsController;
8      }
9
10     public FXMLMainController() {
11
12     }
13     private CustomerController customerController;
14     private OrderTransactionController orderTransactionController;
15
16     private static FXMLMainController instance;
17     private boolean drawerImageOpen = false;
18     @FXML
19     private ImageView Bills;
20
21     @FXML
22     private ImageView Customer;
23
24     @FXML
25     private ImageView Home;
26
27     @FXML
28     private ImageView Logout;
29
30     @FXML
31     private ImageView OrderReceipt;
32
33     @FXML
34     private ImageView Product;
35
36     @FXML
37     private ImageView ProductDetails;
38     @FXML
39     private ImageView Setting;
40     @FXML
41     private ImageView Selling;
42
43     @FXML
44     private ImageView Staff;
45
46     @FXML
47     private ImageView Supplier;
48
49     @FXML
50     private Button BtnSetting;
51     @FXML
52     private Label Time;
53     private Timeline timeline;
54     @FXML
55     private Label expiryCountLabel;
56     @FXML
57     private Button btnLogOut;
58     @FXML
59     private Button Export;
60     @FXML
61     private Button User;
62     @FXML
63     private TextField textField;
64     @FXML
65     public AnchorPane opacityPane, drawerPane, main;
66     @FXML
67     private Button drawerImage;
68     @FXML
69     private Label UsernameTF;
70     @FXML
71     private ImageView exit;
72     @FXML
73     private TextArea expiringProductsDetails;
74     @FXML
75     private Label orderCountLabelInvoide;
76     @FXML
77     private Label orderCountLabel;
```

4.1 Code

```
● ● ●
1  public static FXMLMainController getInstance() {
2      return instance;
3  }
4
5  public void setFullName(String fullName) {
6      UsernameTF.setText(fullName);
7      UsernameTF.setFont(Font.font("System", 20.0));
8  }
9
10 public void loadFXML(String fxml) {
11     try {
12         FXMLLoader loader = new FXMLLoader(getClass().getResource(fxml));
13         Parent root = loader.load();
14
15         if (fxml.equals("/demoproject1/FXMLChamluong.fxml")) {
16             FXMLChamluongController controller = loader.getController();
17             controller.setMainController(this); // Thiết lập mainController cho FXMLChamluongController
18         } else if (fxml.equals("/demoproject1/FXMLNhanVien.fxml")) {
19             FXMLNhanVienController controller = loader.getController();
20             controller.setMainController(this); // Thiết lập mainController cho FXMLNhanvienController
21         } else if (fxml.equals("/demoproject1/FMLBangluong.fxml")) {
22             FMLBangluongController controller = loader.getController();
23             controller.setMainController(this); // Thiết lập mainController cho FMLBangluongController
24         }
25
26         main.getChildren().clear();
27         main.getChildren().add(root);
28
29     } catch (IOException e) {
30         e.printStackTrace();
31     }
32 }
33 }
34
35 @FXML
36 private void handleSupplier() {
37     try {
38         FXMLLoader loader = new FXMLLoader(getClass().getResource("/eproject/Supplier.fxml"));
39         Parent productPane = loader.load();
40
41         main.getChildren().clear();
42         main.getChildren().add(productPane);
43         opacityPane.setVisible(true);
44
45     } catch (IOException e) {
46         e.printStackTrace();
47     }
48     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
49     fadeTransition.setFromValue(0.15);
50     fadeTransition.setToValue(0);
51     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
52     fadeTransition.play();
53
54     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
55     translateTransition.setByX(-600);
56     translateTransition.play();
57
58     drawerImageOpen = false;
59 }
60 }
```

```
 1 @FXML
 2     private void handleOrderReceipt() {
 3         try {
 4             FXMLLoader loader = new FXMLLoader(getClass().getResource("/e-project/ImportOrderReceipt.fxml"));
 5             Parent productPane = loader.load();
 6
 7             main.getChildren().clear();
 8             main.getChildren().add(productPane);
 9             opacityPane.setVisible(true);
10
11         } catch (IOException e) {
12             e.printStackTrace();
13         }
14         FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
15         fadeTransition.setFromValue(0.15);
16         fadeTransition.setToValue(0);
17         fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
18         fadeTransition.play();
19
20         TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
21         translateTransition.setByX(-600);
22         translateTransition.play();
23
24         drawerImageOpen = false;
25     }
26
27 @FXML
28     private void handleProduct() {
29         try {
30             FXMLLoader loader = new FXMLLoader(getClass().getResource("/Druglist/Druglist.fxml"));
31             Parent productPane = loader.load();
32
33             main.getChildren().clear();
34             main.getChildren().add(productPane);
35             opacityPane.setVisible(true);
36
37         } catch (IOException e) {
38             e.printStackTrace();
39         }
40         FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
41         fadeTransition.setFromValue(0.15);
42         fadeTransition.setToValue(0);
43         fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
44         fadeTransition.play();
45
46         TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
47         translateTransition.setByX(-600);
48         translateTransition.play();
49
50         drawerImageOpen = false;
51     }
52
53 @FXML
54     private void handleSetting() {
55         try {
56             FXMLLoader loader = new FXMLLoader(getClass().getResource("/AccountInfo/AccountInfo.fxml"));
57             Parent productDetails = loader.load();
58
59             main.getChildren().clear();
60             main.getChildren().add(productDetails);
61             opacityPane.setVisible(true);
62
63         } catch (IOException e) {
64             e.printStackTrace();
65         }
66         FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
67         fadeTransition.setFromValue(0.15);
68         fadeTransition.setToValue(0);
69         fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
70         fadeTransition.play();
71
72         TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
73         translateTransition.setByX(-600);
74         translateTransition.play();
75
76         drawerImageOpen = false;
77     }
```

```
1  @FXML
2  private void handleProductDetails() {
3      try {
4          FXMLLoader loader = new FXMLLoader(getClass().getResource("/ImportMedicineDetails/ImportMedicineDetails.fxml"));
5          Parent productDetails = loader.load();
6
7          main.getChildren().clear();
8          main.getChildren().add(productDetails);
9          opacityPane.setVisible(true);
10
11     } catch (IOException e) {
12         e.printStackTrace();
13     }
14     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
15     fadeTransition.setFromValue(0.15);
16     fadeTransition.setToValue(0);
17     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
18     fadeTransition.play();
19
20     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
21     translateTransition.setByX(-600);
22     translateTransition.play();
23
24     drawerImageOpen = false;
25 }
26
27 @FXML
28 private void handleSellingButtonClick() {
29     try {
30         // Load the OrderTransaction.fxml file
31         FXMLLoader loader = new FXMLLoader(getClass().getResource("OrderTransaction.fxml"));
32         Parent orderTransactionPane = loader.load();
33
34         // Access the controller instance from the loader
35         OrderTransactionController orderTransactionController = loader.getController();
36
37         // Set the CustomerController instance in OrderTransactionController
38         orderTransactionController.setCustomerController(new CustomerController());
39
40         // Clear the main pane and add OrderTransaction.fxml to it
41         main.getChildren().clear();
42         main.getChildren().add(orderTransactionPane);
43
44         // Show the opacity pane
45         opacityPane.setVisible(true);
46
47     } catch (IOException e) {
48         e.printStackTrace();
49     }
50
51     // Animation to fade out opacity pane
52     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
53     fadeTransition.setFromValue(0.15);
54     fadeTransition.setToValue(0);
55     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
56     fadeTransition.play();
57
58     // Animation to slide drawer pane out of view
59     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
60     translateTransition.setByX(-600);
61     translateTransition.play();
62
63     drawerImageOpen = false;
64 }
65 }
```

```
1  @FXML
2  private void handleCustomerClick() {
3      try {
4          FXMLLoader loader = new FXMLLoader(getClass().getResource("Customer.fxml"));
5          Parent customerPane = loader.load();
6
7          main.getChildren().clear();
8          main.getChildren().add(customerPane);
9          opacityPane.setVisible(true);
10
11     } catch (IOException e) {
12         e.printStackTrace();
13     }
14     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
15     fadeTransition.setFromValue(0.15);
16     fadeTransition.setToValue(0);
17     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
18     fadeTransition.play();
19
20     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
21     translateTransition.setByX(-600);
22     translateTransition.play();
23
24     drawerImageOpen = false;
25 }
26
27 @FXML
28 private void handleBillsClick() {
29     try {
30         FXMLLoader loader = new FXMLLoader(getClass().getResource("Bills.fxml"));
31         Parent customerPane = loader.load();
32
33         main.getChildren().clear();
34         main.getChildren().add(customerPane);
35         opacityPane.setVisible(true);
36
37     } catch (IOException e) {
38         e.printStackTrace();
39     }
40     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
41     fadeTransition.setFromValue(0.15);
42     fadeTransition.setToValue(0);
43     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
44     fadeTransition.play();
45
46     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
47     translateTransition.setByX(-600);
48     translateTransition.play();
49
50     drawerImageOpen = false;
51 }
52
53 @FXML
54 private void handleDashboardButtonClick() {
55     try {
56
57         FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLMain.fxml"));
58         Parent mainPane = loader.load();
59
60         main.getChildren().clear();
61         main.getChildren().add(mainPane);
62
63         opacityPane.setVisible(true);
64
65     } catch (IOException e) {
66         e.printStackTrace();
67     }
68
69     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
70     fadeTransition.setFromValue(0.15);
71     fadeTransition.setToValue(0);
72     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
73     fadeTransition.play();
74
75     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
76     translateTransition.setByX(-600); // Move drawerPane to the left
77     translateTransition.play();
78
79     drawerImageOpen = false;
80 }
81 }
```

```

1  @FXML
2  private void handleUserButtonClick() {
3
4      loadFXML("/demoproject1/FXMLNhanVien.fxml");
5      FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
6      fadeTransition.setFromValue(0.15);
7      fadeTransition.setToValue(0);
8      fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
9      fadeTransition.play();
10
11     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
12     translateTransition.setByX(-600);
13     translateTransition.play();
14
15     drawerImageOpen = false;
16 }
17
18 public void updateLabels() {
19     // Call updateTotalBillsAndRevenue from BillsController
20     billsController.updateTotalBillsAndRevenue();
21     // Retrieve updated values
22     int totalBills = billsController.getTotalBills();
23     int totalRevenue = billsController.getTotalRevenue();
24
25     // Update labels in FXMLStaff
26     orderCountLabel.setText(String.format("%d Invoices", totalBills));
27     orderCountLabelInvoide.setText(String.format("%d", totalRevenue));
28 }
29
30 @FXML
31 private LineChart<String, Number> revenueChart;
32 private final CategoryAxis xAxis = new CategoryAxis();
33 private final NumberAxis yAxis = new NumberAxis();
34
35 public void updateRevenueChart() {
36     revenueChart.getData().clear(); // Xóa dữ liệu hiện tại trên biểu đồ
37
38     // Lấy dữ liệu tổng doanh thu theo từng ngày trong tháng hiện tại
39     XYChart.Series<String, Number> series = new XYChart.Series<>();
40     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
41     Calendar calendar = Calendar.getInstance();
42     calendar.set(Calendar.DAY_OF_MONTH, 4); // Đặt ngày là ngày đầu tiên của tháng
43
44     try {
45
46         Connection conn = DB.ConnectDB.getConnectDB();
47         // Lặp qua từng ngày trong tháng và lấy tổng doanh thu
48         while (calendar.get(Calendar.YEAR) == Calendar.getInstance().get(Calendar.YEAR)
49               && calendar.get(Calendar.MONTH) <= Calendar.getInstance().get(Calendar.MONTH)) {
50             String currentDate = sdf.format(calendar.getTime());
51             String sql = "SELECT SUM(TotalPrice) AS TotalRevenue FROM Bills WHERE BillDate = ?";
52
53             PreparedStatement pst = conn.prepareStatement(sql);
54             pst.setString(1, currentDate);
55             ResultSet rs = pst.executeQuery();
56
57             if (rs.next()) {
58                 int totalRevenue = rs.getInt("TotalRevenue");
59                 int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
60                 // Thêm dữ liệu vào series, chuyển totalRevenue thành Integer
61                 series.getData().add(new XYChart.Data<>(currentDate, totalRevenue));
62             }
63
64             pst.close();
65             calendar.add(Calendar.DAY_OF_MONTH, 1); // Chuyển sang ngày tiếp theo
66
67         }
68
69         conn.close();
70     } catch (SQLException e) {
71         e.printStackTrace();
72     }
73
74     // Thêm series vào biểu đồ
75     revenueChart.getData().add(series);
76 }

```

```
1  @FXML
2  private void checkAllProductsExpiryInfo() {
3      List<ProductExpiryTable> expiringProductsList = fetchExpiringProductsFromDatabase();
4
5      // Update UI on the JavaFX Application Thread
6      Platform.runLater(() -> {
7          expiryCountLabel.setText("There are " + expiringProductsList.size() + " items that will expire within 1 month.");
8      });
9  }
10
11 @FXML
12 private void showProductExpiryInfoForm() {
13     List<ProductExpiryTable> expiringProductsList = fetchExpiringProductsFromDatabase();
14
15     try {
16         FXMLLoader loader = new FXMLLoader(getClass().getResource("/sam/ProductExpiryInfo.fxml"));
17         Parent root = loader.load();
18         ProductExpiryInfoController productExpiryInfoController = loader.getController();
19         productExpiryInfoController.setExpiringProducts(expiringProductsList);
20
21         Stage stage = new Stage();
22         stage.setScene(new Scene(root));
23         stage.initStyle(StageStyle.UNDECORATED);
24         stage.show();
25
26     } catch (IOException ex) {
27         System.out.println("Error loading ProductExpiryInfo.fxml: " + ex.getMessage());
28     }
29 }
30
31 private List<ProductExpiryTable> fetchExpiringProductsFromDatabase() {
32     List<ProductExpiryTable> expiringProductsList = new ArrayList<>();
33     String sql = "SELECT p.MedicineID, p.MedicineName, pd.ManufacturingDate, pd.ExpireDate "
34     + "FROM tblProductDetails pd "
35     + "JOIN tblProduct p ON pd.MedicineID = p.MedicineID";
36
37     try (Connection conn = DB.ConnectDB.getConnectDB(); PreparedStatement pst = conn.prepareStatement(sql); ResultSet rs = pst.executeQuery()) {
38
39         while (rs.next()) {
40             String medicineID = rs.getString("MedicineID");
41             String medicineName = rs.getString("MedicineName");
42             LocalDate manufacturingDate = rs.getDate("ManufacturingDate").toLocalDate();
43             LocalDate expireDate = rs.getDate("ExpireDate").toLocalDate();
44
45             long daysLeft = ChronoUnit.DAYS.between(LocalDate.now(), expireDate);
46
47             if (daysLeft >= 0 && daysLeft <= 30) {
48                 // Tính số ngày còn lại đến ngày hết hạn
49                 long daysUntilExpired = ChronoUnit.DAYS.between(LocalDate.now(), expireDate);
50
51                 // Tạo đối tượng Product và thêm vào danh sách
52                 ProductExpiryTable product = new ProductExpiryTable(medicineID, medicineName, manufacturingDate, expireDate, daysUntilExpired);
53                 expiringProductsList.add(product);
54             }
55         }
56
57     } catch (Exception e) {
58         System.out.println("Error fetching product details: " + e.getMessage());
59     }
60
61     return expiringProductsList;
62 }
63
64 @FXML
65 private void handleLogout() {
66     try {
67         // Load login.fxml
68         Parent root = FXMLLoader.load(getClass().getResource("/Login/Login.fxml"));
69         Scene scene = new Scene(root);
70         String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
71         scene.getStylesheets().add(cssPath);
72
73         Stage stage = (Stage) btnLogOut.getScene().getWindow(); // Get the current stage
74         stage.setScene(scene);
75         stage.centerOnScreen();
76         stage.show();
77     } catch (IOException e) {
78         e.printStackTrace(); // Handle exception properly in your application
79     }
80 }
```

```

1  @Override
2  public void initialize(URL location, ResourceBundle resources) {
3      //load
4      //timeline
5      timeline = new Timeline(
6          new KeyFrame(Duration.millis(500), event -> {
7              updateTime(); // Cập nhật thời gian mỗi giây
8          })
9      );
10     timeline.setCycleCount(Animation.INDEFINITE); // Lặp vô hạn
11     timeline.play();
12     //labelBill
13     billsController = new BillsController();
14     updateBill();
15     //linechart
16     revenueChart.getData().clear();
17     updateRevenueChart();
18     //productExpiry
19     checkAllProductsExpiryInfo();
20     exit.setOnMouseClicked(event -> {
21         System.exit(0);
22     });
23
24     opacityPane.setVisible(false);
25
26     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
27     fadeTransition.setFromValue(1);
28     fadeTransition.setToValue(0);
29     fadeTransition.play();
30
31     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
32     translateTransition.setByX(-600);
33     translateTransition.play();
34
35     drawerImage.setOnMouseClicked(event -> {
36         if (!drawerImageOpen) {
37             opacityPane.setVisible(true);
38             FadeTransition fadeTransition1 = new FadeTransition(Duration.seconds(0.5), opacityPane);
39             fadeTransition1.setFromValue(0);
40             fadeTransition1.setToValue(0.15);
41             fadeTransition1.play();
42
43             TranslateTransition translateTransition1 = new TranslateTransition(Duration.seconds(0.1), drawerPane);
44             translateTransition1.setByX(600);
45             translateTransition1.play();
46
47             drawerImageOpen = true;
48         } else {
49             FadeTransition fadeTransition1 = new FadeTransition(Duration.seconds(0.5), opacityPane);
50             fadeTransition1.setFromValue(0.15);
51             fadeTransition1.setToValue(0);
52             fadeTransition1.setOnFinished(event1 -> opacityPane.setVisible(false));
53             fadeTransition1.play();
54
55             TranslateTransition translateTransition1 = new TranslateTransition(Duration.seconds(0.1), drawerPane);
56             translateTransition1.setByX(-600); // Move drawerPane to the left
57             translateTransition1.play();
58
59             drawerImageOpen = false;
60         }
61     });
62     Home.setOnMouseClicked(event -> {
63         loadFXML("FXMLMain.fxml");
64     });
65     );
66     Staff.setOnMouseClicked(event -> {
67         loadFXML("./demoproject1/FXMLNhanVien.fxml");
68     });
69     Customer.setOnMouseClicked(event -> {
70         loadFXML("Customer.fxml");
71     });
72     Product.setOnMouseClicked(event -> {
73         loadFXML("./Druglist/Druglist.fxml");
74     });
75     Bills.setOnMouseClicked(event -> {
76         loadFXML("Bills.fxml");
77     });
78     ProductDetails.setOnMouseClicked(event -> {
79         loadFXML("./ImportMedicineDetails/ImportMedicineDetails.fxml");
80     });
81     OrderReceipt.setOnMouseClicked(event -> {
82         loadFXML("./e-project/ImportOrderReceipt.fxml");
83     });
84     Setting.setOnMouseClicked(event -> {
85         loadFXML("./AccountInfo/AccountInfo.fxml");
86     });
87     Supplier.setOnMouseClicked(event -> {
88         loadFXML("./e-project/Supplier.fxml");
89     });
90     Selling.setOnMouseClicked(event -> {
91         loadFXML("OrderTransaction.fxml");
92     });
93     Logout.setOnMouseClicked(event -> {
94         try {
95             // Load login.fxml
96             Parent root = FXMLLoader.load(getClass().getResource("/Login/Login.fxml"));
97             Scene scene = new Scene(root);
98             String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
99             scene.getStylesheets().add(cssPath);
100
101             Stage stage = (Stage) Logout.getScene().getWindow(); // Get the current stage
102             stage.setScene(scene);
103             stage.centerOnScreen();
104             stage.show();
105         } catch (IOException e) {
106             e.printStackTrace(); // Handle exception properly in your application
107         }
108     });
109
110
111     private void updateTime() {
112         LocalDateTime now = localDateTime.now();
113         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm d MMMM yyyy");
114         String formattedDateTime = now.format(formatter);
115         Time.setText(formattedDateTime); // Đặt giá trị vào Label có fx:id là Time
116     }
117 }
118

```

5 Dashboard Staff

```
1  public class FXMLStaffController implements Initializable {
2
3      public void setBillsController(BillsController billsController) {
4          this.billsController = billsController;
5      }
6      private CustomerController customerController;
7      private OrderTransactionController orderTransactionController;
8      private boolean drawerImageOpen = false;
9      @FXML
10     private ImageView Bill;
11
12     @FXML
13     private ImageView Customer;
14
15     @FXML
16     private ImageView Home;
17
18     @FXML
19     private ImageView Logout;
20
21     @FXML
22     private ImageView Order;
23     @FXML
24     private Label Time;
25     private Timeline timeline;
26     @FXML
27     private Label expiryCountLabel;
28     @FXML
29     private Button btnLogOut;
30     @FXML
31     private Button admin;
32     @FXML
33     private Label totalBillsLabel;
34     @FXML
35     private Label UserNameLB;
36     @FXML
37     private Label totalRevenueLabel;
38     @FXML
39     private Button drawerImage;
40     @FXML
41     private Button Selling;
42     @FXML
43     private AnchorPane drawerPane;
44     @FXML
45     private AnchorPane main;
46     private LoginController loginController;
47     private BillsController billsController;
48     @FXML
49     private AnchorPane opacityPane;
50
51     @FXML
52     private ImageView exit;
53
54     public void setFullName(String fullName) {
55         UserNameLB.setText(fullName);
56         UserNameLB.setFont(Font.font("System", 20.0));
57     }
```

5.1 Code

```
1 public void handleDashboardButtonClick() {
2     try {
3
4         FXMLLoader loader = new FXMLLoader(getClass().getResource("/sam/FXMLStaff.fxml"));
5         Parent mainPane = loader.load();
6
7         main.getChildren().clear();
8         main.getChildren().add(mainPane);
9
10        opacityPane.setVisible(true);
11    } catch (IOException e) {
12        e.printStackTrace();
13    }
14
15    }
16
17    FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
18    fadeTransition.setFromValue(0.15);
19    fadeTransition.setToValue(0);
20    fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
21    fadeTransition.play();
22
23    TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
24    translateTransition.setByX(-600); // Move drawerPane to the left
25    translateTransition.play();
26
27    drawerImageOpen = false;
28}
29
30 @FXML
31 private void handleCustomer() {
32     try {
33         FXMLLoader loader = new FXMLLoader(getClass().getResource("Customer.fxml"));
34         Parent customerPane = loader.load();
35
36         main.getChildren().clear();
37         main.getChildren().add(customerPane);
38         opacityPane.setVisible(true);
39
40     } catch (IOException e) {
41         e.printStackTrace();
42     }
43     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
44     fadeTransition.setFromValue(0.15);
45     fadeTransition.setToValue(0);
46     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
47     fadeTransition.play();
48
49     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
50     translateTransition.setByX(-600);
51     translateTransition.play();
52
53     drawerImageOpen = false;
54 }
55
56 @FXML
57 private void handleBills() {
58     try {
59         FXMLLoader loader = new FXMLLoader(getClass().getResource("Bills.fxml"));
60         Parent customerPane = loader.load();
61
62         main.getChildren().clear();
63         main.getChildren().add(customerPane);
64         opacityPane.setVisible(true);
65
66     } catch (IOException e) {
67         e.printStackTrace();
68     }
69     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
70     fadeTransition.setFromValue(0.15);
71     fadeTransition.setToValue(0);
72     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
73     fadeTransition.play();
74
75     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
76     translateTransition.setByX(-600);
77     translateTransition.play();
78
79     drawerImageOpen = false;
80 }
81 }
```

```

1  public void updateLabels() {
2      // Call updateTotalBillsAndRevenue from BillsController
3      billsController.updateTotalBillsAndRevenue();
4
5      // Retrieve updated values
6      int totalBills = billsController.getTotalBills();
7      int totalRevenue = billsController.getTotalRevenue();
8
9      // Update labels in FXMLStaff
10     totalBillsLabel.setText(String.format("%d Invoices", totalBills));
11     totalRevenueLabel.setText(String.format("%dd", totalRevenue));
12 }
13
14 @FXML
15 private void handleSellng() {
16     try {
17         // Load the OrderTransaction.fxml file
18         FXMLLoader loader = new FXMLLoader(getClass().getResource("OrderTransaction.fxml"));
19         Parent orderTransactionPane = loader.load();
20
21         // Access the controller instance from the loader
22         OrderTransactionController orderTransactionController = loader.getController();
23
24         // Set the CustomerController instance in OrderTransactionController
25         orderTransactionController.setCustomerController(new CustomerController());
26
27         // Clear the main pane and add OrderTransaction.fxml to it
28         main.getChildren().clear();
29         main.getChildren().add(orderTransactionPane);
30
31         // Show the opacity pane
32         opacityPane.setVisible(true);
33
34     } catch (IOException e) {
35         e.printStackTrace();
36     }
37
38     // Animation to fade out opacity pane
39     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
40     fadeTransition.setFromValue(0.15);
41     fadeTransition.setToValue(0);
42     fadeTransition.setOnFinished(event -> opacityPane.setVisible(false));
43     fadeTransition.play();
44
45     // Animation to slide drawer pane out of view
46     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
47     translateTransition.setByX(-600);
48     translateTransition.play();
49
50     drawerImageOpen = false;
51 }
52
53 @FXML
54 private LineChart<String, Number> revenueChart;
55 private final CategoryAxis xAxis = new CategoryAxis();
56 private final NumberAxis yAxis = new NumberAxis();
57
58 public void updateRevenueChart() {
59     revenueChart.getData().clear(); // Xóa dữ liệu hiện tại trên biểu đồ
60
61     // Lấy dữ liệu tổng doanh thu theo từng ngày trong tháng hiện tại
62     XYChart.Series<String, Number> series = new XYChart.Series<>();
63     SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
64     Calendar calendar = Calendar.getInstance();
65     calendar.set(Calendar.DAY_OF_MONTH, 1); // Đặt ngày là ngày đầu tiên của tháng
66
67     try {
68
69         Connection conn = DB.ConnectDB.getConnectDB();
70         // Lặp qua từng ngày trong tháng và lấy tổng doanh thu
71         while (calendar.get(Calendar.YEAR) == Calendar.getInstance().get(Calendar.YEAR)
72             && calendar.get(Calendar.MONTH) <= Calendar.getInstance().get(Calendar.MONTH)) {
73             String currentDate = sdf.format(calendar.getTime());
74             String sql = "SELECT SUM(TotalPrice) AS TotalRevenue FROM Bills WHERE BillDate = ?";
75
76             PreparedStatement pst = conn.prepareStatement(sql);
77             pst.setString(1, currentDate);
78             ResultSet rs = pst.executeQuery();
79
80             if (rs.next()) {
81                 int totalRevenue = rs.getInt("TotalRevenue");
82                 int dayOfMonth = calendar.get(Calendar.DAY_OF_MONTH);
83                 // Thêm dữ liệu vào series, chuyển totalRevenue thành Integer
84                 series.getData().add(new XYChart.Data<>(currentDate, totalRevenue));
85             }
86
87             pst.close();
88             calendar.add(Calendar.DAY_OF_MONTH, 1); // Chuyển sang ngày tiếp theo
89         }
90
91         conn.close();
92     } catch (SQLException e) {
93         e.printStackTrace();
94     }
95
96     // Thêm series vào biểu đồ
97     revenueChart.getData().add(series);
98 }
99

```



```

1  @FXML
2  private void handleLogout() {
3      try {
4          // Load login.fxml
5          Parent root = FXMLLoader.load(getClass().getResource("/Login/Login.fxml"));
6          Scene scene = new Scene(root);
7          String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
8          scene.getStylesheets().add(cssPath);
9
10         Stage stage = (Stage) btnLogOut.getScene().getWindow(); // Get the current stage
11         stage.setScene(scene);
12         stage.centerOnScreen();
13         stage.show();
14     } catch (IOException e) {
15         e.printStackTrace(); // Handle exception properly in your application
16     }
17 }
18
19 @Override
20 public void initialize(URL url, ResourceBundle rb) {
21     //Timeline
22     timeline = new Timeline(
23         new KeyFrame(Duration.millis(500), event -> {
24             updateTime(); // Cập nhật thời gian mỗi giây
25         })
26     );
27     timeline.setCycleCount(Animation.INDEFINITE); // Lặp vô hạn
28     timeline.play();
29     //Label Bills
30     billsController = new BillsController();
31     updateLabels();
32     //productExpiry
33     checkAllProductsExpiryInfo();
34     //linechar
35     revenueChart.getData().clear();
36     updateRevenueChart();
37
38     exit.setOnMouseClicked(event -> {
39         System.exit(0);
40     });
41     opacityPane.setVisible(false);
42
43     // Initial fade out and translation of the drawer
44     FadeTransition fadeTransition = new FadeTransition(Duration.seconds(0.5), opacityPane);
45     fadeTransition.setFromValue(1);
46     fadeTransition.setToValue(0);
47     fadeTransition.play();
48
49     TranslateTransition translateTransition = new TranslateTransition(Duration.seconds(0.1), drawerPane);
50     translateTransition.setByYX(-600);
51     translateTransition.play();
52
53     drawerImage.setOnMouseClicked(event -> {
54         if (!drawerImageOpen) {
55             opacityPane.setVisible(true);
56             FadeTransition fadeTransition1 = new FadeTransition(Duration.seconds(0.5), opacityPane);
57             fadeTransition1.setFromValue(0);
58             fadeTransition1.setToValue(0.15);
59             fadeTransition1.play();
60
61             TranslateTransition translateTransition1 = new TranslateTransition(Duration.seconds(0.1), drawerPane);
62             translateTransition1.setByYX(600); // Move drawerPane to the right
63             translateTransition1.play();
64
65             drawerImageOpen = true; // Set state to open
66         } else { // Close drawer if it's open
67             FadeTransition fadeTransition1 = new FadeTransition(Duration.seconds(0.5), opacityPane);
68             fadeTransition1.setFromValue(0.15);
69             fadeTransition1.setToValue(0);
70             fadeTransition1.setOnFinished(event1 -> opacityPane.setVisible(false));
71             fadeTransition1.play();
72
73             TranslateTransition translateTransition1 = new TranslateTransition(Duration.seconds(0.1), drawerPane);
74             translateTransition1.setByYX(-600); // Move drawerPane to the left
75             translateTransition1.play();
76
77             drawerImageOpen = false; // Set state to close
78         }
79     });
80     Home.setOnMouseClicked(event -> {
81         loadFXML("/sam/FXMLMain.fxml");
82     });
83     Customer.setOnMouseClicked(event -> {
84         loadFXML("/sam/Customer.fxml");
85     });
86     Order.setOnMouseClicked(event -> {
87         loadFXML("/sam/OrderTransaction.fxml");
88     });
89     Bill.setOnMouseClicked(event -> {
90         loadFXML("/sam/Bills.fxml");
91     });
92     Logout.setOnMouseClicked(event -> {
93         try {
94             // Load login.fxml
95             Parent root = FXMLLoader.load(getClass().getResource("/Login/Login.fxml"));
96             Scene scene = new Scene(root);
97             String cssPath = getClass().getResource("/sam/button-style.css").toExternalForm();
98             scene.getStylesheets().add(cssPath);
99
100            Stage stage = (Stage) Logout.getScene().getWindow(); // Get the current stage
101            stage.setScene(scene);
102            stage.centerOnScreen();
103            stage.show();
104        } catch (IOException e) {
105            e.printStackTrace(); // Handle exception properly in your application
106        }
107    });
108 }

```

```

1  @FXML
2  private void checkAllProductsExpiryInfo() {
3      List<ProductExpiryTable> expiringProductsList = fetchExpiringProductsFromDatabase();
4
5      Platform.runLater(() -> {
6          expiryCountLabel.setText("There are " + expiringProductsList.size() + " items that will expire within 1 month.");
7      });
8  }
9
10 @FXML
11 private void showProductExpiryInfoForm() {
12     List<ProductExpiryTable> expiringProductsList = fetchExpiringProductsFromDatabase();
13
14     try {
15         FXMLLoader loader = new FXMLLoader(getClass().getResource("/sam/ProductExpiryInfo.fxml"));
16         Parent root = loader.load();
17         ProductExpiryInfoController productExpiryInfoController = loader.getController();
18         productExpiryInfoController.setExpiringProducts(expiringProductsList);
19
20         Stage stage = new Stage();
21         stage.setScene(new Scene(root));
22         stage.initStyle(StageStyle.UNDECORATED);
23         stage.show();
24
25     } catch (IOException ex) {
26         System.out.println("Error loading ProductExpiryInfo.fxml: " + ex.getMessage());
27     }
28 }
29
30 private List<ProductExpiryTable> fetchExpiringProductsFromDatabase() {
31     List<ProductExpiryTable> expiringProductsList = new ArrayList<>();
32     String sql = "SELECT p.MedicineID, p.MedicineName, pd.ManufacturingDate, pd.ExpireDate "
33         + "FROM tblProductDetails pd "
34         + "JOIN tblProduct p ON pd.MedicineID = p.MedicineID";
35
36     try (Connection conn = DB.ConnectDB.getConnectDB(); PreparedStatement pst = conn.prepareStatement(sql); ResultSet rs = pst.executeQuery()) {
37
38         while (rs.next()) {
39             String medicineID = rs.getString("MedicineID");
40             String medicineName = rs.getString("MedicineName");
41             LocalDate manufacturingDate = rs.getDate("ManufacturingDate").toLocalDate();
42             LocalDate expireDate = rs.getDate("ExpireDate").toLocalDate();
43
44             long daysLeft = ChronoUnit.DAYS.between(LocalDate.now(), expireDate);
45
46             if (daysLeft >= 0 && daysLeft <= 30) {
47                 // Tính số ngày còn lại đến ngày hết hạn
48                 long daysUntilExpired = ChronoUnit.DAYS.between(LocalDate.now(), expireDate);
49
50                 // Tạo đối tượng Product và thêm vào danh sách
51                 ProductExpiryTable product = new ProductExpiryTable(medicineID, medicineName, manufacturingDate, expireDate, daysUntilExpired);
52                 expiringProductsList.add(product);
53             }
54         }
55
56     } catch (Exception e) {
57         System.out.println("Error fetching product details: " + e.getMessage());
58     }
59
60     return expiringProductsList;
61 }
62
63 private void updateTime() {
64     LocalDateTime now = LocalDateTime.now();
65     DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm d MMMM yyyy");
66     String formattedDateTime = now.format(formatter);
67     Time.setText(formattedDateTime); // Đặt giá trị vào Label có fx:id là Time
68 }
69
70 public void loadFXML(String fxml) {
71     try {
72         FXMLLoader loader = new FXMLLoader(getClass().getResource(fxml));
73         Parent root = loader.load();
74
75         main.getChildren().clear();
76         main.getChildren().add(root);
77
78     } catch (IOException e) {
79         e.printStackTrace();
80     }
81
82 }
83
84 }
85

```

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

```
1 @FXML
2     private void addCustomer() {
3         AlertMessage alert = new AlertMessage();
4         if (txtname.getText().isEmpty() || txtaddress.getText().isEmpty() || txtphone.getText().isEmpty() || txtemail.getText().isEmpty()
5             || txtpoint.getText().isEmpty()) {
6             alert.errorMessage("Please enter complete information");
7             return;
8         }
9
10        // Kiểm tra xem 'Nam' hoặc 'Nữ' đã được chọn chưa
11        if (!(rdmale.isSelected() ^ rdfemale.isSelected())) {
12            alert.errorMessage("Please select 'Male' or 'Female'");
13            return;
14        }
15
16        // Kiểm tra định dạng email có hợp lệ không
17        if (!txtemail.getText().matches("[a-zA-Z0-9_+&*-]+(?:\\.[a-zA-Z0-9_+&*-]+)*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$")) {
18            alert.errorMessage("Please enter the correct Email");
19            return;
20        }
21
22        // Kiểm tra xem số điện thoại đã tồn tại chưa
23        String checkPhoneQuery = "SELECT * FROM Customer WHERE Phone = ?";
24        try {
25
26            Connection conn = DB.ConnectDB.getConnectDB();
27
28            PreparedStatement checkPhoneStmt = conn.prepareStatement(checkPhoneQuery);
29            checkPhoneStmt.setString(1, txtphone.getText());
30            ResultSet existingPhone = checkPhoneStmt.executeQuery();
31            if (existingPhone.next()) {
32                alert.errorMessage("Phone number already exists");
33                return; // Thoát phương thức nếu số điện thoại đã tồn tại
34            }
35
36            // Kiểm tra xem email đã tồn tại chưa
37            String checkEmailQuery = "SELECT * FROM Customer WHERE Email = ?";
38            PreparedStatement checkEmailStmt = conn.prepareStatement(checkEmailQuery);
39            checkEmailStmt.setString(1, txtemail.getText());
40            ResultSet existingEmail = checkEmailStmt.executeQuery();
41            if (existingEmail.next()) {
42                alert.errorMessage("Email address already exists");
43                return; // Thoát phương thức nếu địa chỉ email đã tồn tại
44            }
45
46            // Nếu các kiểm tra đều đúng, tiến hành thêm vào cơ sở dữ liệu
47            String sql = "INSERT INTO Customer(CustomerName, Address, Gender, Phone, Email, Point) VALUES (?, ?, ?, ?, ?, ?)";
48            PreparedStatement pst = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
49            pst.setString(1, txtname.getText());
50            pst.setString(2, txtaddress.getText());
51            String gender = rdmale.isSelected() ? "Male" : "Female";
52            pst.setString(3, gender);
53            pst.setString(4, txtphone.getText());
54            pst.setString(5, txtemail.getText());
55            pst.setString(6, txtpoint.getText());
56            pst.executeUpdate();
57        }
```

```
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

```
1  public class BillsController implements Initializable {
2
3      @FXML
4      private Button Export;
5
6      @FXML
7      private TextField OrderSearchItem;
8
9      @FXML
10     private Text TotalPrice;
11
12     @FXML
13     private TableColumn<TableBills, String> columnBillID;
14
15     @FXML
16     private TableColumn<TableBills, String> columnPhone;
17
18     @FXML
19     private TableColumn<TableBills, String> columnCustomerName;
20
21     @FXML
22     private TableColumn<TableBills, Integer> columnTotalPrice;
23
24     @FXML
25     private TableColumn<TableBills, String> columnPaymentMethod;
26
27     @FXML
28     private TableColumn<TableBills, String> columnBillDate;
29
30     @FXML
31     private VBox vboxBillItems;
32
33     @FXML
34     void Table(MouseEvent event) {
35
36         @FXML
37         void osearch_item() {
38             String searchText = normalizeSearchText(OrderSearchItem.getText());
39
40             if (searchText.isEmpty()) {
41                 // If search text is empty, show all bills
42                 tableBills.setItems(bills);
43                 return;
44             }
45             String[] keywords = searchText.split("\\s+");
46
47             ObservableList<TableBills> filteredList = FXCollections.observableArrayList();
48
49             for (TableBills bill : bills) {
50                 String customerPhone = normalizeSearchText(bill.getCustomerPhone());
51                 String billID = normalizeSearchText(bill.getBillID());
52                 boolean matchesAnyKeyword = false;
53
54                 // Check if any keyword is found in customer phone or BillID
55                 for (String keyword : keywords) {
56                     if (customerPhone.contains(keyword) || billID.contains(keyword)) {
57                         matchesAnyKeyword = true;
58                         break;
59                     }
60                 }
61
62                 if (matchesAnyKeyword) {
63                     filteredList.add(bill);
64                 }
65             }
66             // Reset TableView with the filtered list
67             tableBills.setItems(filteredList);
68         }
69     }
70 }
```

7.1 Code

```

1  private String normalizeSearchText(String text) {
2      if (text == null) {
3          return "";
4      }
5      // Normalize text: remove accents and convert to lowercase
6      String normalizedText = Normalizer.normalize(text, Normalizer.Form.NFD)
7          .replaceAll("\\p{InCombiningDiacriticalMarks}+", "");
8      return normalizedText.toLowerCase().trim();
9  }
10
11 private ObservableList<TableBills> bills = FXCollections.observableArrayList();
12 private ObservableList<TableBillsItem> billItems = FXCollections.observableArrayList();
13
14 @Override
15 public void initialize(URL url, ResourceBundle resourceBundle) {
16     initializeBillTable();
17 }
18
19 private void initializeBillTable() {
20     columnBillID.setCellValueFactory(new PropertyValueFactory<>("billID"));
21     columnCustomerName.setCellValueFactory(new PropertyValueFactory<>("customerName"));
22     columnPhone.setCellValueFactory(new PropertyValueFactory<>("customerPhone"));
23     columnTotalPrice.setCellValueFactory(new PropertyValueFactory<>("totalPrice"));
24     columnPaymentMethod.setCellValueFactory(new PropertyValueFactory<>("paymentMethod"));
25     columnBillDate.setCellValueFactory(new PropertyValueFactory<>("billDate"));
26
27     loadBillsFromDatabase();
28     tableBills.setItems(bills);
29
30     // Double-click to show bill items
31     tableBills.setOnMouseClicked(event -> {
32         if (event.getClickCount() == 2) {
33             TableBills selectedBill = tableBills.getSelectionModel().getSelectedItem();
34             if (selectedBill != null) {
35                 loadBillItemsFromDatabase(selectedBill.getBillID());
36                 displayTotalPayment(selectedBill.getTotalPrice());
37             }
38         }
39     });
40 }
41
42 private void displayTotalPayment(double totalPayment) {
43     TotalPrice.setText(String.format("%.1f ₫", totalPayment));
44     TotalPrice.setStyle("-fx-font-size: 20px;");
45 }
46
47 private void loadBillsFromDatabase() {
48     bills.clear();
49     String sql = "SELECT * FROM Bills";
50     try {
51
52         Connection conn = DB.ConnectDB.getConnectDB();
53         PreparedStatement pst = conn.prepareStatement(sql);
54         ResultSet rs = pst.executeQuery();
55
56         while (rs.next()) {
57             java.sql.Date billDateSql = rs.getDate("BillDate");
58             TableBills bill = new TableBills(
59                 rs.getString("BillID"),
60                 rs.getString("CustomerId"),
61                 rs.getString("CustomerName"),
62                 rs.getString("PhoneCustomer"),
63                 rs.getInt("TotalPrice"),
64                 rs.getString("PaymentMethod"),
65                 billDateSql // Giữ nguyên là java.sql.Date
66             );
67             bills.add(bill);
68         }
69
70         pst.close();
71         conn.close();
72     } catch (SQLException e) {
73         e.printStackTrace();
74     }
75 }
76

```

```

1  private void loadBillItemsFromDatabase(String billID) {
2      vboxBillItems.getChildren().clear(); // Clear previous items
3      String sql = "SELECT * FROM BillItems WHERE BillID = ?";
4      try {
5
6          Connection conn = DB.ConnectDB.getConnectDB();
7          PreparedStatement pst = conn.prepareStatement(sql);
8          pst.setString(1, billID);
9          ResultSet rs = pst.executeQuery();
10
11         while (rs.next()) {
12             String itemName = rs.getString("ItemName");
13             int quantity = rs.getInt("Quantity");
14             int unitPrice = rs.getInt("UnitPrice");
15             String itemType = rs.getString("ItemType");
16             String medicineID = rs.getString("MedicineID");
17
18             TableBillsItem billItem = new TableBillsItem(itemName, quantity, unitPrice, itemType, medicineID);
19             AnchorPane pane = createBillItemPane(billItem);
20             vboxBillItems.getChildren().add(pane);
21         }
22
23         pst.close();
24         conn.close();
25     } catch (SQLException e) {
26         e.printStackTrace();
27     }
28 }
29
30 private AnchorPane createBillItemPane(TableBillsItem billItem) {
31     AnchorPane anchorPane = new AnchorPane();
32     anchorPane.setPrefHeight(75.0);
33     anchorPane.setPrefWidth(285.0);
34     anchorPane.setStyle("-fx-border-color: blue; -fx-border-radius: 10px; -fx-border-width: 1px;");
35     anchorPane.setMinWidth(285.0);
36     anchorPane.setMaxWidth(285.0);
37     anchorPane.setMinHeight(75.0);
38     anchorPane.setMaxHeight(75.0);
39
40     // Tạo các thành phần trong AnchorPane
41     Text itemNameText = new Text(billItem.getItemName());
42     itemNameText.setId("itemNameText");
43     itemNameText.setLayoutX(14.0);
44     itemNameText.setLayoutY(18.0);
45     itemNameText.setFont(Font.font("Arial Black", 12.0));
46     anchorPane.getChildren().add(itemNameText);
47
48     Text medicineIdText = new Text(billItem.getMedicineID());
49     medicineIdText.setId("medicineID");
50     medicineIdText.setLayoutX(200.0);
51     medicineIdText.setLayoutY(18.0);
52     medicineIdText.setFont(Font.font("Arial Black", 12.0));
53     anchorPane.getChildren().add(medicineIdText);
54
55     Text quantityText = new Text("Quantity");
56     quantityText.setLayoutX(14.0);
57     quantityText.setLayoutY(41.0);
58     anchorPane.getChildren().add(quantityText);
59
60     TextField quantityField = new TextField(String.valueOf(billItem.getQuantity()));
61     quantityField.setId("quantityField");
62     quantityField.setLayoutX(64.0);
63     quantityField.setLayoutY(20.0);
64     quantityField.setPrefHeight(18.0);
65     quantityField.setPrefWidth(45.0);
66     quantityField.setStyle("-fx-text-fill: black; -fx-border-color: black; -fx-border-width: 1px; -fx-border-radius: 2px; -fx-alignment: center;");
67     anchorPane.getChildren().add(quantityField);
68
69     Text unitText = new Text(billItem.getItemType());
70     unitText.setLayoutX(115.0);
71     unitText.setLayoutY(41.0);
72     anchorPane.getChildren().add(unitText);
73     Text priceText = new Text("Price:");
74     priceText.setLayoutX(170.0);
75     priceText.setLayoutY(67.0);
76     priceText.setFont(Font.font("Arial Bold", 14.0));
77     anchorPane.getChildren().add(priceText);
78
79     Text itemPriceText = new Text(billItem.getUnitPrice() + "đ");
80     itemPriceText.setId("unitPriceText");
81     itemPriceText.setLayoutX(220.0);
82     itemPriceText.setLayoutY(67.0);
83     itemPriceText.setFont(Font.font("Arial Bold", 14.0));
84     anchorPane.getChildren().add(itemPriceText);
85
86     VBox.setMargin(anchorPane, new Insets(2.0));
87
88     return anchorPane;
89 }
90

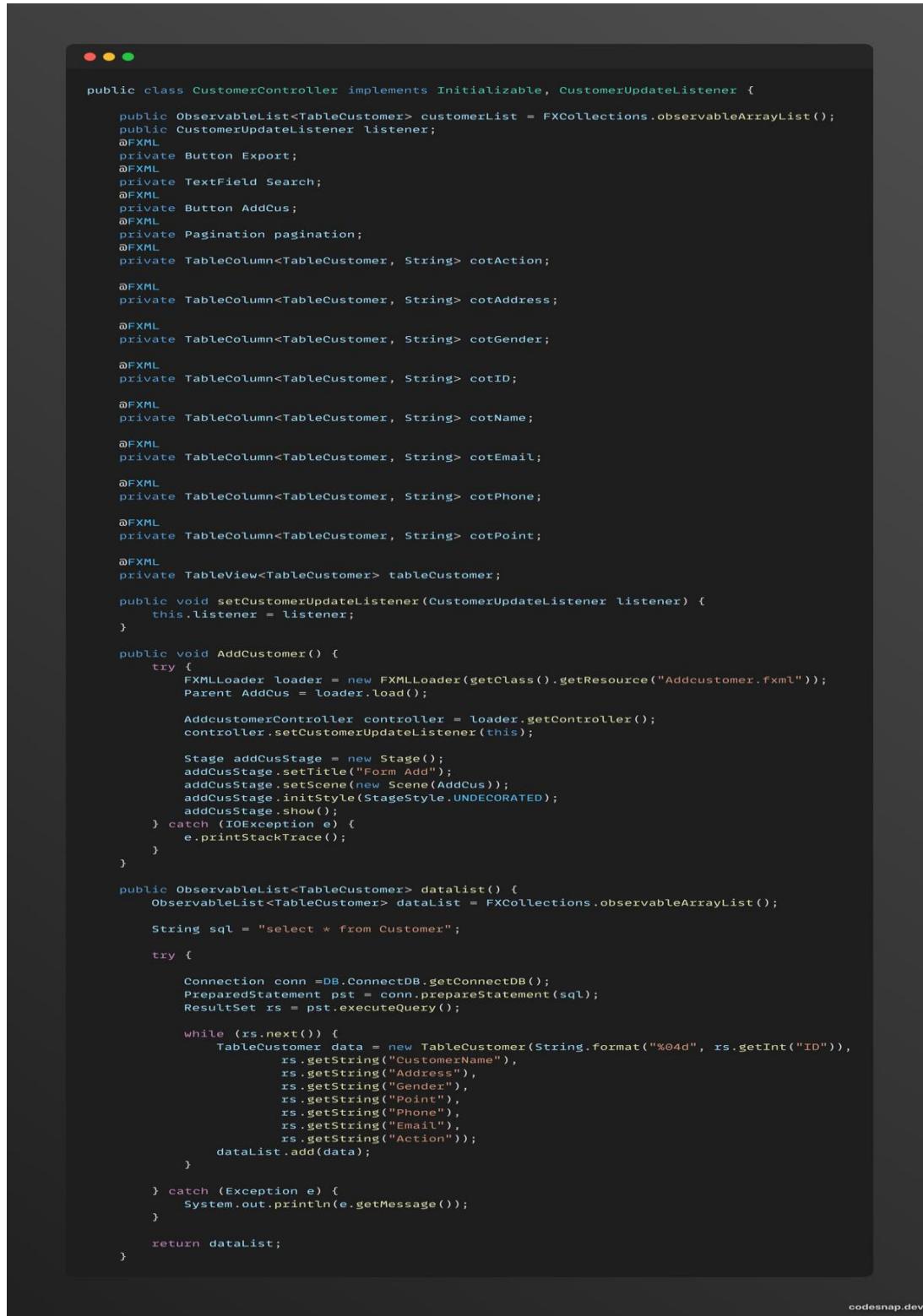
```

```

1  private int totalBills;
2  private int totalRevenue;
3
4  public void updateTotalBillsAndRevenue() {
5      SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");
6      String currentDate = sdf.format(new Date());
7
8      String countSql = "SELECT COUNT(*) AS TotalBills FROM Bills WHERE BillDate = ?";
9      String sumSql = "SELECT SUM(TotalPrice) AS TotalRevenue FROM Bills WHERE BillDate = ?";
10
11     try {
12
13         Connection conn = DB.ConnectDB.getConnectDB();
14
15         PreparedStatement countPst = conn.prepareStatement(countSql);
16         countPst.setString(1, currentDate);
17         ResultSet countRs = countPst.executeQuery();
18         if (countRs.next()) {
19             totalBills = countRs.getInt("TotalBills");
20         }
21
22         countPst.close();
23
24         // Sum total revenue
25         PreparedStatement sumPst = conn.prepareStatement(sumSql);
26         sumPst.setString(1, currentDate);
27         ResultSet sumRs = sumPst.executeQuery();
28         if (sumRs.next()) {
29             totalRevenue = sumRs.getInt("TotalRevenue");
30         }
31
32         sumPst.close();
33         conn.close();
34     } catch (SQLException e) {
35         e.printStackTrace();
36     }
37 }
38
39 // Getter methods for totalBills and totalRevenue
40 public int getTotalBills() {
41     return totalBills;
42 }
43
44 public int getTotalRevenue() {
45     return totalRevenue;
46 }
47
48 public void exportToPDF() {
49     AlertMessage alert = new AlertMessage();
50     FileChooser fileChooser = new FileChooser();
51     fileChooser.setTitle("Save PDF");
52     String fontPath = "src/font/arial.ttf";
53     fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("PDF Files", "*.pdf"));
54     File selectedfile = fileChooser.showSaveDialog(Export.getScene().getWindow());
55
56     if (selectedFile != null) {
57         try {
58
59             PdfFont font = PdfFontFactory.createFont(fontPath, PdfEncodings.IDENTITY_H, true);
60
61             PdfWriter writer = new PdfWriter(selectedFile.getAbsolutePath());
62             PdfDocument pdf = new PdfDocument(writer);
63             Document document = new Document(pdf);
64
65             Paragraph title = new Paragraph("Bills Transaction").setBold().setFontSize(18);
66             document.add(title);
67
68             // Tạo tiêu đề cho các cột
69
70             float[] columnWidths = {100, 100, 100, 100, 100, 100};
71             Table table = new Table(columnWidths);
72
73             TableView<TableBills> tableView = tableBills;
74             for (TableColumn<TableBills> column : tableView.getColumns()) {
75
76                 table.addCell(new Cell().add(new Paragraph(column.getText()).setFont(font)));
77             }
78         }
79     }
80
81
82     ObservableList<TableBills> dataList = tableView.getItems();
83     for (TableBills data : dataList) {
84         table.addCell(new Cell().add(new Paragraph(data.getBillID())));
85         table.addCell(new Cell().add(new Paragraph(data.getCustomerName())));
86         table.addCell(new Cell().add(new Paragraph(data.getCustomerPhone())));
87         table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getTotalPrice()))));
88         table.addCell(new Cell().add(new Paragraph(data.getPaymentMethod())));
89         table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getBillDate()))));
90
91     }
92     document.add(table);
93     document.close();
94     Desktop.getDesktop().open(selectedFile);
95
96     alert.successMessage("Export successful!");
97 } catch (IOException e) {
98     alert.errorMessage("Export failed: " + e.getMessage());
99 }
100 }
101 }
102 }
103

```

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;
    }
}

```

codesnap.dev

8.1 Code

```

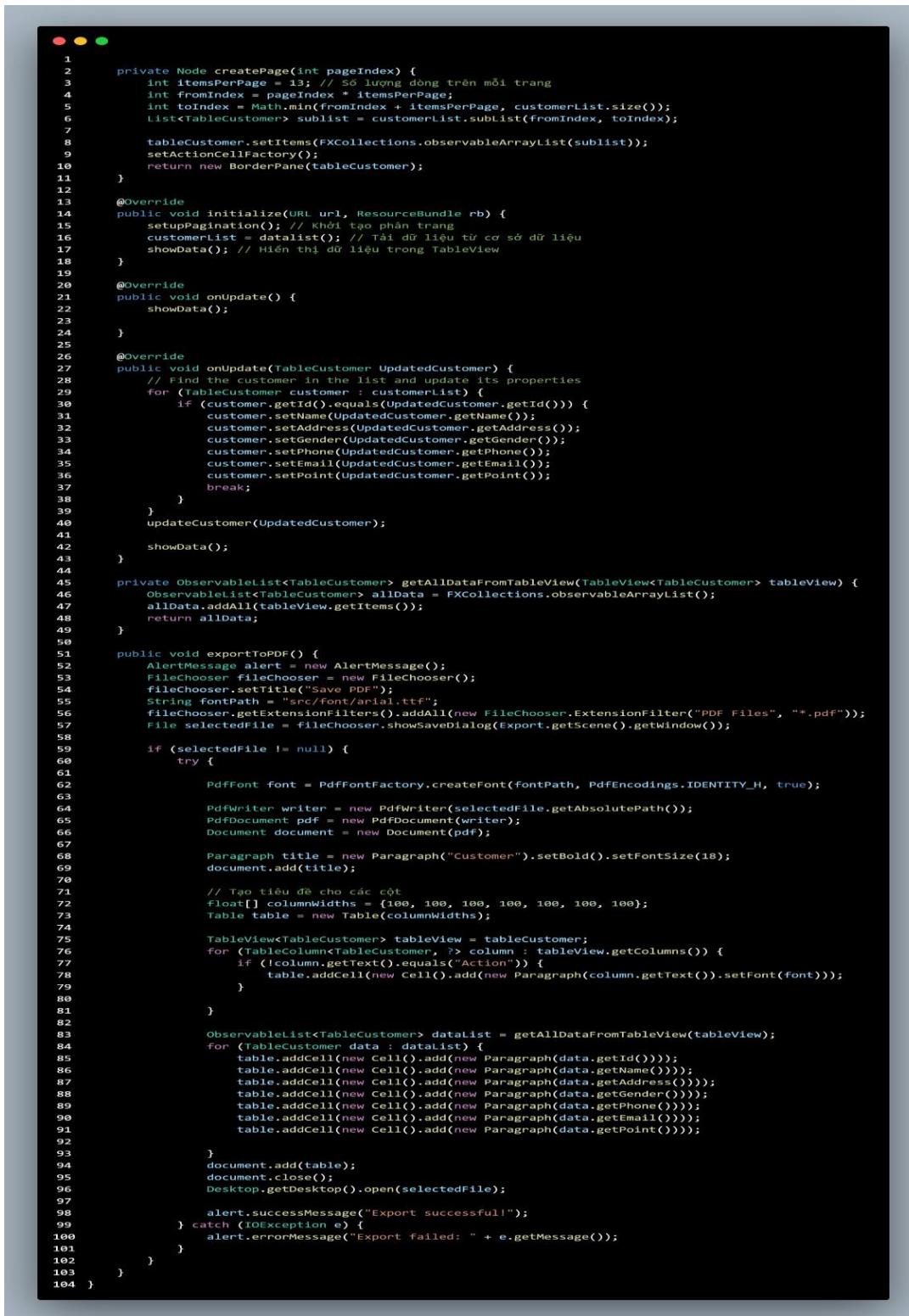
1  public void showData() {
2      // Update observable list from database
3      customerList = dataList();
4
5      // Set cell value factories and cell factories
6      cotID.setCellValueFactory(new PropertyValueFactory<>("id"));
7      cotName.setCellValueFactory(new PropertyValueFactory<>("name"));
8      cotAddress.setCellValueFactory(new PropertyValueFactory<>("address"));
9      cotGender.setCellValueFactory(new PropertyValueFactory<>("gender"));
10     cotPhone.setCellValueFactory(new PropertyValueFactory<>("phone"));
11     cotEmail.setCellValueFactory(new PropertyValueFactory<>("email"));
12     cotPoint.setCellValueFactory(new PropertyValueFactory<>("point"));
13     cotAction.setCellValueFactory(new PropertyValueFactory<>("action"));
14
15     // Set cell factories for action buttons
16     setActionCellFactory();
17
18     // Set TableView items and height
19     tableCustomer.setItems(customerList);
20     tableCustomer.setFixedCellSize(40); // Fixed cell height
21     tableCustomer.prefHeightProperty().bind(Bindings.size(tableCustomer.getItems()).multiply(tableCustomer.getFixedCellSize()).add(30));
22
23     // Update pagination and set to first page
24     pagination.setPageCount(getPageCount());
25     pagination.setPageFactory(this::createPage);
26     pagination.setCurrentPageIndex(0);
27 }
28
29 public void setActionCellFactory() {
30     Callback<TableColumn<TableCustomer, String>, TableCell<TableCustomer, String>> cellFactory
31         = new Callback<TableColumn<TableCustomer, String>, TableCell<TableCustomer, String>>() {
32         @Override
33         public TableCell<TableCustomer, String> call(Final TableColumn<TableCustomer, String> param) {
34             final TableCell<TableCustomer, String> cell = new TableCell<>() {
35                 private final Button updateButton = new Button("Update");
36                 private final Button deleteButton = new Button("Delete");
37                 private final HBox hbox = new HBox(10, updateButton, deleteButton);
38
39                 {
40                     // Button styling and actions
41                     updateButton.getStyleClass().add("update-button");
42                     deleteButton.getStyleClass().add("delete-button");
43
44                     updateButton.setOnAction(event -> {
45                         TableCustomer customer = getTableView().getItems().get(getIndex());
46                         try {
47                             FXMLLoader loader = new FXMLLoader(getClass().getResource("UpdateCustomer.fxml"));
48                             Parent updateCustomerParent = loader.load();
49
50                             UpdateCustomerController controller = loader.getController();
51                             controller.setCustomer(customer);
52                             controller.setCustomerUpdateListener(CustomerController.this); // Pass listener reference
53
54                             Stage updateStage = new Stage();
55                             updateStage.setTitle("Update Customer");
56                             updateStage.setScene(new Scene(updateCustomerParent));
57                             updateStage.initStyle(StageStyle.UNDECORATED);
58                             updateStage.show();
59
60                         } catch (IOException e) {
61                             e.printStackTrace();
62                         }
63                     });
64
65                     deleteButton.setOnAction(event -> {
66                         TableCustomer customer = getTableView().getItems().get(getIndex());
67                         deleteCustomer(customer);
68                     });
69
70                     StackPane.setAlignment(updateButton, Pos.CENTER);
71                     StackPane.setAlignment(deleteButton, Pos.CENTER);
72                 }
73
74             @Override
75             public void updateItem(String item, boolean empty) {
76                 super.updateItem(item, empty);
77
78                 // Set the cell's graphic based on whether the cell is empty
79                 if (empty) {
80                     setGraphic(null);
81                 } else {
82                     setGraphic(hbox);
83                 }
84             }
85         };
86
87         return cell;
88     };
89 }
90
91 // Set cell factory for action column
92 cotAction.setCellFactory(cellFactory);
93 }
```

```

1  @FXML
2  public void deleteCustomer(TableCustomer customer) {
3      AlertMessage alert = new AlertMessage();
4      String sql = "DELETE FROM Customer WHERE ID = ?";
5      try {
6
7          Connection conn = DB.ConnectDB.getConnectDB();
8          PreparedStatement pst = conn.prepareStatement(sql);
9          pst.setInt(1, Integer.parseInt(customer.getId()));
10         pst.executeUpdate();
11
12         // Remove customer from ObservableList and update TableView
13         customerList.remove(customer);
14         showData(); // Update TableView
15         alert.successMessage("Customer deleted successfully");
16
17     } catch (Exception e) {
18         e.printStackTrace();
19         alert.errorMessage("Error deleting customer: " + e.getMessage());
20     }
21 }
22
23 @FXML
24 public void updateCustomer(TableCustomer updatedCustomer) {
25     AlertMessage alert = new AlertMessage();
26     String sql = "UPDATE Customer SET CustomerName=?, Address=?, Gender=?, Phone=?, Email=?, Point=? WHERE ID=?";
27
28     try {
29
30         Connection conn = DB.ConnectDB.getConnectDB();
31         PreparedStatement pst = conn.prepareStatement(sql);
32         pst.setString(1, updatedCustomer.getName());
33         pst.setString(2, updatedCustomer.getAddress());
34         pst.setString(3, updatedCustomer.getGender());
35         pst.setString(4, updatedCustomer.getPhone());
36         pst.setString(5, updatedCustomer.getEmail());
37         pst.setString(6, updatedCustomer.getPoint());
38         pst.setString(7, updatedCustomer.getAction());
39         pst.setInt(7, Integer.parseInt(updatedCustomer.getId())); // ID is int type in database
40
41         int affectedRows = pst.executeUpdate();
42         if (affectedRows > 0) {
43
44             if (listener != null) {
45                 listener.onUpdate(updatedCustomer); // Notify listener to update TableView
46             }
47             else {
48                 System.out.println("Failed to update customer.");
49             }
50
51         } catch (SQLException e) {
52             System.out.println("Error updating customer: " + e.getMessage());
53         }
54     }
55
56     // Refresh the TableView to display updated data
57     // tableCustomer.getItems().clear();
58     // tableCustomer.getItems().addAll(customerList);
59     // }
60
61     @FXML
62     private void handleSearch() {
63         String searchText = normalizeSearchText(Search.getText());
64
65         if (searchText.isEmpty()) {
66             // If search text is empty, show all customers
67             tableCustomer.setItems(customerList);
68             pagination.setPageFactory(this::createPage);
69             return;
70         }
71         String[] keywords = searchText.split("\\s+");
72
73         ObservableList<TableCustomer> filteredList = FXCollections.observableArrayList();
74
75         for (TableCustomer customer : customerList) {
76             String customerName = normalizeSearchText(customer.getName());
77             boolean matchesAllKeywords = true;
78
79             // Check if all keywords are found in customer name
80             for (String keyword : keywords) {
81                 if (!customerName.contains(keyword)) {
82                     matchesAllKeywords = false;
83                     break;
84                 }
85             }
86
87             if (matchesAllKeywords) {
88                 filteredList.add(customer);
89             }
90         }
91
92         // Reset TableView with the filtered list
93         tableCustomer.setItems(filteredList);
94     }
95 }

```

```
1 private String normalize searchText(String text) {
2     // Normalize text: remove accents and convert to lowercase
3     String normalizedText = Normalizer.normalize(text, Normalizer.Form.NFD)
4         .replaceAll("\\p{InCombiningDiacriticalMarks}+", "");
5     return normalizedText.toLowerCase().trim();
6 }
7
8 public void refreshCustomerList() {
9     customerList.clear(); // Xóa danh sách khách hàng hiện tại
10    customerList.addAll(dataList()); // Cập nhật lại danh sách từ cơ sở dữ liệu
11 }
12
13
14 public TableCustomer findCustomerByPhone(String phone) {
15     refreshCustomerList();
16     for (TableCustomer customer : customerList) {
17         if (customer.getPhone().equals(phone)) {
18             return customer;
19         }
20     }
21     return null; // Trả về null nếu không tìm thấy khách hàng
22 }
23
24 public void updateCustomerPoints(String phoneNumber, int pointsToAdd) {
25     // Tìm khách hàng dựa trên số điện thoại
26     TableCustomer customer = findCustomerByPhone(phoneNumber);
27     if (customer != null) {
28         // Cập nhật điểm tích lũy cho khách hàng
29         int currentPoints = Integer.parseInt(customer.getPoint());
30         int newPoints = currentPoints + pointsToAdd;
31
32         // Update điểm tích lũy vào cơ sở dữ liệu
33         String sql = "UPDATE Customer SET Point = ? WHERE Phone = ?";
34         try {
35
36             Connection conn = DB.ConnectDB.getConnectDB();
37             PreparedStatement pst = conn.prepareStatement(sql);
38             pst.setInt(1, newPoints);
39             pst.setString(2, phoneNumber);
40             pst.executeUpdate();
41
42         } catch (SQLException e) {
43             System.out.println(e.getMessage());
44         }
45     } else {
46         System.out.println("Không tìm thấy khách hàng với số điện thoại " + phoneNumber);
47     }
48 }
49
50 private static final int itemsPerPage = 13; // Số lượng mục trên mỗi trang
51
52 private ObservableList<TableCustomer> paginatedData; // Danh sách dữ liệu được phân trang
53
54 private void setupPagination() {
55     int pageCount = getPageCount();
56     pagination.setPageCount(pageCount); // Thiết lập số lượng trang
57
58     pagination.setPageFactory(this::createPage); // Thiết lập page factory để tạo nội dung từng trang
59 }
60
61 private int getPageCount() {
62     // Return the number of pages based on your data and itemsPerPage
63     // For example:
64     int itemsPerPage = 13; // Số lượng dòng trên mỗi trang
65     int pageCount = (int) Math.ceil((double) customerList.size() / itemsPerPage);
66     return pageCount;
67 }
```



```

1  private Node createPage(int pageIndex) {
2      int itemsPerPage = 13; // Số lượng dòng trên mỗi trang
3      int fromIndex = pageIndex * itemsPerPage;
4      int toIndex = Math.min(fromIndex + itemsPerPage, customerList.size());
5      List<TableCustomer> sublist = customerList.subList(fromIndex, toIndex);
6
7      tableCustomer.setItems(FXCollections.observableArrayList(sublist));
8      setActionCellFactory();
9      return new BorderPane(tableCustomer);
10 }
11 }
12
13 @Override
14 public void initialize(URL url, ResourceBundle rb) {
15     setupPagination(); // Khởi tạo phân trang
16     customerList = dataList(); // Tải dữ liệu từ cơ sở dữ liệu
17     showData(); // Hiển thị dữ liệu trong TableView
18 }
19
20 @Override
21 public void onUpdate() {
22     showData();
23 }
24 }
25
26 @Override
27 public void onUpdate(TableCustomer UpdatedCustomer) {
28     // Find the customer in the list and update its properties
29     for (TableCustomer customer : customerList) {
30         if (customer.getId().equals(UpdatedCustomer.getId())) {
31             customer.setName(UpdatedCustomer.getName());
32             customer.setAddress(UpdatedCustomer.getAddress());
33             customer.setGender(UpdatedCustomer.getGender());
34             customer.setPhone(UpdatedCustomer.getPhone());
35             customer.setEmail(UpdatedCustomer.getEmail());
36             customer.setPoint(UpdatedCustomer.getPoint());
37             break;
38         }
39     }
40     updateCustomer(UpdatedCustomer);
41 }
42
43 showData();
44
45 private ObservableList<TableCustomer> getAllDataFromTableView(TableView<TableCustomer> tableView) {
46     ObservableList<TableCustomer> allData = FXCollections.observableArrayList();
47     allData.addAll(tableView.getItems());
48     return allData;
49 }
50
51 public void exportToPDF() {
52     AlertMessage alert = new AlertMessage();
53     FileChooser fileChooser = new FileChooser();
54     fileChooser.setTitle("Save PDF");
55     String fontPath = "src/font/arial.ttf";
56     fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("PDF Files", "*.pdf"));
57     File selectedFile = fileChooser.showSaveDialog(Export.getScene().getWindow());
58
59     if (selectedFile != null) {
60         try {
61             PdfFont font = PdfFontFactory.createFont(fontPath, PdfEncodings.IDENTITY_H, true);
62
63             PdfWriter writer = new PdfWriter(selectedFile.getAbsolutePath());
64             PdfDocument pdf = new PdfDocument(writer);
65             Document document = new Document(pdf);
66
67             Paragraph title = new Paragraph("Customer").setBold().setFontSize(18);
68             document.add(title);
69
70             // Tạo tiêu đề cho các cột
71             float[] columnWidths = {100, 100, 100, 100, 100, 100, 100};
72             Table table = new Table(columnWidths);
73
74             TableView<TableCustomer> tableView = tableCustomer;
75             for (TableColumn<TableCustomer, ?> column : tableView.getColumns()) {
76                 if (column.getText().equals("Action")) {
77                     table.addCell(new Cell().add(new Paragraph(column.getText()).setFont(font)));
78                 }
79             }
80         }
81     }
82
83     ObservableList<TableCustomer> dataList = getAllDataFromTableView(tableView);
84     for (TableCustomer data : dataList) {
85         table.addCell(new Cell().add(new Paragraph(data.getId())));
86         table.addCell(new Cell().add(new Paragraph(data.getName())));
87         table.addCell(new Cell().add(new Paragraph(data.getAddress())));
88         table.addCell(new Cell().add(new Paragraph(data.getGender())));
89         table.addCell(new Cell().add(new Paragraph(data.getPhone())));
90         table.addCell(new Cell().add(new Paragraph(data.getEmail())));
91         table.addCell(new Cell().add(new Paragraph(data.getPoint())));
92     }
93     document.add(table);
94     document.close();
95     Desktop.getDesktop().open(selectedFile);
96
97     alert.successMessage("Export successful!");
98 } catch (IOException e) {
99     alert.errorMessage("Export failed: " + e.getMessage());
100 }
101 }
102 }
103 }
104 }

```

9 Order Transaction

```

1 public class OrderTransactionController implements Initializable {
2
3     @FXML
4     private RadioButton PaymentCard;
5
6     @FXML
7     private RadioButton PaymentCash;
8
9     @FXML
10    private TextField OrderSearchItem;
11    @FXML
12    private Text TotalPrice;
13    @FXML
14    private TextField OrderSearchIdus;
15
16    private CustomerController customerController = new CustomerController();
17
18 // Phương thức để inject CustomerController vào OrderTransactionController
19    public void setCustomerController(CustomerController customerController) {
20        this.customerController = customerController;
21    }
22
23    @FXML
24    private TableColumn<TableProduct, String> Column_Batch;
25
26    @FXML
27    private TableColumn<TableProduct, Date> Column_ExpDate;
28
29    @FXML
30    private TableColumn<TableProduct, String> Column_ItemID;
31
32    @FXML
33    private TableColumn<TableProduct, String> Column_ItemName;
34
35    @FXML
36    private TableColumn<TableProduct, String> Column_Item;
37
38    @FXML
39    private TableColumn<TableProduct, Integer> Column_Price;
40
41    @FXML
42    private TableColumn<TableProduct, Integer> Column_Quantity;
43
44    @FXML
45    private TableView<TableProduct> tableProduct;
46
47    @FXML
48    private VBox vbox;
49
50    private ObservableList<TableProduct> productList = FXCollections.observableArrayList();
51
52    public ObservableList<TableProduct> dataList() {
53        ObservableList<TableProduct> dataList = FXCollections.observableArrayList();
54
55        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 "
56        + "FROM tblProduct p "
57        + "INNER JOIN tblProductDetails pd ON p.MedicineID = pd.MedicineID "
58        + "WHERE pd.Status <> 'Unavailable'";
59
60        try {
61            Connection conn = DB.ConnectDB.getConnectDB();
62            PreparedStatement pst = conn.prepareStatement(sql);
63            ResultSet rs = pst.executeQuery();
64
65            while (rs.next()) {
66
67                String unit = rs.getString("Item");
68                Integer quantity = rs.getInt("Quantity3");
69                if (unit == null || unit.isEmpty()) {
70                    unit = rs.getString("Subunit");
71                    quantity = rs.getInt("Quantity2");
72                    if (unit == null || unit.isEmpty()) {
73                        unit = rs.getString("Unit");
74                        quantity = rs.getInt("Quantity1");
75                    }
76                }
77            }
78
79            TableProduct data = new TableProduct(
80                rs.getString("MedicineID"),
81                rs.getString("MedicineName"),
82                rs.getString("Barcode"),
83                rs.getString("BatchNo"),
84                rs.getDate("ExpireDate"),
85                unit,
86                rs.getInt("SellingPrice"),
87                quantity
88            );
89            dataList.add(data);
90        }
91        conn.close();
92    } catch (Exception e) {
93        System.out.println(e.getMessage());
94    }
95
96    return dataList;
97 }

```

9.1 Code

```

1 private void showdata() {
2     productList = dataList();
3     Column_ItemID.setCellValueFactory(new PropertyValueFactory<>("productId"));
4     Column_ItemName.setCellValueFactory(new PropertyValueFactory<>("productName"));
5     Column_Batch.setCellValueFactory(new PropertyValueFactory<>("batchNo"));
6     Column_ExpDate.setCellValueFactory(new PropertyValueFactory<>("expDate"));
7     Column_Item.setCellValueFactory(new PropertyValueFactory<>("item"));
8     Column_Price.setCellValueFactory(new PropertyValueFactory<>("unitPrice"));
9     Column_Quantity.setCellValueFactory(new PropertyValueFactory<>("quantityInStock"));
10
11    tableProduct.setItems(productList);
12 }
13
14 public AnchorPane createProductAnchorPane(String itemName, String itemPrice, String itemType, String medicineID) {
15     // Tạo AnchorPane mới
16     AnchorPane anchorPane = new AnchorPane();
17     anchorPane.setPrefHeight(75.0);
18     anchorPane.setPrefWidth(290.0);
19     anchorPane.setStyle("-fx-border-color: blue; -fx-border-radius: 8px; -fx-border-width: 1px;");
20     anchorPane.setMinWidth(290.0);
21     anchorPane.setMaxWidth(290.0);
22     anchorPane.setMinHeight(75.0);
23     anchorPane.setMaxHeight(75.0);
24
25     // Tao các thành phần trong AnchorPane
26     Text itemNameText = new Text(itemName);
27     itemNameText.setId("itemNameText");
28     itemNameText.setLayoutX(14.0);
29     itemNameText.setLayoutY(18.0);
30     itemNameText.setFont(Font.font("Arial Black", 12.0));
31     anchorPane.getChildren().add(itemNameText);
32
33     Text medicineIdText = new Text(medicineID);
34     medicineIdText.setId("medicineID");
35     medicineIdText.setLayoutX(200.0);
36     medicineIdText.setLayoutY(18.0);
37     medicineIdText.setFont(Font.font("Arial Black", 12.0));
38     anchorPane.getChildren().add(medicineIdText);
39
40     Text quantityText = new Text("Quantity");
41     quantityText.setLayoutX(14.0);
42     quantityText.setLayoutY(41.0);
43     anchorPane.getChildren().add(quantityText);
44
45     TextField quantityField = new TextField("i");
46     quantityField.setId("quantityField");
47     quantityField.setLayoutX(64.0);
48     quantityField.setLayoutY(20.0);
49     quantityField.setPrefHeight(18.0);
50     quantityField.setPrefWidth(45.0);
51     quantityField.setStyle("-fx-text-fill: black; -fx-border-color: black; -fx-border-width: 1px; -fx-border-radius: 2px; -fx-alignment: center;");
52     anchorPane.getChildren().add(quantityField);
53
54     Text unitText = new Text(itemType);
55     unitText.setId("itemTypeText");
56     unitText.setLayoutX(115.0);
57     unitText.setLayoutY(41.0);
58     anchorPane.getChildren().add(unitText);
59
60     Text priceText = new Text("Price:");
61     priceText.setLayoutX(170.0);
62     priceText.setLayoutY(67.0);
63     priceText.setFont(Font.font("Arial Bold", 14.0));
64     anchorPane.getChildren().add(priceText);
65
66     Text itemPriceText = new Text(itemPrice + "đ");
67     itemPriceText.setId("unitPriceText");
68     itemPriceText.setLayoutX(220.0);
69     itemPriceText.setLayoutY(67.0);
70     itemPriceText.setFont(Font.font("Arial Bold", 14.0));
71     anchorPane.getChildren().add(itemPriceText);
72
73     FontAwesomeIcon trashIcon = new FontAwesomeIcon();
74     trashIcon.setGlyphName("TRASH");
75     trashIcon.setLayoutX(14.0);
76     trashIcon.setLayoutY(69.0);
77     trashIcon.setSize("20px");
78     anchorPane.getChildren().add(trashIcon);
79
80     VBox.setMargin(anchorPane, new Insets(2.0));

```

```
1 trashIcon.setOnMouseClicked(event -> {
2     Node source = (Node) event.getSource();
3     AnchorPane paneToRemove = (AnchorPane) source.getParent();
4
5     Text itemNameTextToRemove = (Text) paneToRemove.lookup("#itemNameText");
6     if (itemNameTextToRemove != null) {
7         String itemNameToRemove = itemNameTextToRemove.getText().trim();
8
9         // Tìm và xóa sản phẩm trong selectedProducts dựa trên productName
10        TableProduct productToRemove = null;
11        for (TableProduct product : selectedProducts) {
12            if (product.getProductName().equals(itemNameToRemove)) {
13                productToRemove = product;
14                break;
15            }
16        }
17
18        if (productToRemove != null) {
19            selectedProducts.remove(productToRemove);
20        }
21
22        vbox.getChildren().remove(paneToRemove);
23        calculateTotalPrice();
24
25        // Cho phép thêm sản phẩm mới sau khi xóa
26        vbox.setDisable(false); // Cho phép sự kiện double-click hoạt động lại
27    }
28 });
29
30 // Thêm sự kiện thay đổi số lượng
31 quantityField.textProperty().addListener((observable, oldValue, newValue) -> {
32     try {
33         int quantity = Integer.parseInt(newValue);
34         if (quantity < 0) {
35             quantityField.setText(oldValue);
36             return;
37         }
38         double totalPrice = quantity * Double.parseDouble(itemPrice);
39         itemPriceText.setText(totalPrice + "đ");
40
41         // Gọi phương thức tính tổng giá sau khi cập nhật giá
42         calculateTotalPrice();
43     } catch (NumberFormatException e) {
44         if (!newValue.isEmpty()) {
45             quantityField.setText(oldValue);
46         }
47     }
48 });
49
50     return anchorPane;
51 }
52
53 private List<TableProduct> selectedProducts = new ArrayList<>();
54
55 @FXML
56 public void Table(MouseEvent event) {
57     if (!vbox.isDisabled() && event.getButton() == MouseButton.PRIMARY && event.getClickCount() == 2) {
58         TableProduct product = tableProduct.getSelectionModel().getSelectedItem();
59         if (product != null && !selectedProducts.contains(product)) {
60             // Tạo AnchorPane cho sản phẩm
61             String itemType = product.getItem();
62             AnchorPane anchorPane = createProductAnchorPane(product.getProductName(), String.valueOf(product.getUnitPrice()), itemType, product.getProductId());
63             vbox.getChildren().add(anchorPane); // Thêm anchorPane vào VBox
64
65             // Thêm sản phẩm vào danh sách đã chọn
66             selectedProducts.add(product);
67
68             // Tính toán tổng giá sau khi thêm sản phẩm
69             calculateTotalPrice();
70         }
71     }
72 }
```

```
1 @FXML
2     private void searchItem() {
3         String searchItem = OrderSearchItem.getText().trim().replaceAll("\\s+", ""); // Loại bỏ khoảng trắng trong từ khóa tìm kiếm
4         productList.clear(); // Xóa dữ liệu cũ
5
6         if (searchItem.isEmpty()) {
7             // Nếu không có gì được nhập, hiển thị lại toàn bộ dữ liệu
8             productList.addAll(dataList);
9         } else {
10            // Nếu có từ khóa tìm kiếm, thực hiện truy vấn để lấy dữ liệu tương ứng
11            String sql = "SELECT p.MedicineID, pd.Barcode, p.Item ,p.MedicineName, pd.BatchNo, pd.ExpireDate, p.SellingPrice, pd.Quantity1 "
12                + "FROM tblProduct p "
13                + "INNER JOIN tblProductDetails pd ON p.MedicineID = pd.MedicineID";
14
15            try {
16
17                Connection conn = DB.ConnectDB.getConnectDB();
18                PreparedStatement pst = conn.prepareStatement(sql);
19                ResultSet rs = pst.executeQuery();
20
21                while (rs.next()) {
22                    String medicineName = rs.getString("MedicineName").replaceAll("\\s+", ""); // Loại bỏ khoảng trắng trong dữ liệu
23                    String barCode = rs.getString("Barcode").replaceAll("\\s+", "");
24                    if (medicineName.toLowerCase().contains(searchItem.toLowerCase()) || barCode.toLowerCase().contains(searchItem.toLowerCase())) {
25                        TableProduct data = new TableProduct(rs.getString("MedicineID"),
26                            rs.getString("MedicineName"),
27                            rs.getString("Barcode"),
28                            rs.getString("BatchNo"),
29                            rs.getDate("ExpireDate"),
30                            rs.getString("Item"),
31                            rs.getInt("SellingPrice"),
32                            rs.getInt("Quantity1"));
33                        productList.add(data);
34                    }
35                }
36            } catch (Exception e) {
37                System.out.println(e.getMessage());
38            }
39        }
40
41        tableProduct.setItems(productList);
42    }
43
44    private void calculateTotalPrice() {
45        double totalPrice = 0;
46
47        for (Node node : vbox.getChildren()) {
48            if (node instanceof AnchorPane) {
49                AnchorPane anchorPane = (AnchorPane) node;
50                for (Node child : anchorPane.getChildren()) {
51                    if (child instanceof Text && ((Text) child).getText().contains("đ")) {
52                        String priceText = ((Text) child).getText().replace("đ", "").trim();
53                        try {
54                            double price = Double.parseDouble(priceText);
55                            totalPrice += price;
56                        } catch (NumberFormatException e) {
57                            // Ignore invalid price text
58                        }
59                    }
60                }
61            }
62        }
63
64        TotalPrice.setText(totalPrice + "đ");
65    }
```

```

1  private void updateQuantityInDatabase(String itemName, int quantityToDeduct, String unit) {
2      String sql = "";
3      String quantityColumn = "";
4
5      if (unit.equalsIgnoreCase("Pill")) {
6          quantityColumn = "Quantity3";
7      } else if (unit.equalsIgnoreCase("Blister")) {
8          quantityColumn = "Quantity2";
9      } else if (unit.equalsIgnoreCase("Box")) {
10         quantityColumn = "Quantity1";
11     } else {
12         // Handle default case if needed
13     }
14
15     if (!quantityColumn.isEmpty()) {
16         sql = "UPDATE pd SET " + quantityColumn + " = " + quantityColumn + " - ? "
17             + "FROM tblProductDetails pd "
18             + "JOIN tblProduct p ON pd.MedicineID = p.MedicineID "
19             + "WHERE p.MedicineName = ? AND " + quantityColumn + " >= ?";
20     } else {
21         // Handle case where quantityColumn is not set
22         System.out.println("Không xác định được cột số lượng phù hợp cho đơn vị " + unit);
23         return;
24     }
25
26     try {
27
28         Connection conn = DB.ConnectDB.getConnectDB();
29         PreparedStatement pst = conn.prepareStatement(sql);
30
31         // Thiết lập các tham số trong câu lệnh SQL
32         pst.setInt(1, quantityToDeduct);
33         pst.setString(2, itemName);
34         pst.setInt(3, quantityToDeduct); // Ensure enough quantity to deduct
35
36         int updatedRows = pst.executeUpdate();
37         if (updatedRows > 0) {
38             System.out.println("Cập nhật số lượng thành công.");
39             showdata(); // Refresh the table data
40         } else {
41             System.out.println("Không thành công khi cập nhật số lượng sản phẩm trong cơ sở dữ liệu.");
42         }
43         pst.close();
44         conn.close();
45     } catch (Exception e) {
46         e.printStackTrace();
47     }
48 }
49
50 @Override
51 public void initialize(URL url, ResourceBundle rb) {
52     showdata();
53
54     tableProduct.setOnMouseClicked(this::Table); // Ensure the event is set here
55 }
56
57 @FXML
58 void cancel_all(ActionEvent event) {
59     vbox.getChildren().clear();
60     vbox.setDisable(false);
61     selectedProducts.clear();
62     // Gọi phương thức tính tổng giá sau khi xóa hết anchorPane
63     calculateTotalPrice();
64 }
65
66 @FXML
67 private void handlePayment() {
68     AlertMessage alert = new AlertMessage();
69
70     String phone = OrderSearchcus.getText().trim();
71     TableCustomer customer = null;
72     int pointsToAdd = 0;
73
74     // Kiểm tra nếu số điện thoại không rỗng thì mới tiến hành tìm kiếm khách hàng
75     if (Iphone.isEmpty()) {
76         customerController = new CustomerController();
77         // Gọi phương thức findCustomerByPhone của customerController để tìm khách hàng
78         customer = customerController.findCustomerByPhone(phone);
79         if (customer != null) {
80             // Tính tổng thanh toán từ các sản phẩm
81             int totalPayment = calculateTotalPayment();
82             // Tính điểm tích lũy dựa trên tổng thanh toán
83             pointsToAdd = totalPayment / 10000; // Mỗi 10,000 VND tương ứng với 1 điểm
84             // Cập nhật điểm tích lũy cho khách hàng
85             customerController.updateCustomerPoints(customer.getPhone(), pointsToAdd);
86         } else {
87             // Hiển thị thông báo lỗi nếu không tìm thấy khách hàng với số điện thoại nhập vào
88             alert.errorMessage("Không tìm thấy khách hàng với số điện thoại: " + phone);
89             return; // Thoát khỏi phương thức nếu không tìm thấy khách hàng
90         }
91     }
92     int totalPayment = calculateTotalPayment();
93     String billID = generateBillID();
94     String paymentMethod = PaymentCash.isSelected() ? "Cash" : "Card";
95     boolean billInserted = false;
96     String CName = "";
97     String CPhone = "";
98     String cid = "";
99     if (customer != null) {
100        CName = customer.getName();
101        CPhone = customer.getPhone();
102        cid = customer.getId();
103    }
104 }

```

```
1 // Thêm hóa đơn vào cơ sở dữ liệu
2     billInserted = insertIntoBills(billID, CName, CPhone, totalPayment, paymentMethod, new Date(System.currentTimeMillis()), cid);
3
4 // Kiểm tra kết quả và xử lý khi không thêm được hóa đơn
5 if (!billInserted) {
6     // Xử lý khi không thêm được hóa đơn vào cơ sở dữ liệu
7     System.out.println("Failed to insert bill into database.");
8     // Hoặc có thể hiển thị thông báo lỗi cho người dùng
9 } else {
10    // Xử lý khi thêm hóa đơn thành công (ví dụ: hiển thị thông báo thành công)
11    System.out.println("Bill inserted successfully.");
12 }
13 for (Node node : vbox.getChildren()) {
14     if (node instanceof AnchorPane) {
15         AnchorPane anchorPane = (AnchorPane) node;
16
17         // Lấy thông tin sản phẩm từ AnchorPane
18         Text itemNameText = (Text) anchorPane.lookup("#itemNameText");
19         TextField quantityField = (TextField) anchorPane.lookup("#quantityField");
20         Text unitPriceText = (Text) anchorPane.lookup("#unitPriceText");
21         Text itemTypeText = (Text) anchorPane.lookup("#itemTypeText");
22         Text medicineIdText = (Text) anchorPane.lookup("#medicineID");
23
24         if (itemNameText != null && quantityField != null && unitPriceText != null && itemTypeText != null && medicineIdText != null) {
25             String itemName = itemNameText.getText().trim();
26             int quantity = 0;
27             try {
28                 quantity = Integer.parseInt(quantityField.getText().trim());
29             } catch (NumberFormatException e) {
30                 e.printStackTrace();
31                 continue; // Bỏ qua nếu không thể chuyển đổi thành số nguyên
32             }
33
34             int unitPrice = 0;
35             try {
36                 String priceText = unitPriceText.getText().replace("đ", "").trim();
37                 double unitPriceDouble = Double.parseDouble(priceText);
38                 unitPrice = (int) unitPriceDouble;
39             } catch (NumberFormatException e) {
40                 e.printStackTrace();
41                 continue;
42             }
43
44             String itemType = itemTypeText.getText().trim();
45             String productId = medicineIdText.getText().trim();
46             // Thêm vào bảng BillItems
47             boolean itemInserted = insertIntoBillItems(billID, itemName, quantity, unitPrice, itemType, productId);
48             if (!itemInserted) {
49                 alert.errorMessage("Lỗi khi lưu thông tin chi tiết hóa đơn vào cơ sở dữ liệu.");
50                 return;
51             }
52         }
53     }
54 }
```

10 Expiry

```
1  public class ProductExpiryInfoController {  
2      @FXML  
3      private Button exit;  
4      @FXML  
5      private TableColumn<ProductExpiryTable, Long> daysLeftColumn;  
6  
7      @FXML  
8      private TableColumn<ProductExpiryTable, LocalDate> expireDateColumn;  
9  
10     @FXML  
11     private TableView<ProductExpiryTable> expiringProductsTable;  
12  
13     @FXML  
14     private TableColumn<ProductExpiryTable, LocalDate> manufacturingDateColumn;  
15  
16     @FXML  
17     private TableColumn<ProductExpiryTable, String> medicineIDColumn;  
18  
19     @FXML  
20     private TableColumn<ProductExpiryTable, String> medicineNameColumn;  
21  
22     private List<ProductExpiryTable> expiringProducts;  
23  
24     public void initialize() {  
25         // Initialize table columns  
26         medicineIDColumn.setCellValueFactory(new PropertyValueFactory<>("medicineID"));  
27         medicineNameColumn.setCellValueFactory(new PropertyValueFactory<>("medicineName"));  
28         manufacturingDateColumn.setCellValueFactory(new PropertyValueFactory<>("manufacturingDate"));  
29         expireDateColumn.setCellValueFactory(new PropertyValueFactory<>("expireDate"));  
30         daysLeftColumn.setCellValueFactory(new PropertyValueFactory<>("daysLeft"));  
31  
32         exit.setOnMouseClicked(event -> {  
33             Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();  
34             stage.close();  
35         });  
36     }  
37  
38     public void setExpiringProducts(List<ProductExpiryTable> products) {  
39         expiringProductsTable.setItems(FXCollections.observableArrayList(products));  
40     }  
41 }  
42 }  
43 }
```

11 Update Customer

```
1 public class UpdateCustomerController implements Initializable {
2
3     @FXML
4     private ImageView exit;
5     @FXML
6     private TextField txtName;
7
8     @FXML
9     private TextField txtAddress;
10
11    @FXML
12    private TextField txtPhone;
13
14    @FXML
15    private TextField txtEmail;
16
17    @FXML
18    private RadioButton radioMale;
19
20    @FXML
21    private RadioButton radioFemale;
22
23    @FXML
24    private TextArea txtPoint;
25
26    private TableView<TableCustomer> tableCustomer;
27    public TableCustomer customer;
28    public CustomerUpdateListener listener;
29
30    public void setCustomerUpdateListener(CustomerUpdateListener listener) {
31        this.listener = listener;
32    }
33
34
35    public void setCustomer(TableCustomer customer) {
36        this.customer = customer;
37
38        txtName.setText(customer.getName());
39        txtAddress.setText(customer.getAddress());
40        txtPhone.setText(customer.getPhone());
41        txtEmail.setText(customer.getEmail());
42        if ("Male".equalsIgnoreCase(customer.getGender())) {
43            radioMale.setSelected(true);
44        } else if ("Female".equalsIgnoreCase(customer.getGender())) {
45            radioFemale.setSelected(true);
46        }
47        txtPoint.setText(customer.getPoint());
48    }
49
50    @FXML
51    public void handleUpdate() {
52        AlertMessage alert = new AlertMessage();
53
54        // Kiểm tra dữ liệu nhập vào
55        if (txtName.getText().isEmpty() || txtAddress.getText().isEmpty() || txtPhone.getText().isEmpty() || txtEmail.getText().isEmpty()
56            || txtPoint.getText().isEmpty()) {
57            alert.errorMessage("Please enter complete information");
58            return;
59        }
60
61        // Kiểm tra xem 'Nam' hoặc 'Nữ' đã được chọn chưa
62        if (!(radioMale.isSelected() ^ radioFemale.isSelected())) {
63            alert.errorMessage("Please select 'Male' or 'Female'");
64            return;
65        }
66
67        // Kiểm tra định dạng email có hợp lệ không
68        if (!txtEmail.getText().matches("[a-zA-Z0-9_.+&*-]+(?:\\.[a-zA-Z0-9_-+&*-]+)*(?:[a-zA-Z0-9_-+\\.]|[a-zA-Z]{2,7})$")) {
69            alert.errorMessage("Please enter the correct Email");
70            return;
71        }
72    }
}
```

```
1 // Kiểm tra số điện thoại trùng
2 String checkPhoneQuery = "SELECT * FROM Customer WHERE Phone = ? AND ID != ?";
3 try {
4
5     Connection conn = DB.ConnectDB.getConnectDB();
6     PreparedStatement checkPhoneStmt = conn.prepareStatement(checkPhoneQuery);
7     checkPhoneStmt.setString(1, txtPhone.getText());
8     checkPhoneStmt.setString(2, customer.getId()); // customer.getId() là phương thức lấy ID của khách hàng
9     ResultSet existingPhone = checkPhoneStmt.executeQuery();
10    if (existingPhone.next()) {
11        alert.errorMessage("Phone number already exists");
12        return; // Thoát phương thức nếu số điện thoại đã tồn tại
13    }
14 } catch (SQLException ex) {
15     ex.printStackTrace();
16 }
17
18 // Kiểm tra email trùng
19 String checkEmailQuery = "SELECT * FROM Customer WHERE Email = ? AND ID != ?";
20 try {
21
22     Connection conn = DB.ConnectDB.getConnectDB();
23     PreparedStatement checkEmailStmt = conn.prepareStatement(checkEmailQuery);
24     checkEmailStmt.setString(1, txtEmail.getText());
25     checkEmailStmt.setString(2, customer.getId()); // customer.getId() là phương thức lấy ID của khách hàng
26     ResultSet existingEmail = checkEmailStmt.executeQuery();
27     if (existingEmail.next()) {
28         alert.errorMessage("Email address already exists");
29         return; // Thoát phương thức nếu địa chỉ email đã tồn tại
30     }
31 } catch (SQLException ex) {
32     ex.printStackTrace();
33 }
34
35 // Cập nhật thông tin khách hàng
36 customer.setName(txtName.getText());
37 customer.setAddress(txtAddress.getText());
38 customer.setPhone(txtPhone.getText());
39 customer.setEmail(txtEmail.getText());
40 customer.setGender(radioMale.isSelected() ? "Nam" : "Nữ");
41 customer.setPoint(txtPoint.getText());
42
43 // Gọi phương thức onUpdate của listener để thông báo cập nhật
44 if (listener != null) {
45     listener.onUpdate(customer);
46 }
47
48 // Hiển thị thông báo cập nhật thành công và đóng cửa sổ
49 alert.successMessage("Successfully update Customers");
50 Stage stage = (Stage) txtName.getScene().getWindow();
51 stage.close();
52 }
53
54 @FXML
55 public void handleCancel() {
56     Stage stage = (Stage) txtName.getScene().getWindow();
57     stage.close();
58 }
59
60 @Override
61 public void initialize(URL url, ResourceBundle rb) {
62     exit.setOnMouseClicked(event -> {
63         Stage stage = (Stage) ((javafx.scene.Node) event.getSource()).getScene().getWindow();
64         stage.close();
65     });
66 }
67 }
68 }
```

12 Update Drug

```
1  public class UpdateDrugController implements Initializable {
2
3      @FXML
4      private Button btnUpdateClose;
5
6      @FXML
7      private Button btnCancelUpdate;
8
9      @FXML
10     private Button btnUpdate;
11
12     @FXML
13     private CheckBox ckUPItem;
14
15     @FXML
16     private CheckBox ckUPsubunit;
17
18     @FXML
19     private TextField txtUPAPI;
20
21     @FXML
22     private TextField txtUPCostprice;
23
24     @FXML
25     private TextField txtUPDrugName;
26
27     @FXML
28     private TextField txtUPItem;
29
30     @FXML
31     private TextField txtUPItempersubunit;
32
33     @FXML
34     private TextField txtUPQuantity;
35
36     @FXML
37     private TextField txtUPSellingprice;
38
39     @FXML
40     private TextField txtUPSubunit;
41
42     @FXML
43     private TextField txtUPSubunitsperunit;
44
45     @FXML
46     private TextField txtUPUnit;
47
48     @FXML
49     void Cancelbtn(ActionEvent event) {
50         updateDruglistForm();
51         Stage stage = (Stage) btnCancelUpdate.getScene().getWindow();
52         stage.close();
53     }
54
55     @FXML
56     void Closebtn(ActionEvent event) {
57         updateDruglistForm();
58         Stage stage = (Stage) btnUpdateClose.getScene().getWindow();
59         stage.close();
60     }
```

12.1 Code

```

1  private String MedicineID;
2
3  Message alert = new Message();
4  private Connection conn;
5  private PreparedStatement ps;
6  private ResultSet rs;
7
8  private boolean isPositiveNumber(String value) {
9      try {
10          int intValue = Integer.parseInt(value);
11          return intValue >= 0;
12      } catch (Exception e) {
13          return false;
14      }
15  }
16
17  private boolean containsOnlyLetters(String text) {
18      return text.matches("[a-zA-Z\\s]+");
19  }
20
21
22  private void updateDruglistForm() {
23
24      DruglistController druglistController = ControllerHolder.getInstance().getDruglistController();
25      if (druglistController != null) {
26          druglistController.showData();
27          druglistController.Pagination();
28      }
29  }
30
31  public boolean isDrugNameExists(String drugName, String medicineID) {
32      try {
33          String sqlCheck = "SELECT COUNT(*) FROM tblProduct WHERE MedicineName = ? AND MedicineID <> ?";
34          conn = DB.ConnectDB.getConnection();
35          ps = conn.prepareStatement(sqlCheck);
36          ps.setString(1, drugName);
37          ps.setString(2, medicineID);
38          rs = ps.executeQuery();
39          if (rs.next()) {
40              int count = rs.getInt(1);
41              return count > 0;
42          }
43      } catch (Exception e) {
44          e.printStackTrace();
45          alert.errorMessage("Error checking drug name: " + e.getMessage());
46      } finally {
47          try {
48              if (rs != null) {
49                  rs.close();
50              }
51              if (ps != null) {
52                  ps.close();
53              }
54              if (conn != null) {
55                  conn.close();
56              }
57          } catch (Exception e) {
58              e.printStackTrace();
59          }
60      }
61      return false;
62  }
63
64  public void updatebtn() {
65      if (txtUPDrugName.getText().isEmpty()
66          || txtUPAPI.getText().isEmpty()
67          || txtUPAPI.getText().isEmpty()
68          || txtUPCostprice.getText().isEmpty()
69          || txtUPSellingprice.getText().isEmpty()
70          || txtUPQuantity.getText().isEmpty()
71          || txtUPUnit.getText().isEmpty())
72          alert.errorMessage("Please fill all blank fields");
73      } else if (containsOnlyLetters(txtUPCostprice.getText())
74          || containsOnlyLetters(txtUPSellingprice.getText())
75          || containsOnlyLetters(txtUPQuantity.getText())){
76          alert.errorMessage("Please enter Number for price only");
77      } else if (containsOnlyLetters(txtUPQuantity.getText())){
78          alert.errorMessage("Please enter Number for Quantity only");
79      } else if (!isPositiveNumber(txtUPCostprice.getText())
80          || !isPositiveNumber(txtUPSellingprice.getText())
81          || Integer.parseInt(txtUPQuantity.getText()) <= 0) {
82          alert.errorMessage("Please enter valid positive number");
83      } else if (!containsOnlyLetters(txtUPUnit.getText())) {
84          System.out.println(txtUPUnit.getText());
85          alert.errorMessage("Please enter Letters for Unit 1 only");
86      } else {
87
88          String DrugName = txtUPDrugName.getText();
89          if (isDrugNameExists(DrugName, MedicineID)) {
90              alert.errorMessage("Drug name already exists. Please enter a different name.");
91              return;
92          }
93          String API = txtUPAPI.getText();
94          int CostPrice = Integer.parseInt(txtUPCostprice.getText());
95          int SellingPrice = Integer.parseInt(txtUPSellingprice.getText());
96          int Quantity = Integer.parseInt(txtUPQuantity.getText());
97          String Unit = txtUPUnit.getText();
98          int SubunitsPerUnit = 0;
99          String Subunit = "";
100         int ItemsPerSubunit = 0;
101         String Item = "";
102

```



```

1 if (ckUbsubunit.isSelected()) {
2     if (txtUbsubunit.getText().isBlank()
3         || txtUbsubunitsperunit.getText().isBlank()) {
4         alert.errorMessage("Please fill all blank fields");
5         return;
6     } else if (containsOnlyLetters(txtUbsubunitsperunit.getText())) {
7         alert.errorMessage("Please enter Number for 'Quantity only'");
8         return;
9     } else if ((Integer.parseInt(txtUbsubunitsperunit.getText()) <= 0)) {
10        alert.errorMessage("Please enter valid positive number");
11        return;
12    } else if (!containsOnlyLetters(txtUbsubunit.getText())) {
13        alert.errorMessage("Please enter letters for Unit only");
14        return;
15    }
16    Subunitsperunit = Integer.parseInt(txtUbsubunitsperunit.getText());
17    Subunit = txtUbsubunit.getText();
18    if (ckItem.isSelected()) {
19        if (txtUitmpersubunit.getText().isBlank()
20            || txtUitem.getText().isBlank()) {
21            alert.errorMessage("Please fill all blank fields");
22            return;
23        } else if (containsOnlyLetters(txtUitmpersubunit.getText())) {
24            alert.errorMessage("Please enter Number for 'Quantity only'");
25            return;
26        } else if ((Integer.parseInt(txtUitmpersubunit.getText()) <= 0)) {
27            alert.errorMessage("Please enter valid positive number");
28            return;
29        } else if (!containsOnlyLetters(txtUitem.getText())) {
30            alert.errorMessage("Please enter letters for Unit only");
31            return;
32        }
33        Itemspersubunit = Integer.parseInt(txtUitmpersubunit.getText());
34        Item = txtUitem.getText();
35    }
36 }
37 String sqlDrugListUpdate = "UPDATE tbProduct SET Medicinename = ? , API = ?, Quantity = ?, Unit = ?, Subunitsperunit = ?, Subunit = ?, Itemspersubunit = ?, Item = ?, Costprice = ?, SellingPrice = ? WHERE MedicineID = '' + MedicineID + '' ";
38 conn = DB.getConnection();
39 try {
40     if (alert.confirmMessage("Are you want to update Medicine ID: " + MedicineID + "?")) {
41         ps = conn.prepareStatement(sqlDrugListUpdate);
42         ps.setString(1, Drugname);
43         ps.setString(2, API);
44         ps.setInt(3, Quantity);
45         ps.setString(4, Unit);
46         ps.setString(5, Subunitsperunit);
47         ps.setInt(6, Subunit);
48         ps.setInt(7, Itemspersubunit);
49         ps.setString(8, Item);
50         ps.setInt(9, Costprice);
51         ps.setInt(10, SellingPrice);
52         ps.setString(11, MedicineID);
53         alert.successMessage("Update successfully");
54         Stage stage = (Stage) btnUpdate.getScene().getWindow();
55         stage.close();
56         updateDruglistForm();
57     } else {
58         alert.errorMessage("Failed to update drug");
59     }
60 }
61 catch (Exception e) {
62     e.printStackTrace();
63     alert.errorMessage("Error adding drug: " + e.getMessage());
64 }
65 }
66 }

```

```
1 public void setField() {
2     MedicineID = Data.dl_MedicineID;
3     txtUPDrugName.setText(Data.dl_MedicineName);
4     txtUPAPI.setText(Data.dl_API);
5     txtUPCostprice.setText(String.valueOf(Data.dl_Costprice));
6     txtUPSellingprice.setText(String.valueOf(Data.dl_Sellingprice));
7     txtUPQuantity.setText(String.valueOf(Data.dl_Quantity));
8     txtUPUnit.setText(Data.dl_Unit);
9     txtUPSubunitsperunit.setText(String.valueOf(Data.dl_Subunitsperunit));
10    txtUPSubunit.setText(Data.dl_Subunit);
11    txtUPItempersubunit.setText(String.valueOf(Data.dl_Itemspersubunit));
12    txtUPItem.setText(Data.dl_Item);
13    if (Data.dl_Subunitsperunit == 0) {
14        txtUPSubunitsperunit.setDisable(true);
15        txtUPSubunit.setDisable(true);
16    }
17    } else {
18        txtUPSubunitsperunit.setDisable(false);
19        txtUPSubunit.setDisable(false);
20        ckUPsubunit.setSelected(true);
21    }
22    }
23    if (Data.dl_Itemspersubunit == 0) {
24        txtUPItempersubunit.setDisable(true);
25        txtUPItem.setDisable(true);
26    } else {
27        txtUPItempersubunit.setDisable(false);
28        txtUPItem.setDisable(false);
29        ckUPitem.setSelected(true);
30        ckUPitem.setDisable(false);
31    }
32
33    ckUPsubunit.selectedProperty().addListener((obs, wasSelected, isNowSelected) -> handleSubunitCheck(isNowSelected));
34    ckUPitem.selectedProperty().addListener((obs, wasSelected, isNowSelected) -> handleItemCheck(isNowSelected));
35 }
36
37 private void handleSubunitCheck(boolean isChecked) {
38    if (isChecked) {
39        txtUPSubunitsperunit.setDisable(false);
40        txtUPSubunit.setDisable(false);
41        ckUPitem.setDisable(false);
42    } else {
43        txtUPSubunitsperunit.setDisable(true);
44        txtUPSubunit.setDisable(true);
45        txtUPSubunitsperunit.clear();
46        txtUPSubunit.clear();
47        ckUPitem.setSelected(false);
48        ckUPitem.setDisable(true);
49        txtUPItempersubunit.setDisable(true);
50        txtUPItem.setDisable(true);
51        txtUPItempersubunit.clear();
52        txtUPItem.clear();
53    }
54 }
55
56 private void handleItemCheck(boolean isChecked) {
57    if (isChecked) {
58        txtUPItempersubunit.setDisable(false);
59        txtUPItem.setDisable(false);
60    } else {
61        txtUPItempersubunit.setDisable(true);
62        txtUPItem.setDisable(true);
63        txtUPItempersubunit.clear();
64        txtUPItem.clear();
65    }
66 }
67
68 @Override
69 public void initialize(URL url, ResourceBundle rb) {
70     setField();
71 }
72 }
73 }
74 }
```

13 Import Medicine Details

```
1  public class ImportMedicineDetailsController implements Initializable {
2
3      @FXML
4      private TableColumn<MedicineDetailsData, Void> MD_Col_AC;
5
6      @FXML
7      private TableColumn<MedicineDetailsData, String> MD_Col_Barcod;
8
9      @FXML
10     private TableColumn<MedicineDetailsData, String> MD_Col_Batchno;
11
12     @FXML
13     private TableColumn<MedicineDetailsData, String> MD_Col_MedicineName;
14
15     @FXML
16     private TableColumn<MedicineDetailsData, Integer> MD_Col_Quantity1;
17
18     @FXML
19     private TableColumn<MedicineDetailsData, Integer> MD_Col_Quantity2;
20
21     @FXML
22     private TableColumn<MedicineDetailsData, Integer> MD_Col_Quantity3;
23
24     @FXML
25     private TableColumn<MedicineDetailsData, String> MD_Col_Status;
26
27     @FXML
28     private TableColumn<MedicineDetailsData, String> MD_Col_Unit1;
29
30     @FXML
31     private TableColumn<MedicineDetailsData, String> MD_Col_Unit2;
32
33     @FXML
34     private TableColumn<MedicineDetailsData, String> MD_Col_Unit3;
35
36     @FXML
37     private TableColumn<MedicineDetailsData, Date> MD_Col_ExpD;
38
39     @FXML
40     private TableColumn<MedicineDetailsData, Date> MD_Col_ManuD;
41
42     @FXML
43     private Button btnBG;
44
45     @FXML
46     private Button btnMDAdd;
47
48     @FXML
49     private Button btnMDUpdate;
50
51     @FXML
52     private Button btnCancel;
53
54     @FXML
55     private Button btnTransHistory;
56
57     @FXML
58     private Pagination pgtMD;
59
60     @FXML
61     private TextField searchMD;
62
63     @FXML
64     private TableView<MedicineDetailsData> tablemedicinedetailslist;
65
66     @FXML
67     private TextField txtMDBBarcode;
68
69     @FXML
70     private TextField txtMDBatchno;
71
72     @FXML
73     private DatePicker txtMDExpireDate;
74
75     @FXML
76     private DatePicker txtMDManDate;
77
78     @FXML
79     private TextField txtMDMedicineName;
80
81     @FXML
82     private ComboBox<String> txtMDStatus;
83
84     @FXML
85     private TextField txtQuan1;
86
87     @FXML
88     private TextField txtQuan2;
89
90     @FXML
91     private TextField txtQuan3;
92
93     @FXML
94     private TextField txtUnit1;
95
96     @FXML
97     private TextField txtUnit2;
98
99     @FXML
100    private TextField txtUnit3;
101
102    private Connection conn;
103    private PreparedStatement ps;
104    private ResultSet rs;
105    Message alert = new M
```

13.1 Code

```

1 public boolean isPositiveNumber(String value) {
2     try {
3         int intValue = Integer.parseInt(value);
4         return intValue > 0;
5     } catch (Exception e) {
6         return false; // Nếu không phải số, trả về false
7     }
8 }
9
10 public boolean containsOnlyLetters(String text) {
11     return text.matches("[a-zA-Z]+");
12 }
13 }
14
15 public ObservableList<MedicineDetailsData> DataList() {
16     ObservableList<MedicineDetailsData> DataList = FXCollections.observableArrayList();
17     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 "
18     + "FROM tbIPProductDetails pd "
19     + "JOIN tbIPProduct p ON pd.MedicineID = p.MedicineID";
20     try {
21         conn = DB.ConnectDB.getConnection();
22         ps = conn.prepareStatement(sql);
23         rs = ps.executeQuery();
24         while (rs.next()) {
25             LocalDate manufacturingDate = rs.getDate("ManufacturingDate").toLocalDate();
26             LocalDate expireDate = rs.getDate("ExpireDate").toLocalDate();
27             MedicineDetailsData data = new MedicineDetailsData(
28                 rs.getInt("ID"),
29                 rs.getString("MedicineID"),
30                 rs.getString("MedicineName"),
31                 rs.getString("Barcode"),
32                 rs.getString("Batchno"),
33                 rs.getInt("Quantity1"),
34                 rs.getString("Unit1"),
35                 rs.getInt("Quantity2"),
36                 rs.getString("Unit2"),
37                 rs.getInt("Quantity3"),
38                 rs.getString("Unit3"),
39                 manufacturingDate,
40                 expireDate,
41                 rs.getString("Status")
42             );
43             DataList.add(data);
44         }
45     } catch (Exception e) {
46         System.out.println(e.getMessage());
47     } finally {
48         try {
49             if (rs != null) {
50                 rs.close();
51             }
52             if (ps != null) {
53                 ps.close();
54             }
55             if (conn != null) {
56                 conn.close();
57             }
58         } catch (SQLException e) {
59             System.out.println("Error closing resources: " + e.getMessage());
60         }
61     }
62     return DataList;
63 }
64
65 public void showData() {
66     ObservableList<MedicineDetailsData> showList = DataList();
67     MD_col_MedicineName.setCellValueFactory(new PropertyValueFactory<>("MedicineName"));
68     MD_col_Barcode.setCellValueFactory(new PropertyValueFactory<>("Barcode"));
69     MD_col_Batchno.setCellValueFactory(new PropertyValueFactory<>("Batchno"));
70     MD_col_Manup.setCellValueFactory(new PropertyValueFactory<>("ManufacturingDate"));
71     MD_col_Expd.setCellValueFactory(new PropertyValueFactory<>("ExpireDate"));
72     MD_col_Quantity1.setCellValueFactory(new PropertyValueFactory<>("Quantity1"));
73     MD_col_Unit1.setCellValueFactory(new PropertyValueFactory<>("Unit1"));
74     MD_col_Quantity2.setCellValueFactory(new PropertyValueFactory<>("Quantity2"));
75     MD_col_Unit2.setCellValueFactory(new PropertyValueFactory<>("Unit2"));
76     MD_col_Quantity3.setCellValueFactory(new PropertyValueFactory<>("Quantity3"));
77     MD_col_Unit3.setCellValueFactory(new PropertyValueFactory<>("Unit3"));
78     MD_col_Status.setCellValueFactory(new PropertyValueFactory<>("Status"));
79 }
```

```
1 MD_Col_AC.setCellFactory(new Callback<TableColumn<MedicineDetailsData, Void>, TableCell<MedicineDetailsData, Void>>() {
2     @Override
3     public TableCell<MedicineDetailsData, Void> call(final TableColumn<MedicineDetailsData, Void> param) {
4         final TableCell<MedicineDetailsData, Void> cell = new TableCell<MedicineDetailsData, Void>() {
5             private final Button btnDelete = new Button("Delete");
6
7             {
8                 btnDelete.getStyleClass().add("btn_delete");
9                 btnDelete.setOnAction((ActionEvent event) -> {
10                     try {
11                         MedicineDetailsData pdata = getTableView().getItems().get(getIndex());
12                         if (alert.confirmMessage("Are you sure you want to delete ID: " + pdata.getID() + "?")) {
13                             deleteData(pdata.getID());
14                             showData(); // Refresh table
15                             Pagination(); // Refresh Pagination
16                         }
17                     } catch (Exception e) {
18                         e.printStackTrace();
19                     }
20                 });
21                 btnDelete.setAlignment(Pos.CENTER);
22             }
23
24             @Override
25             public void updateItem(Void item, boolean empty) {
26                 super.updateItem(item, empty);
27                 if (empty) {
28                     setGraphic(null);
29                 } else {
30                     HBox hbox = new HBox(5);
31                     hbox.getChildren().addAll(btnDelete);
32                     hbox.setAlignment(Pos.CENTER);
33                     setGraphic(hbox);
34                 }
35             }
36         };
37         return cell;
38     }
39 });
40
41     tablemedicinedetailslist.setItems(showList);
42 }
43
44
45     public void MedicineDetailsSelect() {
46         tablemedicinedetailslist.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) -> {
47             if (newSelection != null) {
48                 // Lấy dữ liệu từ hàng được chọn và đổ vào các TextField và Datepicker
49                 MedicineDetailsData selectedData = newSelection;
50                 txtMDMedicineName.setText(selectedData.getMedicineName());
51                 txtMDMedicineName.setDisable(true);
52                 txtMDBarcode.setText(selectedData.getBarcode());
53                 txtMDBatchno.setText(selectedData.getBatchNo());
54                 txtMDManDate.setValue(selectedData.getManufacturingDate());
55                 txtMDExpireDate.setValue(selectedData.getExpireDate());
56                 txtQuan1.setText(String.valueOf(selectedData.getQuantity1()));
57                 txtUnit1.setText(selectedData.getUnit1());
58                 txtQuan2.setText(String.valueOf(selectedData.getQuantity2()));
59                 txtUnit2.setText(selectedData.getUnit2());
60                 txtQuan3.setText(String.valueOf(selectedData.getQuantity3()));
61                 txtUnit3.setText(selectedData.getUnit3());
62                 txtMDStatus.getSelectionModel().select(selectedData.getStatus());
63             }
64         });
65     }
66 }
```

```
1 // Disable fields based on txtUnit2 and txtUnit3 conditions
2     if (txtUnit2.getText().isEmpty()) {
3         txtQuan2.setDisable(true);
4         txtUnit2.setDisable(true);
5         txtQuan3.setDisable(true);
6         txtUnit3.setDisable(true);
7     } else {
8         txtQuan2.setDisable(false);
9         txtUnit2.setDisable(false);
10    }
11
12    if (txtUnit3.getText().isEmpty()) {
13        txtQuan3.setDisable(true);
14        txtUnit3.setDisable(true);
15    } else {
16        txtQuan3.setDisable(false);
17        txtUnit3.setDisable(false);
18    }
19 } else {
20     // Nếu không có hàng được chọn, xóa trống các TextField và DatePicker
21     clearFields();
22 }
23 });
24 }
25
26 @FXML
27 void MDCancel(ActionEvent event) {
28     showData();
29     Pagination();
30     clearFields();
31 }
32
33 private void deleteData(int id) {
34     String sqlDeleteProductDetails = "DELETE FROM tblProductDetails WHERE ID = ?";
35
36     try {
37
38         conn = DB.ConnectDB.getConnectDB();
39         ps = conn.prepareStatement(sqlDeleteProductDetails);
40         ps.setInt(1, id);
41         ps.executeUpdate();
42         ps.close();
43
44         alert.successMessage("Deleted Successfully");
45     } catch (SQLException e) {
46         e.printStackTrace();
47     } finally {
48
49         try {
50             if (ps != null) {
51                 ps.close();
52             }
53             if (conn != null) {
54                 conn.close();
55             }
56         } catch (SQLException e) {
57             e.printStackTrace();
58         }
59     }
60 }
61 private void setupAutoSuggestion() {
62     TextFields.bindAutoCompletion(txtMDMedicineName, suggestionRequest -> {
63         ObservableList<String> suggestions = FXCollections.observableArrayList();
64         String query = suggestionRequest.getUserText();
65         if (query.length() > 1) { // Fetch suggestions when input length is greater than 2
66             suggestions.addAll(fetchMedicineNames(query));
67         }
68         return suggestions;
69     }).setOnAutoCompleted(event -> {
70         String selectedMedicineName = event.getCompletion();
71         fetchAndSetMedicineID(selectedMedicineName);
72     });
73 }
74 }
```

```
1  private ObservableList<String> fetchMedicineNames(String query) {
2      ObservableList<String> medicineNames = FXCollections.observableArrayList();
3      String sql = "SELECT MedicineName FROM tblProduct WHERE MedicineName LIKE ?";
4      try {
5          conn = DB.ConnectDB.getConnectDB();
6          ps = conn.prepareStatement(sql);
7          ps.setString(1, "%" + query + "%");
8          rs = ps.executeQuery();
9          while (rs.next()) {
10              medicineNames.add(rs.getString("MedicineName"));
11          }
12      } catch (SQLException e) {
13          System.out.println("Error fetching medicine names: " + e.getMessage());
14      } finally {
15          try {
16              if (rs != null) {
17                  rs.close();
18              }
19              if (ps != null) {
20                  ps.close();
21              }
22              if (conn != null) {
23                  conn.close();
24              }
25          } catch (SQLException e) {
26              System.out.println("Error closing resources: " + e.getMessage());
27          }
28      }
29      return medicineNames;
30  }
31
32  private void fetchAndSetMedicineID(String medicineName) {
33      String sql = "SELECT MedicineID, Unit, Subunit, Item FROM tblProduct WHERE MedicineName = ?";
34      try {
35          conn = DB.ConnectDB.getConnectDB();
36          ps = conn.prepareStatement(sql);
37          ps.setString(1, medicineName);
38          rs = ps.executeQuery();
39          if (rs.next()) {
40              String medicineID = rs.getString("MedicineID");
41              String unit = rs.getString("Unit");
42              String subunit = rs.getString("Subunit");
43              String item = rs.getString("Item");
44              txtUnit1.setText(unit);
45
46              if (subunit != null && !subunit.isEmpty()) {
47                  txtUnit2.setText(subunit);
48                  txtQuan2.setDisable(false);
49                  txtUnit2.setDisable(false);
50              } else {
51                  txtQuan2.setDisable(true);
52                  txtUnit2.setDisable(true);
53              }
54
55              if (item != null && !item.isEmpty()) {
56                  txtUnits3.setText(item);
57                  txtQuan3.setDisable(false);
58                  txtUnit3.setDisable(false);
59              } else {
60                  txtQuan3.setDisable(true);
61                  txtUnit3.setDisable(true);
62              }
63          }
64      } catch (SQLException e) {
65          System.out.println("Error fetching medicine ID: " + e.getMessage());
66      } finally {
67          try {
68              if (rs != null) {
69                  rs.close();
70              }
71              if (ps != null) {
72                  ps.close();
73              }
74              if (conn != null) {
75                  conn.close();
76              }
77          } catch (SQLException e) {
78              System.out.println("Error closing resources: " + e.getMessage());
79          }
80      }
81  }
```

```

1  private String fetchAndValidateMedicineID(String medicineName) {
2      String sql = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
3      try {
4          conn = DB.ConnectDB.getConnectDB();
5          ps = conn.prepareStatement(sql);
6          ps.setString(1, medicineName);
7          rs = ps.executeQuery();
8          if (rs.next()) {
9              return rs.getString("MedicineID");
10         }
11     } catch (SQLException e) {
12         System.out.println("Error fetching medicine ID: " + e.getMessage());
13     } finally {
14         try {
15             if (rs != null) {
16                 rs.close();
17             }
18             if (ps != null) {
19                 ps.close();
20             }
21             if (conn != null) {
22                 conn.close();
23             }
24         } catch (SQLException e) {
25             System.out.println("Error closing resources: " + e.getMessage());
26         }
27     }
28     return null; // Return null if MedicineID not found
29 }
30
31 @FXML
32 void AddMD(ActionEvent event) {
33     // Fetch or validate MedicineID
34
35     String medicineID = fetchAndValidateMedicineID(txtMDMedicineName.getText());
36
37     if (medicineID == null) {
38         // Handle case when MedicineID is not found or invalid
39         alert.errorMessage("Invalid Medicine Name. Please select a valid medicine.");
40         return;
41     }
42
43     if (txtMDMedicineName.getText().isEmpty()
44         || txtMDBarcode.getText().isEmpty()
45         || txtMDBatchno.getText().isEmpty()
46         || txtMDManDate.getValue() == null
47         || txtMDExpireDate.getValue() == null
48         || txtQuan1.getText().isEmpty()
49         || txtUnit1.getText().isEmpty()
50         || txtMDStatus.getValue() == null) {
51         alert.errorMessage("Please fill all blank fields");
52         return;
53     } else if (!containsOnlyLetters(txtQuan1.getText())) {
54         alert.errorMessage("Please enter Number for Quantity only");
55         return;
56     } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
57         alert.errorMessage("Please enter valid positive number");
58         return;
59     } else if (!containsOnlyLetters(txtUnit1.getText())) {
60         alert.errorMessage("Please enter Letters for Unit only");
61         return;
62     }
63     String barcode = txtMDBarcode.getText();
64     String batchNo = txtMDBatchno.getText();
65     LocalDate manufacturingDate = txtMDManDate.getValue();
66     LocalDate expireDate = txtMDExpireDate.getValue();
67     // If (!isValidDate(txtMDManDate.getEditor().getText()) || !isValidDate(txtMDExpireDate.getEditor().getText()))
68     // alert.errorMessage("Invalid date format. Please enter dates in the format dd/MM/yyyy.");
69     //
70     //
71
72     if (manufacturingDate.isAfter(expireDate)) {
73         alert.errorMessage("Expiration date must be after manufacturing date.");
74         return;
75     }
76
77     int quantity1 = Integer.parseInt(txtQuan1.getText());
78     String unit1 = txtUnit1.getText();
79     int quantity2 = 0;
80     String unit2 = "";
81     int quantity3 = 0;
82     String unit3 = "";
83     String status = txtMDStatus.getValue();
84     if (!txtUnit2.isDisable()) {
85         if (txtUnit2.getText().isBlank()
86             || txtQuan2.getText().isBlank()) {
87             alert.errorMessage("Please fill all blank 1 fields");
88             return;
89         } else if (!containsOnlyLetters(txtQuan2.getText())) {
90             alert.errorMessage("Please enter Number for Quantity only");
91             return;
92         } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
93             alert.errorMessage("Please enter valid positive number");
94             return;
95         } else if (!containsOnlyLetters(txtUnit2.getText())) {
96             alert.errorMessage("Please enter Letters for Unit only");
97             return;
98         }

```

```

1  private String fetchAndValidateMedicineID(String medicineName) {
2      String sql = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
3      try {
4          conn = DB.ConnectDB.getConnectDB();
5          ps = conn.prepareStatement(sql);
6          ps.setString(1, medicineName);
7          rs = ps.executeQuery();
8          if (rs.next()) {
9              return rs.getString("MedicineID");
10         }
11     } catch (SQLException e) {
12         System.out.println("Error fetching medicine ID: " + e.getMessage());
13     } finally {
14         try {
15             if (rs != null) {
16                 rs.close();
17             }
18             if (ps != null) {
19                 ps.close();
20             }
21             if (conn != null) {
22                 conn.close();
23             }
24         } catch (SQLException e) {
25             System.out.println("Error closing resources: " + e.getMessage());
26         }
27     }
28     return null; // Return null if MedicineID not found
29 }
30
31 @FXML
32 void AddMD(ActionEvent event) {
33     // Fetch or validate MedicineID
34
35     String medicineID = fetchAndValidateMedicineID(txtMDMedicineName.getText());
36
37     if (medicineID == null) {
38         // Handle case when MedicineID is not found or invalid
39         alert.errorMessage("Invalid Medicine Name. Please select a valid medicine.");
40         return;
41     }
42
43     if (txtMDMedicineName.getText().isEmpty()
44         || txtMDBBarcode.getText().isEmpty()
45         || txtMDBatchno.getText().isEmpty()
46         || txtMDManDate.getValue() == null
47         || txtMDExpireDate.getValue() == null
48         || txtQuan1.getText().isEmpty()
49         || txtUnit1.getText().isEmpty()
50         || txtMDStatus.getValue() == null) {
51         alert.errorMessage("Please fill all blank fields");
52         return;
53     } else if (!containsOnlyLetters(txtQuan1.getText())) {
54         alert.errorMessage("Please enter Number for Quantity only");
55         return;
56     } else if (!Integer.parseInt(txtQuan1.getText()) <= 0) {
57         alert.errorMessage("Please enter valid positive number");
58         return;
59     } else if (!containsOnlyLetters(txtUnit1.getText())) {
60         alert.errorMessage("Please enter Letters for Unit only");
61         return;
62     }
63     String barcode = txtMDBBarcode.getText();
64     String batchNo = txtMDBatchno.getText();
65     LocalDate manufacturingDate = txtMDManDate.getValue();
66     LocalDate expireDate = txtMDExpireDate.getValue();
67     // if (!isValidDate(txtMDManDate.getEditor().getText()) || !isValidDate(txtMDExpireDate.getEditor().getText())) {
68     //     alert.errorMessage("Invalid date format. Please enter dates in the format dd/MM/yyyy.");
69     //     return;
70     //}
71     if (manufacturingDate.isAfter(expireDate)) {
72         alert.errorMessage("Expiration date must be after manufacturing date.");
73         return;
74     }
75
76     int quantity1 = Integer.parseInt(txtQuan1.getText());
77     String unit1 = txtUnit1.getText();
78     int quantity2 = 0;
79     String unit2 = "";
80     int quantity3 = 0;
81     String unit3 = "";
82     String status = txtMDStatus.getValue();
83     if (!txtUnit2.isDisable()) {
84         if (txtUnit2.getText().isBlank()
85             || txtQuan2.getText().isBlank()) {
86             alert.errorMessage("Please fill all blank 1 fields");
87             return;
88         } else if (!containsOnlyLetters(txtQuan2.getText())) {
89             alert.errorMessage("Please enter Number for Quantity only");
90             return;
91         } else if (!Integer.parseInt(txtQuan2.getText()) <= 0) {
92             alert.errorMessage("Please enter valid positive number");
93             return;
94         } else if (!containsOnlyLetters(txtUnit2.getText())) {
95             alert.errorMessage("Please enter Letters for Unit only");
96             return;
97         }
98     }

```

```

1  private String fetchAndValidateMedicineID(String medicineName) {
2      String sql = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
3      try {
4          conn = DB.ConnectDB.getConnectDB();
5          ps = conn.prepareStatement(sql);
6          ps.setString(1, medicineName);
7          rs = ps.executeQuery();
8          if (rs.next()) {
9              return rs.getString("MedicineID");
10         }
11     } catch (SQLException e) {
12         System.out.println("Error fetching medicine ID: " + e.getMessage());
13     } finally {
14         try {
15             if (rs != null) {
16                 rs.close();
17             }
18             if (ps != null) {
19                 ps.close();
20             }
21             if (conn != null) {
22                 conn.close();
23             }
24         } catch (SQLException e) {
25             System.out.println("Error closing resources: " + e.getMessage());
26         }
27     }
28     return null; // Return null if MedicineID not found
29 }
30
31 @FXML
32 void AddMD(ActionEvent event) {
33     // Fetch or validate MedicineID
34
35     String medicineID = fetchAndValidateMedicineID(txtMDMedicineName.getText());
36
37     if (medicineID == null) {
38         // Handle case when MedicineID is not found or invalid
39         alert.errorMessage("Invalid Medicine Name. Please select a valid medicine.");
40         return;
41     }
42
43     if (txtMDMedicineName.getText().isEmpty()
44         || txtMDBBarcode.getText().isEmpty()
45         || txtMDBatchno.getText().isEmpty()
46         || txtMDManDate.getValue() == null
47         || txtMDExpireDate.getValue() == null
48         || txtQuan1.getText().isEmpty()
49         || txtUnit1.getText().isEmpty()
50         || txtMDStatus.getValue() == null) {
51         alert.errorMessage("Please fill all blank fields");
52         return;
53     } else if (!containsOnlyLetters(txtQuan1.getText())) {
54         alert.errorMessage("Please enter Number for Quantity only");
55         return;
56     } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
57         alert.errorMessage("Please enter valid positive number");
58         return;
59     } else if (!containsOnlyLetters(txtUnit1.getText())) {
60         alert.errorMessage("Please enter Letters for Unit only");
61         return;
62     }
63     String barcode = txtMDBBarcode.getText();
64     String batchNo = txtMDBatchno.getText();
65     LocalDate manufacturingDate = txtMDManDate.getValue();
66     LocalDate expireDate = txtMDExpireDate.getValue();
67     // if (!isValidDate(txtMDManDate.getEditor().getText()) || !isValidDate(txtMDExpireDate.getText())) {
68     //     alert.errorMessage("Invalid date format. Please enter dates in the format dd/MM/yyyy.");
69     //     return;
70     //}
71     if (manufacturingDate.isAfter(expireDate)) {
72         alert.errorMessage("Expiration date must be after manufacturing date.");
73         return;
74     }
75
76     int quantity1 = Integer.parseInt(txtQuan1.getText());
77     String unit1 = txtUnit1.getText();
78     int quantity2 = 0;
79     String unit2 = "";
80     int quantity3 = 0;
81     String unit3 = "";
82     String status = txtMDStatus.getValue();
83
84     if (!txtUnit2.isDisable()) {
85         if (txtUnit2.getText().isBlank()
86             || txtQuan2.getText().isBlank()) {
87             alert.errorMessage("Please fill all blank 1 fields");
88             return;
89         } else if (!containsOnlyLetters(txtQuan2.getText())) {
90             alert.errorMessage("Please enter Number for Quantity only");
91             return;
92         } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
93             alert.errorMessage("Please enter valid positive number");
94             return;
95         } else if (!containsOnlyLetters(txtUnit2.getText())) {
96             alert.errorMessage("Please enter Letters for Unit only");
97             return;
98         }
}

```

```

1  private String fetchAndValidateMedicineID(String medicineName) {
2      String sql = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
3      try {
4          conn = DB.ConnectDB.getConnectDB();
5          ps = conn.prepareStatement(sql);
6          ps.setString(1, medicineName);
7          rs = ps.executeQuery();
8          if (rs.next()) {
9              return rs.getString("MedicineID");
10         }
11     } catch (SQLException e) {
12         System.out.println("Error fetching medicine ID: " + e.getMessage());
13     } finally {
14         try {
15             if (rs != null) {
16                 rs.close();
17             }
18             if (ps != null) {
19                 ps.close();
20             }
21             if (conn != null) {
22                 conn.close();
23             }
24         } catch (SQLException e) {
25             System.out.println("Error closing resources: " + e.getMessage());
26         }
27     }
28     return null; // Return null if MedicineID not found
29 }
30
31 @FXML
32 void AddMD(ActionEvent event) {
33     // Fetch or validate MedicineID
34
35     String medicineID = fetchAndValidateMedicineID(txtMDMedicineName.getText());
36
37     if (medicineID == null) {
38         // Handle case when MedicineID is not found or invalid
39         alert.errorMessage("Invalid Medicine Name. Please select a valid medicine.");
40         return;
41     }
42
43     if (txtMDMedicineName.getText().isEmpty()
44         || txtMDBarcode.getText().isEmpty()
45         || txtMDBatchno.getText().isEmpty()
46         || txtMDManDate.getValue() == null
47         || txtMDExpireDate.getValue() == null
48         || txtQuan1.getText().isEmpty()
49         || txtUnit1.getText().isEmpty()
50         || txtMDStatus.getValue() == null) {
51         alert.errorMessage("Please fill all blank fields");
52         return;
53     } else if (containsOnlyLetters(txtQuan1.getText())) {
54         alert.errorMessage("Please enter Number for Quantity only");
55         return;
56     } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
57         alert.errorMessage("Please enter valid positive number");
58         return;
59     } else if (!containsOnlyLetters(txtUnit1.getText())) {
60         alert.errorMessage("Please enter Letters for Unit only");
61         return;
62     }
63     String barcode = txtMDBarcode.getText();
64     String batchNo = txtMDBatchno.getText();
65     LocalDate manufacturingDate = txtMDManDate.getValue();
66     LocalDate expireDate = txtMDExpireDate.getValue();
67     // if (!isValidDate(txtMDManDate.getEditor().getText()) || !isValidDate(txtMDExpireDate.getEditor().getText())) {
68     //     alert.errorMessage("Invalid date format. Please enter dates in the format dd/MM/yyyy.");
69     //     return;
70     //}
71
72     if (manufacturingDate.isAfter(expireDate)) {
73         alert.errorMessage("Expiration date must be after manufacturing date.");
74         return;
75     }
76
77     int quantity1 = Integer.parseInt(txtQuan1.getText());
78     String unit1 = txtUnit1.getText();
79     int quantity2 = 0;
80     String unit2 = "";
81     int quantity3 = 0;
82     String unit3 = "";
83     String status = txtMDStatus.getValue();
84     if (!txtUnit2.isDisable()) {
85         if (txtUnit2.getText().isBlank()
86             || txtQuan2.getText().isBlank()) {
87             alert.errorMessage("Please fill all blank 1 fields");
88             return;
89         } else if (containsOnlyLetters(txtQuan2.getText())) {
90             alert.errorMessage("Please enter Number for Quantity only");
91             return;
92         } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
93             alert.errorMessage("Please enter valid positive number");
94             return;
95         } else if (!containsOnlyLetters(txtUnit2.getText())) {
96             alert.errorMessage("Please enter Letters for Unit only");
97             return;
98         }

```

```
1 quantity2 = Integer.parseInt(txtQuan2.getText());
2     unit2 = txtUnit2.getText();
3     if (!txtUnit3.isDisable()) {
4         if (txtQuan3.getText().isBlank()
5             || txtUnit3.getText().isBlank()) {
6             alert.errorMessage("Please fill all blank fields");
7             return;
8         } else if (!containsOnlyLetters(txtQuan3.getText())) {
9             alert.errorMessage("Please enter Number for Quantity only");
10            return;
11        } else if (Integer.parseInt(txtQuan3.getText()) <= 0) {
12            alert.errorMessage("Please enter valid positive number");
13            return;
14        } else if (!containsOnlyLetters(txtUnit3.getText())) {
15            alert.errorMessage("Please enter Letters for Unit only");
16            return;
17        }
18        quantity3 = Integer.parseInt(txtQuan3.getText());
19        unit3 = txtUnit3.getText();
20    }
21 }
22 if (!isBarcodeUnique(barcode)) {
23     alert.errorMessage("Barcode already exists. Please use a different Barcode.");
24     return;
25 }
26 // Insert data into database
27 String sql = "INSERT INTO tblProductDetails (MedicineID, BarCode, Batchno, Quantity1, Unit1, Quantity2, Unit2, Quantity3, Unit3, ManufacturingDate, ExpireDate, Status) "
28     + "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
29 try {
30     conn = DB.ConnectDB.getConnectDB();
31     ps = conn.prepareStatement(sql);
32     ps.setString(1, medicineID);
33     ps.setString(2, barcode);
34     ps.setString(3, batchNo);
35     ps.setInt(4, quantity1);
36     ps.setString(5, unit1);
37     ps.setInt(6, quantity2);
38     ps.setString(7, unit2);
39     ps.setInt(8, quantity3);
40     ps.setString(9, unit3);
41     ps.setDate(10, Date.valueOf(manufacturingDate));
42     ps.setDate(11, Date.valueOf(expireDate));
43     ps.setString(12, status);
44
45     int result = ps.executeUpdate();
46     if (result > 0) {
47         alert.successMessage("Medicine details added successfully.");
48         showData(); // Refresh table view
49         Pagination(); // Refresh Pagination
50         clearfields(); // Optional: Clear input fields after successful addition
51     } else {
52         alert.errorMessage("Failed to add medicine details.");
53     }
54 } catch (SQLException e) {
55     System.out.println("Error adding medicine details: " + e.getMessage());
56     alert.errorMessage("Error adding medicine details. Please try again.");
57 } finally {
58     try {
59         if (ps != null) {
60             ps.close();
61         }
62         if (conn != null) {
63             conn.close();
64         }
65     } catch (SQLException e) {
66         System.out.println("Error closing resources: " + e.getMessage());
67     }
68 }
69 }
```

```

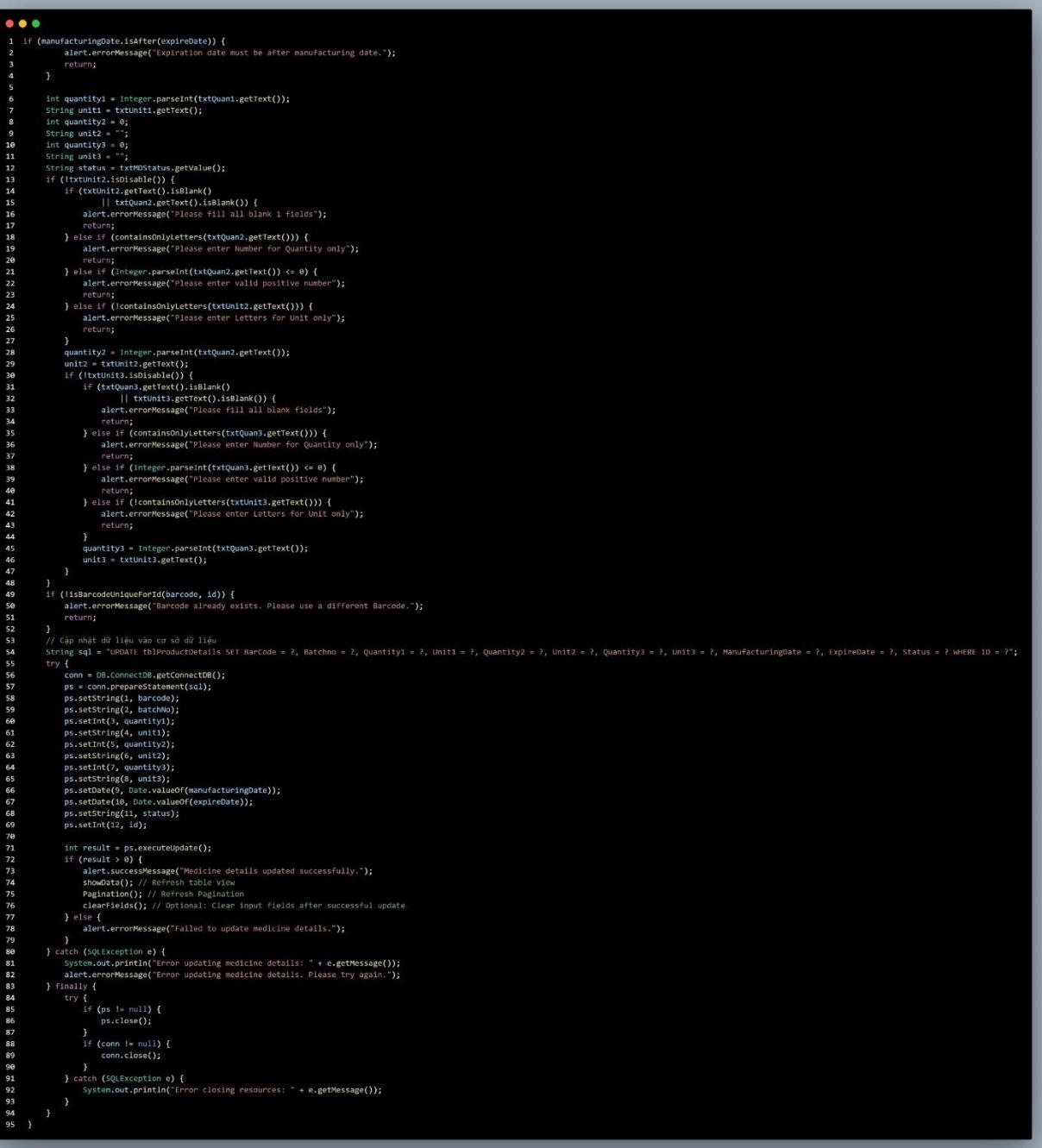
1 private boolean isBarcodeUnique(String barcode) {
2     String sql = "SELECT COUNT(*) AS count FROM tblProductDetails WHERE BarCode = ?";
3     try {
4         conn = DB.ConnectDB.getConnectDB();
5         ps = conn.prepareStatement(sql);
6         ps.setString(1, barcode);
7         rs = ps.executeQuery();
8         if (rs.next()) {
9             int count = rs.getInt("count");
10            return count == 0; // True if no matching records found (Barcode is unique)
11        }
12    } catch (SQLException e) {
13        System.out.println("Error checking Barcode uniqueness: " + e.getMessage());
14    } finally {
15        try {
16            if (rs != null) {
17                rs.close();
18            }
19            if (ps != null) {
20                ps.close();
21            }
22            if (conn != null) {
23                conn.close();
24            }
25        } catch (SQLException e) {
26            System.out.println("Error closing resources: " + e.getMessage());
27        }
28    }
29    return false; // Default to false if an error occurs
30 }
31
32 private void clearFields() {
33     txtMOMedicineName.setDisable(false);
34     txtMOBarcode.clear();
35     txtMOBatchno.clear();
36     txtMOExpireDate.setValue(null);
37     txtMOManDate.setValue(null);
38     txtMOMedicineName.clear();
39     txtQuanti.clear();
40     txtUniti.clear();
41     txtQuan2.clear();
42     txtUnit2.clear();
43     txtQuan3.clear();
44     txtUnits.clear();
45     txtMOSatus.getSelectionModel().clearSelection();
46 }
47
48 //filter
49 private void filterData(String keyword) {
50     ObservableList<MedicineDetailsData> filteredList = FXCollections.observableArrayList();
51     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 "
52     + "FROM tblProduct p ON pd.MedicineID = p.MedicineID "
53     + "JOIN tblProduct p ON pd.MedicineID LIKE ? OR p.MedicineName LIKE ? OR pd.Barcode LIKE ? OR pd.Batchno LIKE ?";
54
55     try {
56         conn = DB.ConnectDB.getConnectDB();
57         ps = conn.prepareStatement(sql);
58         String searchKeyword = "%" + keyword + "%";
59         ps.setString(1, searchKeyword);
60         ps.setString(2, searchKeyword);
61         ps.setString(3, searchKeyword);
62         ps.setString(4, searchKeyword);
63         rs = ps.executeQuery();
64
65         while (rs.next()) {
66             LocalDate manufacturingDate = rs.getDate("ManufacturingDate").toLocalDate();
67             LocalDate expireDate = rs.getDate("ExpireDate").toLocalDate();
68             MedicineDetailsData data = new MedicineDetailsData(
69                 rs.getInt("ID"),
70                 rs.getString("MedicineID"),
71                 rs.getString("MedicineName"),
72                 rs.getString("Barcode"),
73                 rs.getString("Batchno"),
74                 rs.getInt("Quantity1"),
75                 rs.getString("Unit1"),
76                 rs.getInt("Quantity2"),
77                 rs.getString("Unit2"),
78                 rs.getInt("Quantity3"),
79                 rs.getString("Unit3"),
80                 manufacturingDate,
81                 expireDate,
82                 rs.getString("Status")
83             );
84         }
85         filteredList.add(data);
86     }
87 }

```

```

1 } catch (Exception e) {
2     e.printStackTrace();
3 } finally {
4     try {
5         if (rs != null) {
6             rs.close();
7         }
8         if (ps != null) {
9             ps.close();
10        }
11        if (conn != null) {
12            conn.close();
13        }
14    } catch (SQLException e) {
15        e.printStackTrace();
16    }
17 }
18 tablemedicinedetailslist.setItems(filteredList);
19 }
20
21 public void Search() {
22     searchMD.textProperty().addListener((observable, oldValue, newValue) -> {
23         filterData(newValue);
24     });
25 }
26
27 public void StatusList() {
28     List<String> ListS = new ArrayList<>();
29
30     for (String data : Data.Data.status) {
31         ListS.add(data);
32     }
33     ObservableList ListData = FXCollections.observableArrayList(ListS);
34     txtMDStatus.setItems(ListData);
35 }
36
37 private static final int itemsperpage = 21;
38
39 private int getPageCount() {
40     return (int) Math.ceil((double) dataList().size() / itemsperpage);
41 }
42
43 private void updateTable(int pageIndex) {
44     int fromIndex = pageIndex * itemsperpage;
45     int toIndex = Math.min(fromIndex + itemsperpage, dataList().size());
46     tablemedicinedetailslist.setItems(FXCollections.observableArrayList(dataList().subList(fromIndex, toIndex)));
47 }
48
49 private Node createPage(int pageIndex) {
50     updateTable(pageIndex);
51     return tablemedicinedetailslist;
52 }
53
54 public void Pagination() {
55     pgtMD.setPageCount(getPageCount());
56     pgtMD.setPageFactory(this::createPage);
57 }
58
59 @FXML
60 void UpdateMD(ActionEvent event) {
61     // Kiểm tra nếu người dùng đã chọn một hàng từ bảng
62     MedicineDetailsData selectedData = tablemedicinedetailslist.getSelectionModel().getSelectedItem();
63     if (selectedData == null) {
64         alert.errorMessage("Please select a medicine detail to update.");
65         return;
66     }
67     if (txtMDMedicineName.getText().isEmpty()
68         || txtMDBarcode.getText().isEmpty()
69         || txtMDBatchno.getText().isEmpty()
70         || txtMDManDate.getValue() == null
71         || txtMDExpireDate.getValue() == null
72         || txtQuan1.getText().isEmpty()
73         || txtUnit1.getText().isEmpty()
74         || txtMDStatus.getValue() == null) {
75         alert.errorMessage("Please fill all blank fields");
76         return;
77     } else if (!containsOnlyLetters(txtQuan1.getText())) {
78         alert.errorMessage("Please enter Number for Quantity only");
79         return;
80     } else if (Integer.parseInt(txtQuan1.getText()) <= 0) {
81         alert.errorMessage("Please enter valid positive number");
82         return;
83     } else if (!containsOnlyLetters(txtUnit1.getText())) {
84         alert.errorMessage("Please enter Letters for Unit only");
85         return;
86     }
87     // Lấy dữ liệu từ các trường nhập
88     int id = selectedData.getID();
89     String barcode = txtMDBarcode.getText();
90     String batchNo = txtMDBatchno.getText();
91     LocalDate manufacturingDate = txtMDManDate.getValue();
92     LocalDate expireDate = txtMDExpireDate.getValue();
93     String medicineName = txtMDMedicineName.getText();
94

```



```

1 if (manufacturingDate.isAfter(expireDate)) {
2     alert.errorMessage("Expiration date must be after manufacturing date.");
3     return;
4 }
5
6 int quantity1 = Integer.parseInt(txtQuan1.getText());
7 String unit1 = txtUnit1.getText();
8 int quantity2 = 0;
9 String unit2 = "";
10 int quantity3 = 0;
11 String unit3 = "";
12 String status = txtStatus.getValue();
13 if (!txtUnit2.isEditable()) {
14     if (!txtUnit2.getText().isBlank())
15         if (txtQuan1.getText().isBlank() || txtQuan2.getText().isBlank())
16             alert.errorMessage("Please fill all blank 1 fields");
17         return;
18     } else if (containsOnlyLetters(txtQuan2.getText())) {
19         alert.errorMessage("Please enter Number for Quantity only");
20         return;
21     } else if (Integer.parseInt(txtQuan2.getText()) <= 0) {
22         alert.errorMessage("Please enter valid positive number");
23         return;
24     } else if (!containsOnlyLetters(txtUnit2.getText())) {
25         alert.errorMessage("Please enter Letters For Unit only");
26         return;
27     }
28     quantity2 = Integer.parseInt(txtQuan2.getText());
29     unit2 = txtUnit2.getText();
30     if (!txtUnit3.isEditable()) {
31         if (txtQuan3.getText().isBlank() || txtUnit3.getText().isBlank())
32             alert.errorMessage("Please fill all blank Fields");
33         return;
34     } else if (containsOnlyLetters(txtQuan3.getText())) {
35         alert.errorMessage("Please enter Number for Quantity only");
36         return;
37     } else if (Integer.parseInt(txtQuan3.getText()) <= 0) {
38         alert.errorMessage("Please enter valid positive number");
39         return;
40     } else if (!containsOnlyLetters(txtUnit3.getText())) {
41         alert.errorMessage("Please enter Letters for Unit only");
42         return;
43     }
44     quantity3 = Integer.parseInt(txtQuan3.getText());
45     unit3 = txtUnit3.getText();
46 }
47 }
48 if (!isBarcodeUniqueForId(barcode, id)) {
49     alert.errorMessage("Barcode already exists. Please use a different Barcode.");
50     return;
51 }
52 }
53 // Cập nhật dữ liệu vào cơ sở dữ liệu
54 String sql = "UPDATE tblProductDetails SET BarCode = ?, Batchno = ?, Quantity1 = ?, Unit1 = ?, Quantity2 = ?, Unit2 = ?, Quantity3 = ?, Unit3 = ?, Manufacturingdate = ?, Expiredate = ?, Status = ? WHERE ID = ?";
55 try {
56     conn = DB.ConnectDB.getConnectDB();
57     ps = conn.prepareStatement(sql);
58     ps.setString(1, barcode);
59     ps.setString(2, batchNo);
60     ps.setInt(3, quantity1);
61     ps.setString(4, unit1);
62     ps.setInt(5, quantity2);
63     ps.setString(6, unit2);
64     ps.setInt(7, quantity3);
65     ps.setString(8, unit3);
66     ps.setDate(9, Date.valueOf(manufacturingDate));
67     ps.setDate(10, Date.valueOf(expireDate));
68     ps.setString(11, status);
69     ps.setInt(12, id);
70
71     int result = ps.executeUpdate();
72     if (result > 0) {
73         alert.successMessage("Medicine details updated successfully.");
74         showData(); // Refresh table view
75         Pagination(); // Refresh Pagination
76         clearFields(); // Optional: Clear input fields after successful update
77     } else {
78         alert.errorMessage("Failed to update medicine details.");
79     }
80 } catch (SQLException e) {
81     System.out.println("Error updating medicine details: " + e.getMessage());
82     alert.errorMessage("Error updating medicine details. Please try again.");
83 } finally {
84     try {
85         if (ps != null)
86             ps.close();
87     }
88     if (conn != null)
89         conn.close();
90 }
91 } catch (SQLException e) {
92     System.out.println("Error closing resources: " + e.getMessage());
93 }
94 }
95 }

```

```

1  private boolean isBarcodeUniqueForId(String barcode, int id) {
2      String sql = "SELECT COUNT(*) AS count FROM tblProductDetails WHERE BarCode = ? AND ID <> ?";
3      try {
4          conn = DB.ConnectDB.getConnectDB();
5          ps = conn.prepareStatement(sql);
6          ps.setString(1, barcode);
7          ps.setInt(2, id);
8          rs = ps.executeQuery();
9          if (rs.next()) {
10             int count = rs.getInt("count");
11             return count == 0; // True if no matching records found (Barcode is unique for given ID)
12         }
13     } catch (SQLException e) {
14         System.out.println("Error checking Barcode uniqueness for ID: " + e.getMessage());
15     } finally {
16         try {
17             if (rs != null) {
18                 rs.close();
19             }
20             if (ps != null) {
21                 ps.close();
22             }
23             if (conn != null) {
24                 conn.close();
25             }
26         } catch (SQLException e) {
27             System.out.println("Error closing resources: " + e.getMessage());
28         }
29     }
30   }
31   return false; // Default to false if an error occurs
32 }
33
34 @XML
35 private void generateAndSaveBarcode() throws SQLException {
36     // Generate a unique numeric barcode
37     String barcode = generateBarcode();
38
39     // Check if barcode generation failed
40     if (barcode == null) {
41         alert.errorMessage("Failed to generate a unique numeric barcode.");
42         return;
43     }
44
45     // Save generated barcode image to file
46     try {
47         Code128Bean barcodeGenerator = new Code128Bean();
48         final int dpi = 96;
49         barcodeGenerator.setModuleWidth(0.6);
50         barcodeGenerator.doQuietZone(true);
51
52         BitmapCanvasProvider canvas = new BitmapCanvasProvider(dpi, BufferedImage.TYPE_BYTE_BINARY, false, 0);
53         barcodeGenerator.generateBarcode(canvas, barcode);
54
55         FileChooser fileChooser = new FileChooser();
56         FileChooser.ExtensionFilter extFilter = new FileChooser.ExtensionFilter("PNG files (*.png)", "*.png");
57         fileChooser.getExtensionFilters().add(extFilter);
58         fileChooser.showSaveDialog(null);
59
60         if (barcodefile != null) {
61             FileOutputStream fos = new FileOutputStream(barcodefile);
62             canvas.finish();
63             ImageIO.write(canvas.getBufferedImage(), "png", fos);
64             fos.close();
65
66             // Save barcode details to the database
67             saveBarcodeDetails(barcode);
68
69             alert.successMessage("Barcode Saved Successfully.");
70             Desktop.getDesktop().open(barcodefile); // Open the saved barcode image
71         } else {
72             alert.errorMessage("File save cancelled.");
73         }
74     } catch (IOException e) {
75         e.printStackTrace();
76         alert.errorMessage("Error generating or saving barcode: " + e.getMessage());
77     }
78 }
79
80 private String generateBarcode() throws SQLException {
81     String barcode;
82     do {
83         barcode = generateNumericBarcode();
84     } while (!BarcodeUnique(barcode));
85
86     return barcode;
87 }
88
89 private String generateNumericBarcode() {
90     // Generate a random numeric barcode (10 digits)
91     Random random = new Random();
92     int numDigits = 12;
93     StringBuilder sb = new StringBuilder();
94     for (int i = 0; i < numDigits; i++) {
95         sb.append(random.nextInt(10)); // Generate random digit (0-9)
96     }
97     return sb.toString();
98 }
99
100 private boolean BarcodeUnique(String barcode) throws SQLException {
101     // Check if barcode already exists in the database
102     String sql = "SELECT * FROM tblBarcode WHERE BarCode = ?";
103     conn = DB.ConnectDB.getConnectDB();
104     ps = conn.prepareStatement(sql);
105     ps.setString(1, barcode);
106     ResultSet rs = ps.executeQuery();
107     boolean unique = !rs.next(); // true if ResultSet is empty (barcode is unique)
108     rs.close();
109     ps.close();
110     conn.close();
111
112     return unique;
113 }
114
115 private void saveBarcodeDetails(String barcode) throws SQLException {
116     // Save barcode details to the database
117     String sql = "INSERT INTO tblBarcode (Barcode) VALUES (?)";
118     conn = DB.ConnectDB.getConnectDB();
119     ps = conn.prepareStatement(sql);
120     ps.setString(1, barcode);
121     ps.executeUpdate();
122     ps.close();
123     conn.close();
124 }
125
126 @Override
127 public void initialize(URL location, ResourceBundle resources) {
128     showData();
129     setupAutoSuggestion();
130     Statuslist();
131     MedicineDetailsSelect();
132     Search();
133     Pagination();
134 }
135 }
136

```

14 Drug List

```

1  public class DruglistController implements Initializable {
2
3      @FXML
4      private Pagination pgtdL;
5
6      @FXML
7      private TableColumn<DruglistData, Void> DL_Col_AC;
8
9      @FXML
10     private TableColumn<DruglistData, String> DL_Col_API;
11
12     @FXML
13     private TableColumn<DruglistData, Integer> DL_Col_CP;
14
15     @FXML
16     private TableColumn<DruglistData, Integer> DL_Col_IPSU;
17
18     @FXML
19     private TableColumn<DruglistData, String> DL_Col_Item;
20
21     @FXML
22     private TableColumn<DruglistData, String> DL_Col_MedicineID;
23
24     @FXML
25     private TableColumn<DruglistData, String> DL_Col_MedicineName;
26
27     @FXML
28     private TableColumn<DruglistData, Integer> DL_Col_Quantity;
29
30     @FXML
31     private TableColumn<DruglistData, Integer> DL_Col_SP;
32
33     @FXML
34     private TableColumn<DruglistData, String> DL_Col_SU;
35
36     @FXML
37     private TableColumn<DruglistData, Integer> DL_Col_SUPU;
38
39     @FXML
40     private TableColumn<DruglistData, String> DL_Col_Unit;
41
42     @FXML
43     private Button btnAdd;
44
45     @FXML
46     private Button btnExportDL;
47
48     @FXML
49     private TextField searchDL;
50
51     @FXML
52     private TableView<DruglistData> tabeldruglist;
53
54     private Connection conn;
55     private PreparedStatement ps;
56     private ResultSet rs;
57     Message alert = new Message();
58
59     public ObservableList<DruglistData> Datalist() {
60         ObservableList<DruglistData> Datalist = FXCollections.observableArrayList();
61
62         try {
63             conn = DB.ConnectDB.getConnectDB();
64             ps = conn.prepareStatement(sql);
65             rs = ps.executeQuery();
66             DruglistData data;
67             while (rs.next()) {
68                 data = new DruglistData(rs.getString("MedicineID"),
69                                         rs.getString("MedicineName"),
70                                         rs.getDouble("CostPrice"),
71                                         rs.getInt("Quantity"),
72                                         rs.getString("Unit"),
73                                         rs.getString("Subunitsperunit"),
74                                         rs.getString("Subunit"),
75                                         rs.getString("Itemspersubunit"),
76                                         rs.getString("Item"),
77                                         rs.getInt("CostPrice"),
78                                         rs.getDouble("SellingPrice"));
79
80                 Datalist.add(data);
81             }
82         } catch (Exception e) {
83             System.out.println(e.getMessage());
84         } finally {
85             try {
86                 if (rs != null) {
87                     rs.close();
88                 }
89                 if (ps != null) {
90                     ps.close();
91                 }
92                 if (conn != null) {
93                     conn.close();
94                 }
95             } catch (SQLException e) {
96                 System.out.println("Error closing resources: " + e.getMessage());
97             }
98         }
99     }
100
101     return Datalist;
102 }
103
104     public void showData() {
105         ObservableList<DruglistData> showList = Datalist();
106         DL_Col_MedicineID.setCellValueFactory(new PropertyValueFactory<>("MedicineID"));
107         DL_Col_MedicineName.setCellValueFactory(new PropertyValueFactory<>("MedicineName"));
108         DL_Col_API.setCellValueFactory(new PropertyValueFactory<>("API"));
109         DL_Col_CP.setCellValueFactory(new PropertyValueFactory<>("Quantity"));
110         DL_Col_Unit.setCellValueFactory(new PropertyValueFactory<>("Unit"));
111         DL_Col_IPSU.setCellValueFactory(new PropertyValueFactory<>("Subunitsperunit"));
112         DL_Col_SU.setCellValueFactory(new PropertyValueFactory<>("Subunit"));
113         DL_Col_IPSU.setCellValueFactory(new PropertyValueFactory<>("Itemspersubunit"));
114         DL_Col_Item.setCellValueFactory(new PropertyValueFactory<>("Item"));
115         DL_Col_Cost.setCellValueFactory(new PropertyValueFactory<>("CostPrice"));
116         DL_Col_Sell.setCellValueFactory(new PropertyValueFactory<>("SellingPrice"));
117         DL_Col_AC.setCellValueFactory(new Callback<TableColumn<DruglistData, Void>, TableCell<DruglistData, Void>>() {
118             @Override
119             public TableCell<DruglistData, Void> call(final TableColumn<DruglistData, Void> param) {
120                 final TableCell<DruglistData, Void> cell = new TableCell<DruglistData, Void>() {
121
122                     private final Button btnUpdate = new Button("Update");
123                     private final Button btnDelete = new Button("Delete");
124
125                     {
126                         btnUpdate.getStyleClass().add("btn_update");
127                         btnDelete.getStyleClass().add("btn_delete");
128                         btnUpdate.setOnAction((ActionEvent event) -> {
129                             DruglistData pdata = getTableView().getItems().get(getIndex());
130                             openUpdateForm(pdata);
131                         });
132                     }
133
134                 };
135             }
136         });
137     }
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
825
826
827
827
828
829
829
830
831
831
832
833
833
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
```

14.1 Code

```

1      btnDelete.setOnAction((ActionEvent event) -> {
2          DruglistData pdata = getTableView().getItems().get(getIndex());
3          if (alert.confirmMessage("Are you sure want to delete Medicine ID: " + pdata.getMedicineID() + "?")) {
4              deleteData(pdata.getMedicineID());
5              showData(); // Refresh table
6              Pagination();
7          }
8      });
9      btnUpdate.setAlignment(Pos.CENTER);
10     btnDelete.setAlignment(Pos.CENTER);
11 }
12 }
13
14 @Override
15 public void updateItem(Void item, boolean empty) {
16     super.updateItem(item, empty);
17     if (empty) {
18         setGraphic(null);
19     } else {
20         HBox hbox = new HBox(5);
21         hbox.getChildren().addAll(btnUpdate, btnDelete);
22         hbox.setAlignment(Pos.CENTER);
23         setGraphic(hbox);
24     }
25 }
26 }
27 return cell;
28 });
29 });
30
31 tabeldruglist.setItems(showList);
32 }
33
34 private void deleteData(String medicineID) {
35     String sqlDeleteProductDetails = "DELETE FROM tblProductDetails WHERE MedicineID = ?";
36     String sqlDeleteProduct = "DELETE FROM tblProduct WHERE MedicineID = ?";
37     try {
38         conn = DB.ConnectDB.getConnectDB();
39         ps = conn.prepareStatement(sqlDeleteProductDetails);
40         ps.setString(1, medicineID);
41         ps.executeUpdate();
42         ps.close();
43
44         ps = conn.prepareStatement(sqlDeleteProduct);
45         ps.setString(1, medicineID);
46         ps.executeUpdate();
47         alert.successMessage("Deleted Successfully");
48     } catch (SQLException e) {
49         e.printStackTrace();
50     } finally {
51         try {
52             if (ps != null) {
53                 ps.close();
54             }
55             if (conn != null) {
56                 conn.close();
57             }
58         } catch (SQLException e) {
59             e.printStackTrace();
60         }
61     }
62 }
63
64
65 private void openUpdateForm(DruglistData pdata) {
66     try {
67         Data.dl_MedicineID = pdata.getMedicineID();
68         Data.dl_MedicineName = pdata.getMedicineName();
69         Data.dl_API = pdata.getAPI();
70         Data.dl_Costprice = pdata.getCostPrice();
71         Data.dl_Sellingprice = pdata.getSellingPrice();
72         Data.dl_Quantity = pdata.getQuantity();
73         Data.dl_Subunitsperunit = pdata.getSubunitsperunit();
74         Data.dl_Itemspersubunit = pdata.getItemspersubunit();
75         Data.dl_Unit = pdata.getUnit();
76         Data.dl_Subunit = pdata.getSubunit();
77         Data.dl_Item = pdata.getItem();
78         FXMLLoader loader = new FXMLLoader(getClass().getResource("/UpdateDrug/UpdateDrug.fxml"));
79
80         Parent root = loader.load();
81
82         Stage stage = new Stage();
83         stage.initStyle(StageStyle.UNDECORATED);
84         stage.setScene(new Scene(root));
85         stage.show();
86     } catch (Exception e) {
87         e.printStackTrace();
88     }
89 }
90
91 @FXML
92 void showformAdd(MouseEvent event) {
93     try {
94
95         FXMLLoader loader = new FXMLLoader(getClass().getResource("/AddDrug/AddDrug.fxml"));
96         Parent root = loader.load();
97         AddDrugController addDrugController = ControllerHolder.getInstance().getAddDrugController();
98         addDrugController.CheckBoxHandlers();
99
100        Stage stage = new Stage();
101        stage.initStyle(StageStyle.UNDECORATED);
102        stage.setScene(new Scene(root));
103        stage.show();
104    } catch (Exception e) {
105        e.printStackTrace();
106    }
107 }
108 }

```

```
 1 private void filterData(String keyword) {
 2     ObservableList<DruglistData> filteredList = FXCollections.observableArrayList();
 3     String sql = "SELECT * FROM tblProduct WHERE MedicineID LIKE ? OR MedicineName LIKE ? OR API LIKE ?";
 4
 5     try {
 6         conn = DB.ConnectDB.getConnectDB();
 7         ps = conn.prepareStatement(sql);
 8         String searchKeyword = "%" + keyword + "%";
 9         ps.setString(1, searchKeyword);
10         ps.setString(2, searchKeyword);
11         ps.setString(3, searchKeyword);
12         rs = ps.executeQuery();
13
14         while (rs.next()) {
15             DruglistData data = new DruglistData(
16                 rs.getString("MedicineID"),
17                 rs.getString("MedicineName"),
18                 rs.getString("API"),
19                 rs.getInt("Quantity"),
20                 rs.getString("Unit"),
21                 rs.getInt("Subunitsperunit"),
22                 rs.getString("Subunit"),
23                 rs.getInt("Itemspersubunit"),
24                 rs.getString("Item"),
25                 rs.getInt("CostPrice"),
26                 rs.getInt("SellingPrice")
27             );
28             filteredList.add(data);
29         }
30     } catch (Exception e) {
31         e.printStackTrace();
32     } finally {
33         try {
34             if (rs != null) {
35                 rs.close();
36             }
37             if (ps != null) {
38                 ps.close();
39             }
40             if (conn != null) {
41                 conn.close();
42             }
43         } catch (SQLException e) {
44             e.printStackTrace();
45         }
46     }
47
48     tabledruglist.setItems(filteredList);
49 }
50
51 public void Search() {
52     searchDL.textProperty().addListener((observable, oldValue, newValue) -> {
53         filterData(newValue);
54     });
55 }
56 private static final int itemsperpage = 23;
57
58 private int getPageCount() {
59     return (int) Math.ceil((double) dataList().size() / itemsperpage);
60 }
61
62 private void updateTable(int pageIndex) {
63     int fromIndex = pageIndex * itemsperpage;
64     int toIndex = Math.min(fromIndex + itemsperpage, dataList().size());
65     tabledruglist.setItems(FXCollections.observableArrayList(dataList().subList(fromIndex, toIndex)));
66 }
67
68 private Node createPage(int pageIndex) {
69     updateTable(pageIndex);
70     return tabledruglist;
71 }
```

```
 1  public void Pagination() {
 2      pgtDL.setPageCount(getPageCount());
 3      pgtDL.setPageFactory(this::createPage);
 4  }
 5
 6  public void exportToPDF() {
 7      FileChooser fileChooser = new FileChooser();
 8      fileChooser.setTitle("Save PDF");
 9      String fontPath = "src/font/arial.ttf";
10      fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("PDF Files", "*.pdf"));
11      File selectedFile = fileChooser.showSaveDialog(btnExportDL.getScene().getWindow());
12
13      if (selectedFile != null) {
14          try {
15
16              PdfFont font = PdfFontFactory.createFont(fontPath, PdfEncodings.IDENTITY_H, true);
17
18              PdfWriter writer = new PdfWriter(selectedFile.getAbsolutePath());
19              PdfDocument pdf = new PdfDocument(writer);
20              Document document = new Document(pdf);
21
22              Paragraph title = new Paragraph("Drug List").setBold().setFontSize(18);
23              document.add(title);
24
25              float[] columnWidths = {100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100};
26              Table table = new Table(columnWidths);
27
28              TableView<DruglistData> tableView = tableDruglist;
29              for (TableColumn<DruglistData, ?> column : tableView.getColumns()) {
30                  if (!column.getText().equals("Action")) {
31                      table.addCell(new Cell().add(new Paragraph(column.getText()).setFont(font)));
32                  }
33              }
34
35              ObservableList<DruglistData> dataList = tableView.getItems();
36              for (DruglistData data : dataList) {
37                  table.addCell(new Cell().add(new Paragraph(data.getMedicineID())));
38                  table.addCell(new Cell().add(new Paragraph(data.getMedicineName())));
39                  table.addCell(new Cell().add(new Paragraph(data.getAPI())));
40                  table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getQuantity()))));
41                  table.addCell(new Cell().add(new Paragraph(data.getUnit())));
42                  table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getSubunitsperunit()))));
43                  table.addCell(new Cell().add(new Paragraph(data.getSubunit())));
44                  table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getItemspersubunit()))));
45                  table.addCell(new Cell().add(new Paragraph(data.getItem())));
46                  table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getCostPrice()))));
47                  table.addCell(new Cell().add(new Paragraph(String.valueOf(data.getSellingPrice()))));
48              }
49              document.add(table);
50              document.close();
51              Desktop.getDesktop().open(selectedFile);
52
53              alert.successMessage("Export successful!");
54          } catch (IOException e) {
55              alert.errorMessage("Export failed: " + e.getMessage());
56          }
57      }
58  }
59
60  @Override
61  public void initialize(URL url, ResourceBundle rb) {
62      ControllerHolder.getInstance().setDruglistController(this);
63      showData();
64      Search();
65      Pagination();
66  }
67
68 }
```

15 Payroll

```
1  public class FXMLBangluongController implements Initializable {
2
3      @FXML
4      private FXMLMainController mainController;
5      @FXML
6      private Button btnExport;
7
8      @FXML
9      private Button btnthempayroll;
10
11     @FXML
12     private TableColumn<TableBangluong, Integer> cotgiotieuchuan;
13
14     @FXML
15     private TableColumn<TableBangluong, String> cotstaffid;
16
17     @FXML
18     private TableColumn<TableBangluong, Date> cotngaybatdau;
19
20     @FXML
21     private TableColumn<TableBangluong, Date> cotngayketthuc;
22
23     @FXML
24     private TableColumn<TableBangluong, Integer> cotngaytieuchuan;
25
26     @FXML
27     private TableColumn<TableBangluong, BigDecimal> cotsongaylam;
28
29     @FXML
30     private TableColumn<TableBangluong, BigDecimal> cottangca;
31
32     @FXML
33     private TableColumn<TableBangluong, BigDecimal> cottraluong;
34
35     @FXML
36     private TableColumn<TableBangluong, Integer> cotID;
37
38     @FXML
39     private TableColumn<TableBangluong, BigDecimal> cotSalary;
40
41     @FXML
42     private TableView<TableBangluong> tablepayroll;
43
44     @FXML
45     private TextField txttukhoa;
46
47     @FXML
48     private Button btnStaff;
49
50     @FXML
51     private Button btnTimekeeping;
52
53     @FXML
54     private TableColumn<TableBangluong, Void> cotaction; // Để sử dụng Void vì không cần dữ liệu, chỉ cần hiển thị nút
55
56     public ConnectDB connect = new ConnectDB();
57     private Connection conn;
58     private PreparedStatement pst;
59     private ResultSet rs;
60
61     public ObservableList<TableBangluong> dataList() {
62         ObservableList<TableBangluong> dataList = FXCollections.observableArrayList();
63         String sql = "SELECT * FROM tblPayroll";
64
65         try {
66             conn = connect.getConnectDB();
67             pst = conn.prepareStatement(sql);
68             rs = pst.executeQuery();
69
70             while (rs.next()) {
71                 TableBangluong data = new TableBangluong(
72                     rs.getInt("ID"),
73                     rs.getString("StaffID"),
74                     rs.getDate("PayPeriodStart"),
75                     rs.getDate("PayPeriodEnd"),
76                     rs.getInt("StandardDays"),
77                     rs.getInt("StandardHours"),
78                     rs.getBigDecimal("TotalWorkDays"),
79                     rs.getBigDecimal("Overtime"),
80                     rs.getBigDecimal("TotalPay"),
81                     rs.getBigDecimal("Salary")
82                 );
83                 dataList.add(data);
84             }
85         } catch (Exception e) {
86             System.out.println("Error while fetching data: " + e.getMessage());
87         }
88
89     return dataList;
90 }
```

15.1 Code

```

1  public void showDataBangluong() {
2      ObservableList<TableBangluong> showList = dataList();
3      cotID.setCellValueFactory(new PropertyValueFactory<>("ID"));
4      cotstaffid.setCellValueFactory(new PropertyValueFactory<>("staffID"));
5      cotngaybatdau.setCellValueFactory(new PropertyValueFactory<>("payPeriodStart"));
6      cotngayketthuc.setCellValueFactory(new PropertyValueFactory<>("payPeriodEnd"));
7      cotgiolieucon.setCellValueFactory(new PropertyValueFactory<>("StandardHours"));
8      cotsongaylam.setCellValueFactory(new PropertyValueFactory<>("totalWorkdays"));
9      cottangca.setCellValueFactory(new PropertyValueFactory<>("overtime"));
10     cottraLuong.setCellValueFactory(new PropertyValueFactory<>("totalPay"));
11     cotSalary.setCellValueFactory(new PropertyValueFactory<>("Salary"));
12
13     cotID.setCellFactory(column -> {
14         return new TableCell<TableBangluong, Integer>() {
15             @Override
16             protected void updateItem(Integer item, boolean empty) {
17                 super.updateItem(item, empty);
18                 if (empty) {
19                     setText(null);
20                 } else {
21                     setText(String.valueOf(getIndex() + 1));
22                 }
23             }
24         };
25     });
26 });
27
28     tablepayroll.setItems(showList);
29 }
30
31 //search
32 public void searchStaff() {
33     String keyword = txttukhoa.getText().toLowerCase().trim();
34
35     ObservableList<TableBangluong> filteredList = FXCollections.observableArrayList();
36
37     for (TableBangluong staff : dataList()) {
38         if (staff.getStaffID().toLowerCase().contains(keyword)) {
39             filteredList.add(staff);
40         }
41     }
42
43     tablepayroll.setItems(filteredList);
44 }
45
46 //chức năng xuất dữ liệu ra excel bằng nút Export
47 public void exportToCSV() {
48     FileChooser fileChooser = new FileChooser();
49     fileChooser.setTitle("Save CSV File");
50     fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "*.csv"));
51     File selectedFile = fileChooser.showSaveDialog(btnExport.getScene().getWindow());
52
53     if (selectedFile != null) {
54         try {
55             Writer writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(selectedFile), StandardCharsets.UTF_8));
56             writer.write("\uFEFF"); // BOM for UTF-8
57
58             // Ghi tiêu đề cột
59             TableView<TableBangluong> tableView = tablepayroll;
60             ObservableList<TableColumn<TableBangluong, ?>> columns = tableView.getColumns();
61             for (int i = 0; i < columns.size(); i++) {
62                 TableColumn<TableBangluong, ?> column = columns.get(i);
63                 if (i > 0) {
64                     writer.write(",");
65                 }
66                 writer.write(column.getText());
67             }
68             writer.write("\n");
69
70             // Ghi dữ liệu từng dòng
71             ObservableList<TableBangluong> dataList = tableView.getItems();
72             for (TableBangluong staff : dataList) {
73                 for (int i = 0; i < columns.size(); i++) {
74                     TableColumn<TableBangluong, ?> column = columns.get(i);
75                     if (i > 0) {
76                         writer.write(",");
77                     }
78                     // Lấy giá trị từ cell của bảng và ghi vào writer
79                     Object cellValue = column.getCellData(staff);
80                     writer.write(cellValue != null ? cellValue.toString() : "");
81                 }
82                 writer.write("\n");
83             }
84             writer.flush();
85             writer.close();
86
87             Thuvien tv = new Thuvien(); // Thêm dòng này nếu Thuvien cần được sử dụng
88             tv.showAlert("Data exported to CSV successfully!");
89
90             // Mở file CSV sau khi export
91             Desktop.getDesktop().open(selectedFile);
92
93         } catch (IOException e) {
94             System.out.println("Error while exporting to CSV: " + e.getMessage());
95         }
96     }
97 }
98
99 //action
100 public void setupActionColumn() {
101     // Thiết lập cell factory cho cột hành động.
102     cotaction.setCellFactory(new Callback<TableColumn<TableBangluong, Void>, TableCell<TableBangluong, Void>>() {
103         @Override
104         public TableCell<TableBangluong, Void> call(TableColumn<TableBangluong, Void> param) {
105             // Tạo một NewTableCell mới để chứa các nút hành động.
106             return new NewTableCell<TableBangluong, Void>() {
107                 // Khai báo và khởi tạo các nút Update và Delete.
108                 private final Button btnUpdate = new Button("Update");
109                 private final Button btnDelete = new Button("Delete");
110
111                 {
112                     // Đặt style cho các nút.
113                     btnUpdate.getStyleClass().add("button-update");
114                     btnDelete.getStyleClass().add("button-delete");
115
116                     // Thiết lập sự kiện khi nhấn nút Update.
117                     btnUpdate.setOnAction(event -> {
118                         // Lấy dữ liệu của dòng hiện tại.
119                         TableBangluong data = getTableView().getItems().get(getIndex());
120                         try {

```

```
1 public void showDataBangluong() {
2     ObservableList<TableBangluong> showList = dataList();
3     cotID.setCellValueFactory(new PropertyValueFactory<>("ID"));
4     cotstaffID.setCellValueFactory(new PropertyValueFactory<>("staffID"));
5     cotngayketthuc.setCellValueFactory(new PropertyValueFactory<>("payPeriodStart"));
6     cotngayketthuc.setCellValueFactory(new PropertyValueFactory<>("payPeriodEnd"));
7     cotngaytieuchuan.setCellValueFactory(new PropertyValueFactory<>("standardDays"));
8     cotgiotieuchuan.setCellValueFactory(new PropertyValueFactory<>("totalWorkdays"));
9     cotsongaylam.setCellValueFactory(new PropertyValueFactory<>("totalWorkdays"));
10    cottangca.setCellValueFactory(new PropertyValueFactory<>("overtime"));
11    cottraluong.setCellValueFactory(new PropertyValueFactory<>("totalPay"));
12    cotSalary.setCellValueFactory(new PropertyValueFactory<>("Salary"));
13
14    cotID.setCellFactory(column -> {
15        return new TableCell<TableBangluong, Integer>() {
16            @Override
17            protected void updateItem(Integer item, boolean empty) {
18                super.updateItem(item, empty);
19                if (empty) {
20                    setText(null);
21                } else {
22                    setText(String.valueOf(getIndex() + 1));
23                }
24            }
25        };
26    });
27
28    tablepayroll.setItems(showList);
29 }
30
31 //search
32 public void searchStaff() {
33     String keyword = txttukhoa.getText().toLowerCase().trim();
34
35     ObservableList<TableBangluong> filteredList = FXCollections.observableArrayList();
36
37     for (TableBangluong staff : dataList()) {
38         if (staff.getStaffID().toLowerCase().contains(keyword)) {
39             filteredList.add(staff);
40         }
41     }
42
43     tablepayroll.setItems(filteredList);
44 }
45
46 //chức năng xuất dữ liệu ra excel bằng nút Export
47 public void exportToCSV() {
48     FileChooser fileChooser = new FileChooser();
49     fileChooser.setTitle("Save CSV File");
50     fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "*.csv"));
51     File selectedfile = fileChooser.showSaveDialog(btnExport.getScene().getWindow());
52
53     if (selectedfile != null) {
54         try {
55             Writer writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(selectedfile), StandardCharsets.UTF_8));
56             writer.write("\uFEFF"); // BOM for UTF-8
57
58             // Ghi tiêu đề cột
59             TableView<TableBangluong> tableView = tablepayroll;
60             ObservableList<TableColumn<TableBangluong, ?>> columns = tableView.getColumns();
61             for (int i = 0; i < columns.size(); i++) {
62                 TableColumn<TableBangluong, ?> column = columns.get(i);
63                 if (i > 0) {
64                     writer.write(",");
65                 }
66                 writer.write(column.getText());
67             }
68             writer.write("\n");
69
70             // Ghi dữ liệu từng dòng
71             ObservableList<TableBangluong> dataList = tableView.getItems();
72             for (TableBangluong staff : dataList) {
73                 for (int i = 0; i < staff.getColumns().size(); i++) {
74                     TableColumn<TableBangluong, ?> column = staff.getColumns().get(i);
75                     if (i > 0) {
76                         writer.write(",");
77                     }
78                     // Lấy giá trị từ cell của bảng và ghi vào writer
79                     Object cellValue = column.getCellData(staff);
80                     writer.write(cellValue != null ? cellValue.toString() : "");
81                 }
82                 writer.write("\n");
83             }
84             writer.flush();
85             writer.close();
86
87             Thuvien tv = new Thuvien(); // Thêm dòng này nếu Thuvien cần được sử dụng
88             tv.showAlert("Data exported to CSV successfully!");
89
90             // Mở file CSV sau khi export
91             Desktop.getDesktop().open(selectedfile);
92
93         } catch (IOException e) {
94             System.out.println("Error while exporting to CSV: " + e.getMessage());
95         }
96     }
97 }
98
99 //action
100 public void setupActionColumn() {
101     // Thiết lập cell factory cho cột hành động.
102     cotaction.setCellFactory(new Callback<TableColumn<TableBangluong, Void>, TableCell<TableBangluong, Void>>() {
103         @Override
104         public TableCell<TableBangluong, Void> call(TableColumn<TableBangluong, Void> param) {
105             // Tạo một TableCell mới để chứa các nút hành động.
106             return new TableCell<TableBangluong, Void>() {
107                 // Khai báo và khởi tạo các nút Update và Delete.
108                 private final Button btnUpdate = new Button("Update");
109                 private final Button btnDelete = new Button("Delete");
110
111                 {
112                     // Đặt style cho các nút.
113                     btnUpdate.getStyleClass().add("button-update");
114                     btnDelete.getStyleClass().add("button-delete");
115
116                     // Thiết lập sự kiện khi nhấn nút Update.
117                     btnUpdate.setOnAction(event -> {
118                         // Lấy dữ liệu của dòng hiện tại.
119                         TableBangluong data = getTableView().getItems().get(getIndex());
120                         try {

```

```

1 // Tải FXML của form cập nhật.
2         FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLUpdatePayroll.fxml"));
3         Parent root = loader.load();
4
5         // Lấy controller của form cập nhật và thiết lập dữ liệu cần cập nhật.
6         FXMLUpdatePayrollController controller = loader.getController();
7         controller.setUpdatePayroll(data);
8         controller.setParentController(FMLLBangluongController.this);
9
10        // Tạo một stage mới và thiết lập các thuộc tính.
11        Scene scene = new Scene(root);
12        Stage newStage = new Stage();
13        newStage.setScene(scene);
14        newStage.initModality(Modality.APPLICATION_MODAL);
15        newStage.initStyle(StageStyle.UNDECORATED);
16
17        // Lấy vị trí của cửa sổ hiện tại để đặt cửa sổ mới ở giữa màn hình.
18        Stage currentStage = (Stage) btnUpdate.getScene().getWindow();
19        double centerXPosition = currentStage.getX() + currentStage.getWidth() / 2d;
20        double centerYPosition = currentStage.getY() + currentStage.getHeight() / 2d;
21        newStage.setX(centerXPosition - root.getWidth() / 2d);
22        newStage.setY(centerYPosition - root.getHeight() / 2d);
23        newStage.show();
24
25        } catch (IOException e) {
26            e.printStackTrace();
27        }
28    });
29
30    // Thiết lập sự kiện khi nhấn nút Delete.
31    btnDelete.setOnAction(event -> {
32        // Lấy dữ liệu của dòng hiện tại.
33        TableView<TableBangluong> data = getTableView().getItems().get(getIndex());
34        // Hiển thị hộp thoại xác nhận xóa.
35        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
36        alert.setTitle("Confirm Delete");
37        alert.setHeaderText("Are you sure you want to delete this record?");
38        alert.setContentText("This action cannot be undone.");
39
40        // Xử lý khi người dùng xác nhận xóa.
41        alert.showAndWait().ifPresent(response -> {
42            if (response == ButtonType.OK) {
43                try {
44                    // Kết nối cơ sở dữ liệu và thực hiện câu lệnh xóa.
45                    conn = connect.getConnection();
46                    String sql = "DELETE FROM tblPayroll WHERE StaffID = ?";
47                    pst = conn.prepareStatement(sql);
48                    pst.setString(1, data.getStaffID());
49                    pst.executeUpdate();
50
51                    // Xóa dòng dữ liệu khỏi bảng.
52                    getTableView().getItems().remove(data);
53                    System.out.println("Delete clicked for " + data.getStaffID());
54                } catch (Exception e) {
55                    System.out.println("Error while deleting payroll: " + e.getMessage());
56                } finally {
57                    try {
58                        if (pst != null) {
59                            pst.close();
60                        }
61                        if (conn != null) {
62                            conn.close();
63                        }
64                    } catch (Exception e) {
65                        System.out.println("Error while closing connection: " + e.getMessage());
66                    }
67                }
68            });
69        });
70    });
71
72    @Override
73    public void updateItem(Void item, boolean empty) {
74        super.updateItem(item, empty);
75
76        // Nếu dòng hiện tại không có dữ liệu, ẩn các nút.
77        if (empty) {
78            setGraphic(null);
79        } else {
80            // Nếu có dữ liệu, hiển thị các nút Update và Delete.
81            HBox hbox = new HBox(btnUpdate, btnDelete);
82            hbox.setSpacing(5);
83            setGraphic(hbox);
84
85        }
86    };
87}
88);
89});
90}
91
92 public void setMainController(FMLMainController mainController) {
93     this.mainController = mainController;
94 }
95
96 //chuyen form nhan vien
97 @FXML
98 private void goToNhanvienForm() {
99     if (mainController != null) {
100         mainController.loadFXML("/demoproject1/FMLNhanVien.fxml");
101     } else {
102         System.out.println("FMLLBangluongController: MainController is null, cannot load FMLNhanVien.fxml");
103     }
104 }

```

```
1  @FXML
2  private void goToChamluongFormFromBangluong() {
3      if (mainController != null) {
4          mainController.loadFXML("/demoproject1/FXMLChamluong.fxml");
5      } else {
6          System.err.println("FXMLBangluongController: MainController is null, cannot load FXMLChamluong.fxml");
7      }
8  }
9
10 @Override
11 public void initialize(URL url, ResourceBundle rb) {
12     showDataBangluong();
13     setupActionColumn(); //them nut vao action
14     //search
15     txttukhoa.textProperty().addListener((observable, oldValue, newValue) -> {
16         searchStaff();
17     });
18     //xuat ra excel
19     btnExport.setOnAction(event -> exportToCSV());
20
21     // Bắt sự kiện khi nhấn nút Chuyển form
22     btnthempayroll.setOnAction(event -> {
23         try {
24             // Load file FXML của form mới
25             FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLThemPayroll.fxml"));
26             Parent root = loader.load();
27
28             // Tạo một Scene mới
29             Scene scene = new Scene(root);
30
31             // Tạo một Stage mới cho form mới
32             Stage newStage = new Stage();
33
34             // Đặt modality để đảm bảo form mới đè lên và không thao tác được form cũ
35             newStage.initModality(Modality.APPLICATION_MODAL);
36
37             // Đặt style để không có nút đóng góc
38             newStage.initStyle(StageStyle.UNDECORATED);
39
40             // Đặt Scene vào Stage mới
41             newStage.setScene(scene);
42
43             // Lấy Stage hiện tại từ Button
44             Stage currentStage = (Stage) btnthempayroll.getScene().getWindow();
45
46             // Tính toán vị trí để đưa form mới vào giữa màn hình
47             double centerXPosition = currentStage.getX() + currentStage.getWidth() / 2d;
48             double centerYPosition = currentStage.getY() + currentStage.getHeight() / 2d;
49
50             // Thiết lập vị trí của Stage mới ở giữa màn hình
51             newStage.setX(centerXPosition - root.getWidth() / 2d);
52             newStage.setY(centerYPosition - root.getHeight() / 2d);
53
54             // Hiển thị Stage mới và không đóng Stage cũ
55             newStage.show();
56
57         } catch (IOException e) {
58             e.printStackTrace();
59         }
60     });
61 }
62 }
63 }
64 }
```

16 TimeKeeping

```

1  public class FXMLChamluongController implements Initializable {
2
3      @FXML
4      private FXMLMainController mainController;
5      @FXML
6      private Button btnPayroll;
7
8      @FXML
9      private Button btnCheckin;
10
11     @FXML
12     private Button btnStaff;
13     @FXML
14     private Button btnExport;
15
16     @FXML
17     private TableColumn<TableChamluong, Integer> cotID;
18
19     @FXML
20     private TableColumn<TableChamluong, String> cotStaffid;
21
22     @FXML
23     private TableColumn<TableChamluong, String> cotStaffname;
24
25     @FXML
26     private TableColumn<TableChamluong, Date> cotcheckin;
27
28     @FXML
29     private TableColumn<TableChamluong, Date> cotcheckout;
30
31     @FXML
32     private TableColumn<TableChamluong, Date> cotday;
33
34     @FXML
35     private TableView<TableChamluong> tableTimekeeping;
36
37     @FXML
38     private TextField txttukhoa;
39
40     public ConnectDB connect = new ConnectDB();
41     private Connection conn;
42     private PreparedStatement pst;
43     private ResultSet rs;
44
45     /**
46      * Initializes the controller class.
47     */
48     public ObservableList<TableChamluong> dataList() {
49         ObservableList<TableChamluong> dataList = FXCollections.observableArrayList();
50         String sql = "SELECT * FROM tblTimekeeping";
51
52         try {
53             conn = connect.getConnectDB();
54             pst = conn.prepareStatement(sql);
55             rs = pst.executeQuery();
56
57             while (rs.next()) {
58                 TableChamluong data = new TableChamluong(
59                     rs.getInt("ID"),
60                     rs.getString("StaffID"),
61                     rs.getString("Staffname"),
62                     rs.getDate("Day"),
63                     rs.getTime("Checkin"),
64                     rs.getTime("Checkout")
65                 );
66                 dataList.add(data);
67             }
68         } catch (SQLException e) {
69             System.out.println("Lỗi khi lấy dữ liệu: " + e.getMessage());
70         } finally {
71             closeResources();
72         }
73
74         return dataList;
75     }
76
77     public void showDataChamluong() {
78         ObservableList<TableChamluong> showList = dataList();
79
80         cotID.setCellValueFactory(new PropertyValueFactory<>("ID"));
81         cotStaffid.setCellValueFactory(new PropertyValueFactory<>("StaffID"));
82         cotStaffname.setCellValueFactory(new PropertyValueFactory<>("Staffname"));
83         cotday.setCellValueFactory(new PropertyValueFactory<>("Day"));
84         cotcheckin.setCellValueFactory(new PropertyValueFactory<>("Checkin"));
85         cotcheckout.setCellValueFactory(new PropertyValueFactory<>("Checkout"));
86
87         cotID.setCellFactory(column -> {
88             return new TableCell<TableChamluong, Integer>() {
89                 @Override
90                 protected void updateItem(Integer item, boolean empty) {
91                     super.updateItem(item, empty);
92                     if (empty) {
93                         setText(null);
94                     } else {
95                         setText(String.valueOf(getIndex() + 1));
96                     }
97                 }
98             };
99         });
100        tableTimekeeping.setItems(showList);
101    }

```

16.1 Code



```

1  @FXML
2  public void refreshTable() {
3      showDataChamLuong();
4  }
5
6  public void searchStaff() {
7      String keyword = txttukhoa.getText().toLowerCase().trim();
8
9      if (keyword.isEmpty()) {
10          showDataChamLuong(); // Nếu ô tìm kiếm trống, hiển thị lại dữ liệu ban đầu
11          return;
12     }
13
14     ObservableList<TableChamLuong> filteredList = FXCollections.observableArrayList();
15
16     // Thực hiện truy vấn tìm kiếm
17     String sql = "SELECT * FROM tbTimekeeping WHERE StaffID LIKE ?";
18
19     try {
20         conn = connect.getConnectDB();
21         pst = conn.prepareStatement(sql);
22         pst.setString(1, "%" + keyword + "%");
23         rs = pst.executeQuery();
24
25         while (rs.next()) {
26             TableChamLuong data = new TableChamLuong(
27                 rs.getInt("ID"),
28                 rs.getString("StaffID"),
29                 rs.getString("Staffname"),
30                 rs.getDate("Day"),
31                 rs.getTime("CheckIn"),
32                 rs.getTime("Checkout")
33             );
34             filteredList.add(data);
35         }
36     } catch (SQLException e) {
37         System.out.println("Lỗi khi tìm kiếm: " + e.getMessage());
38     } Finally {
39         closeResources();
40     }
41
42     // Hiển thị kết quả tìm kiếm trong TableView
43     tableViewTimekeeping.setItems(filteredList);
44 }
45
46 public void setMainController(FXMLMainController mainController) {
47     this.mainController = mainController;
48 }
49
50
51 @FXML
52 private void goToNhanvienForm() {
53     if (mainController != null) {
54         mainController.loadFXML("/demoproject1/FXMLNhanVien.fxml");
55     } else {
56         System.err.println("FXMLChamLuongController: MainController is null, cannot load FXMLNhanVien.fxml");
57     }
58 }
59
60 @FXML
61 private void goToBangluongFormFromChamLuong() {
62     if (mainController != null) {
63         mainController.loadFXML("/demoproject1/FXMLBangluong.fxml");
64     } else {
65         System.err.println("FXMLChamLuongController: MainController is null, cannot load FXMLBangluong.fxml");
66     }
67
68     // Chức năng xuất dữ liệu ra excel bằng nút Export
69     public void exportToCSV() {
70        FileChooser fileChooser = new FileChooser();
71         fileChooser.setInitialDirectory(new File("Save CSV File"));
72         fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "*.csv"));
73         File selectedfile = fileChooser.showSaveDialog(btnExport.getScene().getWindow());
74
75         if (selectedfile != null) {
76             try {
77                 BufferedWriter writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(selectedfile), StandardCharsets.UTF_8));
78                 writer.write("\uFEFF"); // BOM for UTF-8
79
80                 // Ghi tiêu đề cột
81                 TableView<TableChamLuong> tableView = tableViewTimekeeping;
82                 ObservableList<TableColumn<TableChamLuong>> columns = tableView.getColumns();
83                 for (int i = 0; i < columns.size(); i++) {
84                     TableColumn<TableChamLuong> column = columns.get(i);
85                     if (i > 0) {
86                         writer.write(",");
87                     }
88                     writer.write(column.getText());
89                 }
90                 writer.write("\n");
91
92                 // Ghi dữ liệu từng dòng
93                 ObservableList<TableChamLuong> dataList = tableView.getItems();
94                 for (TableChamLuong staff : dataList) {
95                     for (int i = 0; i < columns.size(); i++) {
96                         TableColumn<TableChamLuong> column = columns.get(i);
97                         if (i > 0) {
98                             writer.write(",");
99                         }
100                        // Lấy giá trị từ cell của bảng và ghi vào writer
101                        Object cellValue = column.getCellData(staff);
102                        writer.write(cellValue != null ? cellValue.toString() : "");
103                    }
104                    writer.write("\n");
105                }
106                writer.flush();
107                writer.close();
108
109                Thuvien tv = new Thuvien(); // Thêm dòng này nếu Thuvien cần được sử dụng
110                tv.showAlert("Data exported to CSV successfully!");
111
112                // Mở file CSV sau khi export
113                Desktop.getDesktop().open(selectedfile);
114
115            } catch (IOException e) {
116                System.out.println("Error while exporting to CSV: " + e.getMessage());
117            }
118        }
119    }
}

```

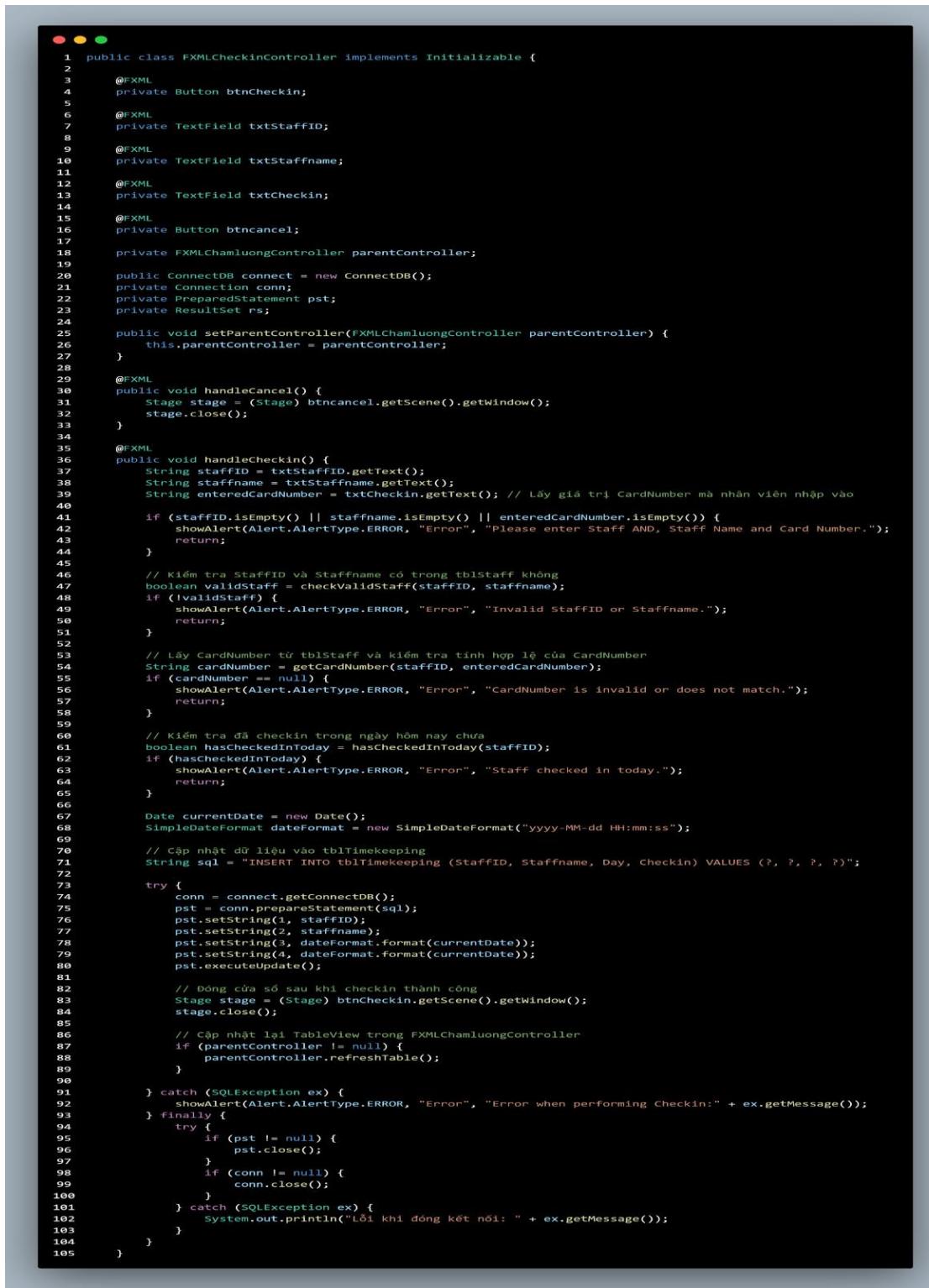
```

1  @Override
2      public void initialize(URL url, ResourceBundle rb) {
3
4          txttukhoa.textProperty().addListener((observable, oldValue, newValue) -> {
5              searchStaff();
6          });
7
8          showDataChamluong();
9
10         // Bắt sự kiện khi nhấn nút Chuyển form
11         btnCheckin.setOnAction(event -> {
12             try {
13                 FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLCheckin.fxml"));
14                 Parent root = loader.load();
15
16                 FXMLCheckinController controller = loader.getController();
17                 controller.setParentController(this); // Truyền FXMLChamluongController vào đây
18
19                 Scene scene = new Scene(root);
20                 Stage newStage = new Stage();
21                 newStage.initModality(Modality.APPLICATION_MODAL);
22                 newStage.initStyle(StageStyle.UNDECORATED);
23                 newStage.setScene(scene);
24
25                 Stage currentStage = (Stage) btnCheckin.getScene().getWindow();
26                 double centerXPosition = currentStage.getX() + currentStage.getWidth() / 2d;
27                 double centerYPosition = currentStage.getY() + currentStage.getHeight() / 2d;
28                 newStage.setX(centerXPosition - root.prefWidth(-1) / 2d);
29                 newStage.setY(centerYPosition - root.prefHeight(-1) / 2d);
30
31                 newStage.show();
32             } catch (IOException e) {
33                 e.printStackTrace();
34             }
35         });
36
37         // Bắt sự kiện khi nhấn đúp vào hàng trong bảng
38         tableTimekeeping.setOnMouseClicked(event -> {
39             if (event.getClickCount() == 2) {
40                 TableChamluong selectedRecord = tableTimekeeping.getSelectionModel().getSelectedItem();
41                 if (selectedRecord != null && selectedRecord.getCheckout() == null) {
42                     try {
43                         FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLCheckout.fxml"));
44                         Parent root = loader.load();
45
46                         FXMLCheckoutController controller = loader.getController();
47                         controller.setParentController(this); // Truyền FXMLChamluongController vào đây
48                         controller.setSelectedRecord(selectedRecord); // Truyền dữ liệu dòng được chọn vào controller của FXMLCheckout
49
50                         Scene scene = new Scene(root);
51                         Stage newstage = new Stage();
52                         newstage.initModality(Modality.APPLICATION_MODAL);
53                         newstage.initStyle(StageStyle.UNDECORATED);
54                         newStage.setScene(scene);
55
56                         Stage currentStage = (Stage) tableTimekeeping.getScene().getWindow();
57                         double centerXPosition = currentStage.getX() + currentStage.getWidth() / 2d;
58                         double centerYPosition = currentStage.getY() + currentStage.getHeight() / 2d;
59                         newStage.setX(centerXPosition - root.prefWidth(-1) / 2d);
60                         newStage.setY(centerYPosition - root.prefHeight(-1) / 2d);
61
62                         newStage.show();
63                     } catch (IOException e) {
64                         e.printStackTrace();
65                     }
66                 } else {
67                     showAlert(Alert.AlertType.ERROR, "Error", "This employee has checked out.");
68                 }
69             }
70         });
71     }
72
73     private void showAlert(Alert.AlertType type, String title, String content) {
74         Alert alert = new Alert(type);
75         alert.setTitle(title);
76         alert.setContentText(content);
77         alert.showAndWait();
78     }
79
80     private void closeResources() {
81         try {
82             if (rs != null) {
83                 rs.close();
84             }
85             if (pst != null) {
86                 pst.close();
87             }
88             if (conn != null) {
89                 conn.close();
90             }
91         } catch (SQLException e) {
92             System.out.println("Lỗi khi đóng các tài nguyên: " + e.getMessage());
93         }
94     }
95 }
96 
```

17 Checkin

```

1  private boolean hasCheckedInToday(String staffID) {
2      String sql = "SELECT * FROM tblTimekeeping WHERE StaffID = ? AND CAST(Day AS DATE) = CAST(GETDATE() AS DATE)";
3
4      try {
5          conn = connect.getConnectDB();
6          pst = conn.prepareStatement(sql);
7          pst.setString(1, staffID);
8          rs = pst.executeQuery();
9
10         // Kiểm tra số lần đã checkin trong ngày hôm nay
11         return rs.next(); // True nếu có kết quả từ câu truy vấn (đã checkin trong ngày hôm nay)
12     } catch (SQLException ex) {
13         System.out.println("Lỗi khi kiểm tra đã checkin: " + ex.getMessage());
14         return false;
15     } finally {
16         try {
17             if (rs != null) {
18                 rs.close();
19             }
20             if (pst != null) {
21                 pst.close();
22             }
23             if (conn != null) {
24                 conn.close();
25             }
26         } catch (SQLException ex) {
27             System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
28         }
29     }
30 }
31 }
32
33 private boolean checkValidStaff(String staffID, String staffname) {
34     String sql = "SELECT * FROM tblistaff WHERE StaffID = ? AND Staffname = ?";
35     try {
36         conn = connect.getConnectDB();
37         pst = conn.prepareStatement(sql);
38         pst.setString(1, staffID);
39         pst.setString(2, staffname);
40         rs = pst.executeQuery();
41         return rs.next(); // True nếu có kết quả từ câu truy vấn
42     } catch (SQLException ex) {
43         System.out.println("Lỗi khi kiểm tra nhân viên: " + ex.getMessage());
44         return false;
45     } finally {
46         try {
47             if (rs != null) {
48                 rs.close();
49             }
50             if (pst != null) {
51                 pst.close();
52             }
53             if (conn != null) {
54                 conn.close();
55             }
56         } catch (SQLException ex) {
57             System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
58         }
59     }
60 }
61 }
62
63 private String getCardNumber(String staffID, String enteredCardNumber) {
64     String sql = "SELECT CardNumber FROM tblistaff WHERE StaffID = ?";
65     try {
66         conn = connect.getConnectDB();
67         pst = conn.prepareStatement(sql);
68         pst.setString(1, staffID);
69         rs = pst.executeQuery();
70         if (rs.next()) {
71             String cardNumberFromDB = rs.getString("CardNumber"); // Loại bỏ khoảng trắng thừa từ cơ sở dữ liệu
72
73             if (enteredCardNumber.equals(cardNumberFromDB.trim())) {
74                 return cardNumberFromDB; // Trả về cardNumberFromDB nếu trùng khớp
75             } else {
76                 System.out.println("Error: CardNumber không trùng khớp. Input: " + enteredCardNumber + ", DB: " + cardNumberFromDB);
77                 return null; // Không trùng khớp thì trả về null
78             }
79         }
80         return null; // Không tìm thấy bản ghi có StaffID tương ứng
81     } catch (SQLException ex) {
82         System.out.println("Lỗi khi lấy CardNumber: " + ex.getMessage());
83         return null;
84     } finally {
85         try {
86             if (rs != null) {
87                 rs.close();
88             }
89             if (pst != null) {
90                 pst.close();
91             }
92             if (conn != null) {
93                 conn.close();
94             }
95         } catch (SQLException ex) {
96             System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
97         }
98     }
99 }
100
101 private void showAlert(Alert.AlertType type, String title, String content) {
102     Alert alert = new Alert(type);
103     alert.setTitle(title);
104     alert.setContentText(content);
105     alert.showAndWait();
106 }
107
108 @Override
109 public void initialize(URL url, ResourceBundle rb) {
110     // TODO
111 }
112 }
113 
```



```

1  public class FXMLCheckinController implements Initializable {
2
3     @FXML
4     private Button btnCheckin;
5
6     @FXML
7     private TextField txtStaffID;
8
9     @FXML
10    private TextField txtStaffname;
11
12    @FXML
13    private TextField txtCheckin;
14
15    @FXML
16    private Button btncancel;
17
18    private FXMLChamluongController parentController;
19
20    public ConnectDB connect = new ConnectDB();
21    private Connection conn;
22    private PreparedStatement pst;
23    private ResultSet rs;
24
25    public void setParentController(FXMLChamluongController parentController) {
26        this.parentController = parentController;
27    }
28
29    @FXML
30    public void handleCancel() {
31        Stage stage = (Stage) btncancel.getScene().getWindow();
32        stage.close();
33    }
34
35    @FXML
36    public void handleCheckin() {
37        String staffID = txtStaffID.getText();
38        String staffname = txtStaffname.getText();
39        String enteredCardNumber = txtCheckin.getText(); // Lấy giá trị CardNumber mà nhân viên nhập vào
40
41        if (staffID.isEmpty() || staffname.isEmpty() || enteredCardNumber.isEmpty()) {
42            showAlert(Alert.AlertType.ERROR, "Error", "Please enter Staff AND, Staff Name and Card Number.");
43            return;
44        }
45
46        // Kiểm tra StaffID và Staffname có trong tblStaff không
47        boolean validStaff = checkValidStaff(staffID, staffname);
48        if (!validStaff) {
49            showAlert(Alert.AlertType.ERROR, "Error", "Invalid StaffID or Staffname.");
50            return;
51        }
52
53        // Lấy CardNumber từ tblStaff và kiểm tra tính hợp lệ của CardNumber
54        String cardNumber = getCardNumber(staffID, enteredCardNumber);
55        if (cardNumber == null) {
56            showAlert(Alert.AlertType.ERROR, "Error", "CardNumber is invalid or does not match.");
57            return;
58        }
59
60        // Kiểm tra đã checkin trong ngày hôm nay chưa
61        boolean hasCheckedInToday = hasCheckedInToday(staffID);
62        if (hasCheckedInToday) {
63            showAlert(Alert.AlertType.ERROR, "Error", "Staff checked in today.");
64            return;
65        }
66
67        Date currentDate = new Date();
68        SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
69
70        // Cập nhật dữ liệu vào tblTimekeeping
71        String sql = "INSERT INTO tblTimekeeping (StaffID, Staffname, Day, Checkin) VALUES (?, ?, ?, ?)";
72
73        try {
74            conn = connect.getConnectDB();
75            pst = conn.prepareStatement(sql);
76            pst.setString(1, staffID);
77            pst.setString(2, staffname);
78            pst.setString(3, dateFormat.format(currentDate));
79            pst.setString(4, dateFormat.format(currentDate));
80            pst.executeUpdate();
81
82            // Đóng cửa sổ sau khi checkin thành công
83            Stage stage = (Stage) btnCheckin.getScene().getWindow();
84            stage.close();
85
86            // Cập nhật lại TableView trong FXMLChamluongController
87            if (parentController != null) {
88                parentController.refreshTable();
89            }
90
91        } catch (SQLException ex) {
92            showAlert(Alert.AlertType.ERROR, "Error", "Error when performing Checkin:" + ex.getMessage());
93        } finally {
94            try {
95                if (pst != null) {
96                    pst.close();
97                }
98                if (conn != null) {
99                    conn.close();
100                }
101            } catch (SQLException ex) {
102                System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
103            }
104        }
105    }

```

18 Checkout

```
1  public class FXMLCheckoutController implements Initializable {
2
3      @FXML
4      private Button btnCheckout;
5
6      @FXML
7      private Button btncancel;
8
9      @FXML
10     private TextField txtStaffID;
11
12     @FXML
13     private TextField txtStaffname;
14
15     private FXMLChamluongController parentController;
16     private TableChamluong selectedRecord;
17
18     public ConnectDB connect = new ConnectDB();
19     private Connection conn;
20     private PreparedStatement pst;
21
22     public void setParentController(FXMLChamluongController parentController) {
23         this.parentController = parentController;
24     }
25
26     public void setSelectedRecord(TableChamluong selectedRecord) {
27         this.selectedRecord = selectedRecord;
28         // Kiểm tra xem selectedRecord có null hay không
29         if (selectedRecord != null) {
30             // Hiển thị dữ liệu lên các ô nhập
31             txtStaffID.setText(selectedRecord.getStaffID());
32             txtStaffname.setText(selectedRecord.getStaffname());
33         } else {
34             System.out.println("selectedRecord is null");
35         }
36     }
37
38     @FXML
39     public void handleCancel() {
40         Stage stage = (Stage) btncancel.getScene().getWindow();
41         stage.close();
42     }
43
44     @FXML
45     public void handleCheckout() {
46         Date currentDate = new Date();
47         SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
48
49         // Cập nhật dữ liệu vào tblTimekeeping
50         String sql = "UPDATE tblTimekeeping SET Checkout = ? WHERE ID = ?";
51
52         try {
53             conn = connect.getConnectDB();
54             pst = conn.prepareStatement(sql);
55             pst.setString(1, dateFormat.format(currentDate));
56             pst.setInt(2, selectedRecord.getID());
57             pst.executeUpdate();
58
59             // Đóng cửa sổ sau khi checkout thành công
60             Stage stage = (Stage) btnCheckout.getScene().getWindow();
61             stage.close();
62
63             // Cập nhật lại TableView trong FXMLChamluongController
64             if (parentController != null) {
65                 parentController.refreshTable();
66             }
67
68         } catch (SQLException ex) {
69             showAlert(Alert.AlertType.ERROR, "Error", "Error when performing Checkout:" + ex.getMessage());
70         } finally {
71             try {
72                 if (pst != null) {
73                     pst.close();
74                 }
75                 if (conn != null) {
76                     conn.close();
77                 }
78             } catch (SQLException ex) {
79                 System.out.println("Lỗi khi đóng kết nối: " + ex.getMessage());
80             }
81         }
82     }
83
84     private void showAlert(Alert.AlertType type, String title, String content) {
85         Alert alert = new Alert(type);
86         alert.setTitle(title);
87         alert.setContentText(content);
88         alert.showAndWait();
89     }
90
91     @Override
92     public void initialize(URL url, ResourceBundle rb) {
93         // TODO
94     }
95 }
```

19 Staff

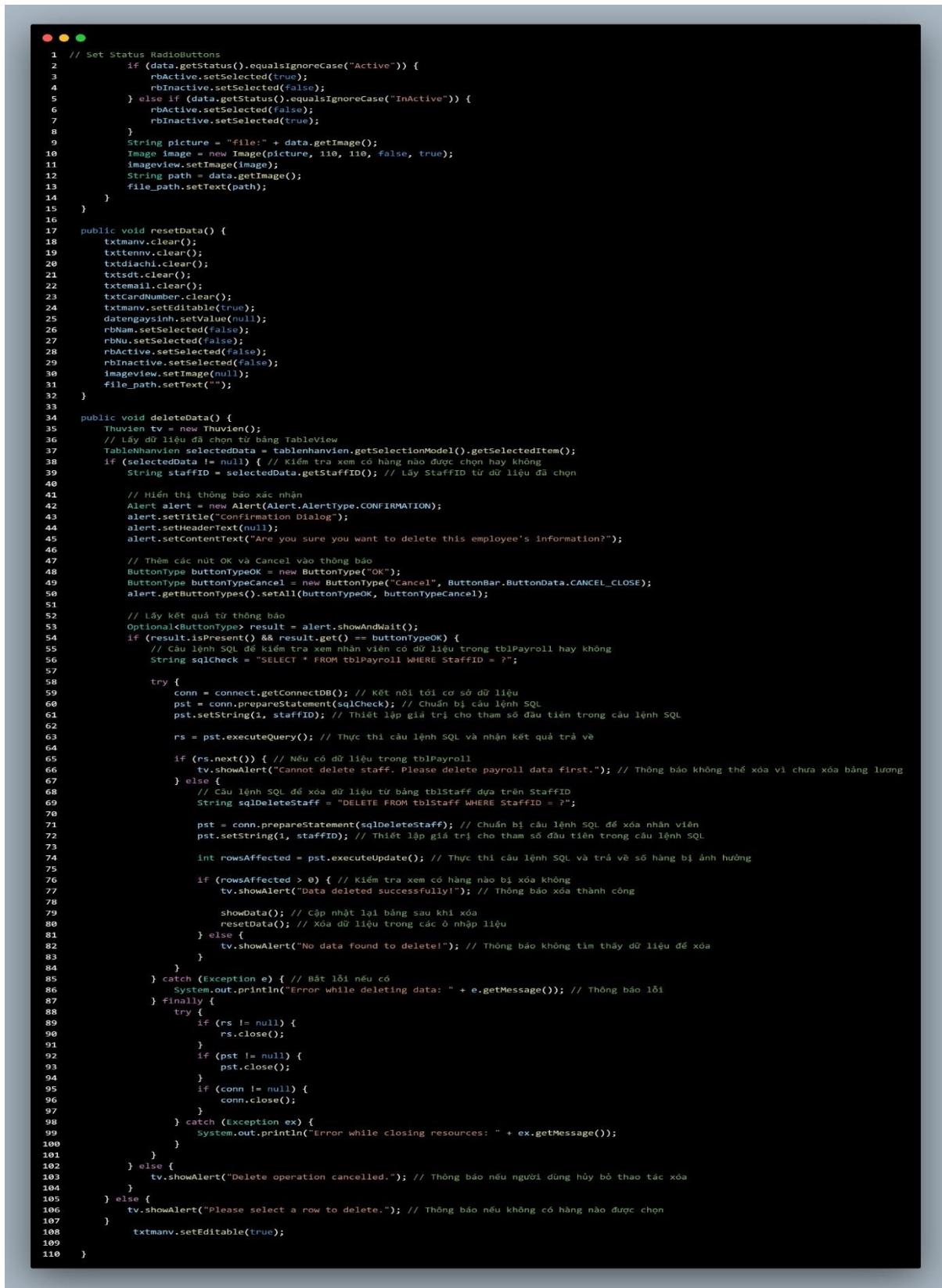
```
1  public class FXMLNhanvienController implements Initializable {
2
3     private BillsController billsController;
4
5     public void setBillsController(Billscontroller billsController) {
6         this.billsController = billsController;
7     }
8
9     @FXML
10    private FXMLMainController mainController;
11
12    private AnchorPane top_main;
13
14    @FXML
15    private Button btnExport;
16
17    @FXML
18    private Button btnreset;
19
20    @FXML
21    private Button btnsua;
22
23    @FXML
24    private TextField txtCardNumber;
25
26    @FXML
27    private Button btnxoa;
28
29    @FXML
30    private RadioButton rbActive;
31
32    @FXML
33    private RadioButton rbInactive;
34
35    @FXML
36    private RadioButton rbNam;
37
38    @FXML
39    private RadioButton rbNu;
40
41    @FXML
42    private Button btnPayroll;
43
44    @FXML
45    private Button btnTimekeeping;
46
47
48    @FXML
49    private TableColumn<TableNhanvien, String> cotaddress;
50
51    @FXML
52    private TableColumn<TableNhanvien, Date> cotbirthday;
53
54    @FXML
55    private TableColumn<TableNhanvien, String> coteemail;
56
57    @FXML
58    private TableColumn<TableNhanvien, String> cotgender;
59
60    @FXML
61    private TableColumn<TableNhanvien, String> cotsdt;
62
63    @FXML
64    private TableColumn<TableNhanvien, String> cotstaffid;
65
66    @FXML
67    private TableColumn<TableNhanvien, String> cotstaffname;
68
69    @FXML
70    private TableColumn<TableNhanvien, String> cotstatus;
71
72    @FXML
73    private DatePicker datengaysinh;
74
75    @FXML
76    private ImageView imageview;
77
78    @FXML
79    private TableView<TableNhanvien> tablenhanvien;
80
81    private TextField txtdiachi;
82
83    @FXML
84    private TextField txtemail;
85
86    @FXML
87    private TextField txtmanv;
88
89    @FXML
90    private TextField txtsdt;
91
92    @FXML
93    private TextField txttennv;
94
95    @FXML
96    private Label file_path;
97
98    private TextField txttukhoa;
99
100   /**
101    * Initializes the controller class.
102    */
103   public ConnectDB connect = new ConnectDB();
104   private Connection conn;
105   private PreparedStatement pst;
106   private ResultSet rs;
107
108   public ObservableList<TableNhanvien> dataList() {
```

19.1 Code

```

1 // Kiểm tra số điện thoại
2     pst = conn.prepareStatement(checkPhoneSQL);
3     pst.setString(1, txtsd.getText());
4     rs = pst.executeQuery();
5     if (rs.next() && rs.getInt("count") > 0) {
6         tv.showAlert("Phone number already exists. Please enter a unique phone number.");
7         return;
8     }
9
10    // Kiểm tra CardNumber
11    pst = conn.prepareStatement(checkCardNumberSQL);
12    pst.setString(1, txtCardNumber.getText());
13    rs = pst.executeQuery();
14    if (rs.next() && rs.getInt("count") > 0) {
15        tv.showAlert("CardNumber already exists. Please enter a unique CardNumber.");
16        return;
17    }
18
19 } catch (Exception e) {
20     System.out.println("Error while checking data: " + e.getMessage());
21     return;
22 }
23
24 // Nếu không có dữ liệu bị trùng lặp, tiếp tục thêm mới nhân viên
25 String sql = "INSERT INTO tblistaff (StaffID, Staffname, Gender, Numberphone, Address, Gmail, Birthday, Status, Image, CardNumber) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
26 try {
27     pst = conn.prepareStatement(sql);
28
29     pst.setString(1, txtmanv.getText()); // StaffID
30     pst.setString(2, txtemnv.getText()); // Staffname
31
32     String gender = rbNam.isSelected() ? "Male" : "Female";
33     pst.setString(3, gender); // Gender
34
35     pst.setString(4, txtsd.getText()); // Numberphone
36     pst.setString(5, txtdiachi.getText()); // Address
37     pst.setString(6, txtemail.getText()); // Gmail
38
39     // Chuyển đổi giá trị Datepicker thành java.sql.Date
40     pst.setDate(7, java.sql.Date.valueOf(datengaysinh.getValue())); // Birthday
41
42     String status = rbActive.isSelected() ? "Active" : "Inactive";
43     pst.setString(8, status); // Status
44
45     pst.setString(9, file_path.getText()); // Image, bạn có thể thay đổi giá trị này để lấy đường dẫn thực tế từ btnclichinh
46
47     pst.setString(10, txtCardNumber.getText()); // CardNumber
48
49     pst.executeUpdate(); // Thực thi câu lệnh SQL
50     tv.showAlert("Staff added successfully!");
51
52     // Cập nhật lại bảng sau khi thêm mới nhân viên
53     showData();
54
55     // Reset các trường dữ liệu sau khi thêm nhân viên
56     resetData();
57
58 } catch (Exception e) {
59     System.out.println(e.getMessage());
60 }
61 }
62
63 public void insertImage() {
64     FileChooser open = new FileChooser();
65     Stage stage = (Stage) top_main.getScene().getWindow();
66     File file = open.showOpenDialog(stage);
67     if (file != null) {
68         String path = file.getAbsolutePath();
69         path = path.replace("\\", "\\\\");
70         file_path.setText(path);
71         Image image = new Image(file.toURI().toString(), 110, 110, false, true);
72         imageview.setImage(image);
73
74     } else {
75
76         System.out.println("NO DATA EXIST!");
77     }
78 }
79
80 // Load dữ liệu lên khung nhân viên
81 public void selectData() {
82     Tableanhvien data = tablenhanvien.getSelectionModel().getSelectedItem();
83     int num = tablenhanvien.getSelectionModel().getSelectedIndex();
84     if (num >= 0) {
85         txtmanv.setText(data.getStaffID());
86         txtsd.setEditable(false); // Vô hiệu hóa chỉnh sửa
87         txtemnv.setText(data.getStaffname());
88         txtdiachi.setText(data.getAddress());
89         txtsd.setText(data.getNumberphone());
90         txtemail.setText(data.getGmail());
91
92         // Set CardNumber
93         txtCardNumber.setText(String.valueOf(data.getCardNumber()));
94
95         // Set Gender RadioButtons
96         if (data.getGender().equalsIgnoreCase("Male")) {
97             rbNam.setSelected(true);
98             rbNu.setSelected(false);
99         } else if (data.getGender().equalsIgnoreCase("Female")) {
100            rbNam.setSelected(false);
101            rbNu.setSelected(true);
102        }
103
104        // Set Datepicker value without using toLocalDate()
105        if (data.getBirthday() != null) {
106            java.sql.Date sqlDate = (java.sql.Date) data.getBirthday();
107            LocalDate localDate = LocalDate.of(sqlDate.getYear() + 1900, sqlDate.getMonth() + 1, sqlDate.getDate());
108            datengaysinh.setValue(localDate);
109        } else {
110            datengaysinh.setValue(null);
111        }
112    }
113 }

```



```

1 // Set Status RadioButtons
2     if (data.getStatus().equalsIgnoreCase("Active")) {
3         rbActive.setSelected(true);
4         rbInactive.setSelected(false);
5     } else if (data.getStatus().equalsIgnoreCase("InActive")) {
6         rbActive.setSelected(false);
7         rbInactive.setSelected(true);
8     }
9     String picture = "file:" + data.getImage();
10    Image image = new Image(picture, 110, 110, false, true);
11    imageview.setImage(image);
12    String path = data.getImage();
13    file_path.setText(path);
14 }
15 }
16
17 public void resetData() {
18     txtmanv.clear();
19     txttenv.clear();
20     txtdiachi.clear();
21     txtsdt.clear();
22     txtemail.clear();
23     txtcardNumber.clear();
24     txtmanv.setEditable(true);
25     datengaysinh.setValue(null);
26     rbName.setSelected(false);
27     rbNu.setSelected(false);
28     rbActive.setSelected(false);
29     rbInactive.setSelected(false);
30     imageview.setImage(null);
31     file_path.setText("");
32 }
33
34 public void deleteData() {
35     Thuvien tv = new Thuvien();
36     // Lấy dữ liệu đã chọn từ bảng TableView
37     Tableanhvien selectedData = tabelanhvien.getSelectionModel().getSelectedItem();
38     if (selectedData != null) { // Kiểm tra xem có hàng nào được chọn hay không
39         String staffID = selectedData.getStaffID(); // Lấy StaffID từ dữ liệu đã chọn
40
41         // Hiển thị thông báo xác nhận
42         Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
43         alert.setTitle("Confirmation Dialog");
44         alert.setHeaderText(null);
45         alert.setContentText("Are you sure you want to delete this employee's information?");
46
47         // Thêm các nút OK và Cancel vào thông báo
48         ButtonType buttonTypeOK = new ButtonType("OK");
49         ButtonType buttonTypeCancel = new ButtonType("Cancel", ButtonBar.ButtonData.CANCEL_CLOSE);
50         alert.getButtonTypes().setAll(buttonTypeOK, buttonTypeCancel);
51
52         // Lấy kết quả từ thông báo
53         Optional<ButtonType> result = alert.showAndWait();
54         if (result.isPresent() && result.get() == buttonTypeOK) {
55             // Câu lệnh SQL để kiểm tra xem nhân viên có dữ liệu trong tblPayroll hay không
56             String sqlCheck = "SELECT * FROM tblPayroll WHERE StaffID = ?";
57
58             try {
59                 conn = connect.getConnectDB(); // Kết nối tới cơ sở dữ liệu
60                 pst = conn.prepareStatement(sqlCheck); // Chuẩn bị câu lệnh SQL
61                 pst.setString(1, staffID); // Thiết lập giá trị cho tham số đầu tiên trong câu lệnh SQL
62
63                 rs = pst.executeQuery(); // Thực thi câu lệnh SQL và nhận kết quả trả về
64
65                 if (rs.next()) { // Nếu có dữ liệu trong tblPayroll
66                     tv.showAlert("Cannot delete staff. Please delete payroll data first."); // Thông báo không thể xóa vì chưa xóa bảng lương
67                 } else {
68                     // Câu lệnh SQL để xóa dữ liệu từ bảng tblStaff dựa trên StaffID
69                     String sqlDeleteStaff = "DELETE FROM tblStaff WHERE StaffID = ?";
70
71                     pst = conn.prepareStatement(sqlDeleteStaff); // Chuẩn bị câu lệnh SQL để xóa nhân viên
72                     pst.setString(1, staffID); // Thiết lập giá trị cho tham số đầu tiên trong câu lệnh SQL
73
74                     int rowsAffected = pst.executeUpdate(); // Thực thi câu lệnh SQL và trả về số hàng bị ảnh hưởng
75
76                     if (rowsAffected > 0) { // Kiểm tra xem có hàng nào bị xóa không
77                         tv.showAlert("Data deleted successfully!"); // Thông báo xóa thành công
78
79                         showData(); // Cập nhật lại bảng sau khi xóa
80                         reloadData(); // Xóa dữ liệu trong các ô nhập liệu
81                     } else {
82                         tv.showAlert("No data found to delete!"); // Thông báo không tìm thấy dữ liệu để xóa
83                     }
84                 }
85             } catch (Exception e) { // Bắt lỗi nếu có
86                 System.out.println("Error while deleting data: " + e.getMessage()); // Thông báo lỗi
87             } finally {
88                 try {
89                     if (rs != null) {
90                         rs.close();
91                     }
92                     if (pst != null) {
93                         pst.close();
94                     }
95                     if (conn != null) {
96                         conn.close();
97                     }
98                 } catch (Exception ex) {
99                     System.out.println("Error while closing resources: " + ex.getMessage());
100                }
101            }
102        } else {
103            tv.showAlert("Delete operation cancelled."); // Thông báo nếu người dùng hủy bỏ thao tác xóa
104        }
105    } else {
106        tv.showAlert("Please select a row to delete."); // Thông báo nếu không có hàng nào được chọn
107    }
108    txtmanv.setEditable(true);
109 }
110 }

```

```

1  public void updateData() {
2      TableNhanvien selectedData = tablenhanvien.getSelectionModel().getSelectedItem();
3
4      if (selectedData != null) {
5          Thuvien tv = new Thuvien();
6
7          // Kiểm tra xem tất cả các trường thông tin có được điền đầy đủ hay không
8          if (txtnamv.getText().isEmpty() || txtCardNumber.getText().isEmpty() || txtnennv.getText().isEmpty() || txtsdt.getText().isEmpty()
9              || txtdiachi.getText().isEmpty() || txtemail.getText().isEmpty() == null) {
10             tv.showAlert("Please enter complete information");
11         } else if (!txtaanv.getText().matches("^\w+\d$")) {
12             tv.showAlert("StaffID must start with 'AN' followed by numbers!");
13         } else if (!txtsdt.getText().matches("^\\d{11}$")) {
14             tv.showAlert("Phone number must contain only digits and be up to 11 digits long!");
15         } else if (!rbNam.isSelected() ^ !rbNu.isSelected()) {
16             tv.showAlert("Please select either 'Male' or 'Female'");
17         } else if (!rbActive.isSelected() ^ !rbInactive.isSelected()) {
18             tv.showAlert("Please select either 'Active' or 'InActive'");
19         } else if (!txtemail.getText().matches("^.+@[a-zA-Z]+\.[a-zA-Z]{2,3}$")) {
20             tv.showAlert("Email must be in the format 'example@gmail.com'");
21         } else {
22             // Kiểm tra StaffID, Gmail, Số điện thoại và CardNumber có bị trùng lặp không
23             String checkGmailSQL = "SELECT COUNT(*) AS count FROM tblstaff WHERE Gmail = ? AND StaffID != ?";
24             String checkPhone = "SELECT COUNT(*) AS count FROM tblstaff WHERE Numberphone = ? AND StaffID != ?";
25             String checkCardNumberSQL = "SELECT COUNT(*) AS count FROM tblstaff WHERE CardNumber = ? AND StaffID != ?";
26
27             try {
28                 conn = connect.getConnection();
29
30                 // Kiểm tra Gmail
31                 pst = conn.prepareStatement(checkGmailSQL);
32                 pst.setString(1, txtemail.getText());
33                 pst.setString(2, selectedData.getStaffID());
34                 ResultSet rs = pst.executeQuery();
35                 if (rs.next() && rs.getInt("count") > 0) {
36                     tv.showAlert("Gmail already exists. Please enter a unique Gmail.");
37                     return;
38                 }
39
40                 // Kiểm tra số điện thoại
41                 pst = conn.prepareStatement(checkPhoneSQL);
42                 pst.setString(1, txtsdt.getText());
43                 pst.setString(2, selectedData.getStaffID());
44                 rs = pst.executeQuery();
45                 if (rs.next() && rs.getInt("count") > 0) {
46                     tv.showAlert("Phone number already exists. Please enter a unique phone number.");
47                     return;
48                 }
49             // Kiểm tra CardNumber
50                 pst = conn.prepareStatement(checkCardNumberSQL);
51                 pst.setString(1, txtCardNumber.getText());
52                 pst.setString(2, selectedData.getStaffID());
53                 rs = pst.executeQuery();
54                 if (rs.next() && rs.getInt("count") > 0) {
55                     tv.showAlert("CardNumber already exists. Please enter a unique CardNumber.");
56                     return;
57                 }
58
59                 String sql = "UPDATE tblstaff SET Staffname = ?, Gender = ?, Numberphone = ?, Address = ?, Gmail = ?, Birthday = ?, Status = ?, Image = ?, CardNumber = ? WHERE StaffID = ?";
60                 pst = conn.prepareStatement(sql);
61
62                 pst.setString(1, txtnamv.getText());
63
64                 String gender = rbNam.isSelected() ? "Male" : "Female";
65                 pst.setString(2, gender);
66
67                 pst.setString(3, txtsdt.getText());
68                 pst.setString(4, txtdiachi.getText());
69                 pst.setString(5, txtemail.getText());
70                 pst.setDate(6, java.sql.Date.valueOf(datengaysinh.getValue()));
71
72                 String status = rbActive.isSelected() ? "Active" : "InActive";
73                 pst.setString(7, status);
74
75                 pst.setString(8, file_path.getText());
76                 pst.setString(9, txtCardNumber.getText());
77                 pst.setString(10, selectedData.getStaffID());
78
79                 int rowsAffected = pst.executeUpdate(); // Thực thi câu lệnh SQL và trả về số hàng bị ảnh hưởng
80
81                 if (rowsAffected > 0) {
82                     tv.showAlert("Data updated successfully!");
83
84                     showData(); // Cập nhật lại bảng sau khi cập nhật dữ liệu
85                     resetData(); // Xóa dữ liệu trong các ô nhập liệu
86                 } else {
87                     tv.showAlert("No data found to update!");
88                 }
89             } catch (Exception e) {
90                 System.out.println("Error while updating data: " + e.getMessage());
91             } finally {
92                 try {
93                     if (rs != null) {
94                         rs.close();
95                     }
96                     if (pst != null) {
97                         pst.close();
98                     }
99                     if (conn != null) {
100                         conn.close();
101                     }
102                 } catch (Exception e) {
103                     System.out.println("Error while closing connection: " + e.getMessage());
104                 }
105             }
106         }
107     }
108 }

```

```

1 } else {
2     Thuvien tv = new Thuvien();
3     tv.showAlert("Please select a row to update.");
4 }
5 }

6 public void searchStaff() {
7     String keyword = txttukhoa.getText().toLowerCase();
8
9     ObservableList<TableNhanvien> filteredList = FXCollections.observableArrayList();
10
11    for (TableNhanvien staff : dataList) {
12        if (staff.getStaffID().toLowerCase().contains(keyword)
13            || staff.getStaffname().toLowerCase().contains(keyword)) {
14            filteredList.add(staff);
15        }
16    }
17
18    tablenhanvien.setItems(filteredList);
19 }
20 }

// Chức năng xuất dữ liệu ra excel bằng nút Export
21 public void exportToCSV() {
22    FileChooser fileChooser = new FileChooser();
23     fileChooser.setTitle("Save CSV File");
24     fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "*.csv"));
25     File selectedFile = fileChooser.showSaveDialog(btnExport.getScene().getWindow());
26
27     if (selectedFile != null) {
28         try {
29             Writer writer = new BufferedWriter(new OutputStreamWriter(new FileOutputStream(selectedFile), StandardCharsets.UTF_8));
30             writer.write('\uFEFF'); // BOM for UTF-8
31
32             // Ghi tiêu đề cột
33             TableView<TableNhanvien> tableView = tablenhanvien;
34             ObservableList<TableColumn<TableNhanvien, ?>> columns = tableView.getColumns();
35             for (int i = 0; i < columns.size(); i++) {
36                 TableColumn<TableNhanvien, ?> column = columns.get(i);
37                 if (i > 0) {
38                     writer.write(",");
39                 }
40                 writer.write(column.getText());
41             }
42             writer.write("\n");
43
44             // Ghi dữ liệu từng dòng
45             ObservableList<TableNhanvien> dataList = tableView.getItems();
46             for (TableNhanvien staff : dataList) {
47                 for (int i = 0; i < columns.size(); i++) {
48                     TableColumn<TableNhanvien, ?> column = columns.get(i);
49                     if (i > 0) {
50                         writer.write(",");
51                     }
52                     writer.write("");
53                 }
54                 // Lấy giá trị từ cell của bảng và ghi vào writer
55                 Object cellValue = column.getCellData(staff);
56                 writer.write(cellValue != null ? cellValue.toString() : "");
57             }
58             writer.write("\n");
59         }
60         writer.flush();
61         writer.close();
62
63         Thuvien tv = new Thuvien(); // Thêm dòng này nếu Thuvien cần được sử dụng
64         tv.showAlert("Data exported to CSV successfully!");
65
66         // Mở file CSV sau khi export
67         Desktop.getDesktop().open(selectedFile);
68
69     } catch (IOException e) {
70         System.out.println("Error while exporting to CSV: " + e.getMessage());
71     }
72 }
73 }

74 public void setMainController(FXMLMainController mainController) {
75     this.mainController = mainController;
76 }
77 }

78 // Chuyển form sang luồng
79 @FXML
80 private void gotoChamLuongForm() {
81     if (mainController != null) {
82         mainController.loadFXML("/demoproject1/FXMLChamluong.fxml");
83     } else {
84         System.err.println("FXMLNhanVienController: MainController is null, cannot load FXMLChamluong.fxml");
85     }
86 }
87 }

88 @FXML
89 private void goToBangLuongForm() {
90     if (mainController != null) {
91         mainController.loadFXML("/demoproject1/FMLBangluong.fxml");
92     } else {
93         System.err.println("FXMLNhanVienController: MainController is null, cannot load FXMLBangluong.fxml");
94     }
95 }

96 //Override
97 public void initialize(URL url, ResourceBundle rb) {
98     // Các phần khởi tạo khác
99     btnreset.setOnAction(event -> resetData());
100    btnxoa.setOnAction(event -> deleteData());
101    btnsua.setOnAction(event -> updateData());
102    btnExport.setOnAction(event -> exportToCSV()); // Xuất ra excel
103    showData();
104
105    // Gọi phương thức tìm kiếm khi người dùng nhập từ khóa
106    txttukhoa.textProperty().addListener((observable, oldValue, newValue) -> {
107        searchStaff();
108    });
109 }
110 }

111 }

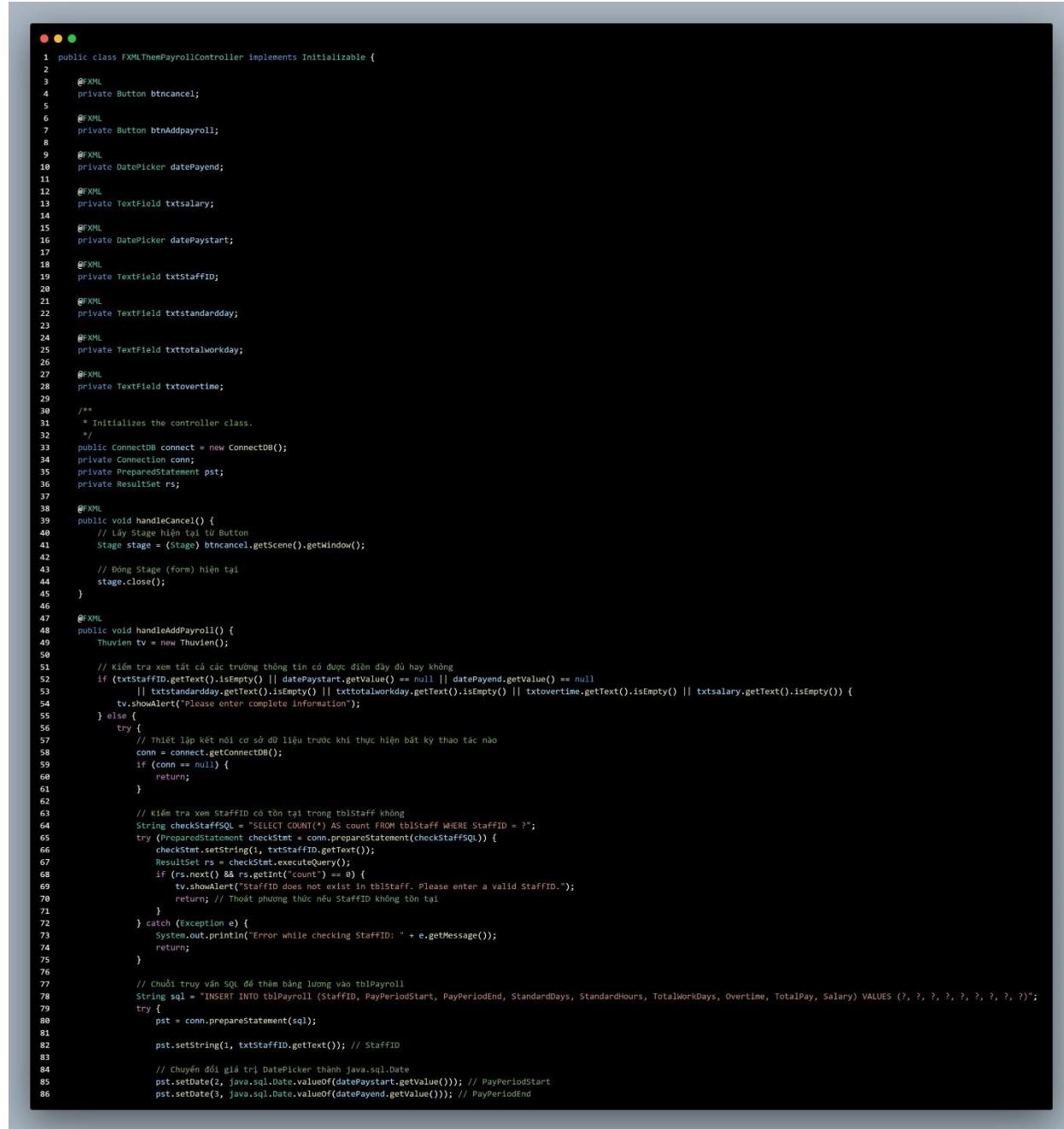
112 }

113 }

114 }

```

20 Add Payroll



```

1  public class FXMLThemPayrollController implements Initializable {
2
3      @FXML
4      private Button btncancel;
5
6      @FXML
7      private Button btnAddpayroll;
8
9      @FXML
10     private DatePicker datePayend;
11
12     @FXML
13     private TextField txtsalary;
14
15     @FXML
16     private DatePicker datePaystart;
17
18     @FXML
19     private TextField txtStaffID;
20
21     @FXML
22     private TextField txtstandardday;
23
24     @FXML
25     private TextField txttotalworkday;
26
27     @FXML
28     private TextField txtovertime;
29
30     /**
31      * Initializes the controller class.
32     */
33     public ConnectDB connect = new ConnectDB();
34     private Connection conn;
35     private PreparedStatement pst;
36     private ResultSet rs;
37
38     @FXML
39     public void handleCancel() {
40         // Lấy Stage hiện tại từ Button
41         Stage stage = (Stage) btncancel.getScene().getWindow();
42
43         // Đóng Stage (form) hiện tại
44         stage.close();
45     }
46
47     @FXML
48     public void handleAddPayroll() {
49         Thuvien tv = new Thuvien();
50
51         // Kiểm tra xem tất cả các trường thông tin có được điền đầy đủ hay không
52         if (txtStaffID.getText().isEmpty() || datePaystart.getValue() == null || datePayend.getValue() == null
53             || txtstandardday.getText().isEmpty() || txttotalworkday.getText().isEmpty() || txtovertime.getText().isEmpty() || txtsalary.getText().isEmpty()) {
54             tv.showAlert("Please enter complete information");
55         } else {
56             try {
57                 // Thiết lập kết nối cơ sở dữ liệu trước khi thực hiện bất kỳ thao tác nào
58                 conn = connect.getConnectDB();
59                 if (conn == null) {
60                     return;
61                 }
62
63                 // Kiểm tra xem StaffID có tồn tại trong tblStaff không
64                 String checkStaffSQL = "SELECT COUNT(*) AS count FROM tblStaff WHERE StaffID = ?";
65                 try {
66                     PreparedStatement checkStatmt = conn.prepareStatement(checkStaffSQL);
67                     checkStatmt.setString(1, txtStaffID.getText());
68                     ResultSet rs = checkStatmt.executeQuery();
69                     if (rs.next() && rs.getInt("count") == 0) {
70                         tv.showAlert("StaffID does not exist in tblStaff. Please enter a valid StaffID.");
71                         return; // Thoát phương thức nếu StaffID không tồn tại
72                     }
73                 } catch (Exception e) {
74                     System.out.println("Error while checking StaffID: " + e.getMessage());
75                     return;
76                 }
77
78                 // Chuỗi truy vấn SQL để thêm bảng lương vào tblPayroll
79                 String sql = "INSERT INTO tblPayroll (StaffID, PayPeriodStart, PayPeriodEnd, StandardDays, StandardHours, TotalWorkDays, Overtime, TotalPay, Salary) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
80                 try {
81                     pst = conn.prepareStatement(sql);
82                     pst.setString(1, txtStaffID.getText()); // StaffID
83
84                     // Chuyển đổi giá trị DatePicker thành java.sql.Date
85                     pst.setDate(2, java.sql.Date.valueOf(datePaystart.getValue())); // PayPeriodStart
86                     pst.setDate(3, java.sql.Date.valueOf(datePayend.getValue())); // PayPeriodEnd

```

20.1 Code

```
1     pst.setInt(4, Integer.parseInt(txtstandardday.getText())); // StandardDays
2
3         // Thiết lập giá trị mặc định cho StandardHours là 8
4         int defaultStandardHours = 8;
5         pst.setInt(5, defaultStandardHours); // StandardHours
6
7         // Thiết lập giá trị TotalWorkDays từ người dùng nhập vào
8         int totalWorkDays = Integer.parseInt(txtnumberofdays.getText());
9         pst.setInt(6, totalWorkDays); // TotalWorkDays
10
11        // Thiết lập giá trị Overtime từ người dùng nhập vào
12        BigDecimal overtime = new BigDecimal(txtovertime.getText());
13        pst.setBigDecimal(7, overtime); // Overtime
14
15        // Thiết lập giá trị Salary từ người dùng nhập vào
16        BigDecimal salary = new BigDecimal(txtsalary.getText());
17        pst.setBigDecimal(8, salary); // Salary
18
19        // Tính toán và thiết lập giá trị cho TotalPay
20        BigDecimal totalPay = calculateTotalPay(salary, Integer.parseInt(txtnumberofdays.getText()), defaultStandardHours, totalWorkDays, overtime);
21        pst.setBigDecimal(9, totalPay); // TotalPay
22
23        // Thực thi câu lệnh SQL
24        pst.executeUpdate();
25        tv.showAlert("Payroll added successfully!");
26
27        // Đóng cửa sổ hiện tại của FXMLThemPayrollController
28        Stage stage = (Stage) btnAddpayroll.getScene().getWindow();
29        stage.close();
30
31        // Cập nhật lại bảng sau khi thêm mới Payroll
32        FXMLLoader loader = new FXMLLoader(getClass().getResource("FXMLBangluong.fxml"));
33        Parent parent = loader.load();
34        FXMLBangluongController controller = loader.getController();
35        controller.showDataBangluong(); // Gọi phương thức cập nhật bảng
36
37    } catch (Exception e) {
38        System.out.println("Error while adding payroll: " + e.getMessage());
39    }
40    } catch (Exception ex) {
41        System.out.println("Error while establishing database connection: " + ex.getMessage());
42    }
43}
44}
45
46 private BigDecimal calculateTotalPay(BigDecimal salary, int standardDays, int standardHours, int totalWorkDays, BigDecimal overtimeHours) {
47     // Tính số tiền làm được trong 1 giờ
48     BigDecimal hourlyRate = salary.divide(BigDecimal.valueOf(standardDays), BigDecimal.ROUND_HALF_UP)
49         .divide(BigDecimal.valueOf(standardHours), BigDecimal.ROUND_HALF_UP);
50
51     // Tính tổng số giờ làm việc trong tháng
52     int totalActualHours = totalWorkDays * standardHours;
53
54     // Tính toán lương từ số giờ làm việc thực tế và số giờ tăng ca
55     BigDecimal totalPay = hourlyRate.multiply(BigDecimal.valueOf(totalActualHours))
56         .add(hourlyRate.multiply(overtimeHours));
57
58     return totalPay;
59 }
60
61 @Override
62 public void initialize(URL url, ResourceBundle rb) {
63     btncancel.setOnAction(event -> handleCancel());
64     btnAddpayroll.setOnAction(event -> handleAddPayroll());
65 }
66 }
```

21 Update Payroll

```

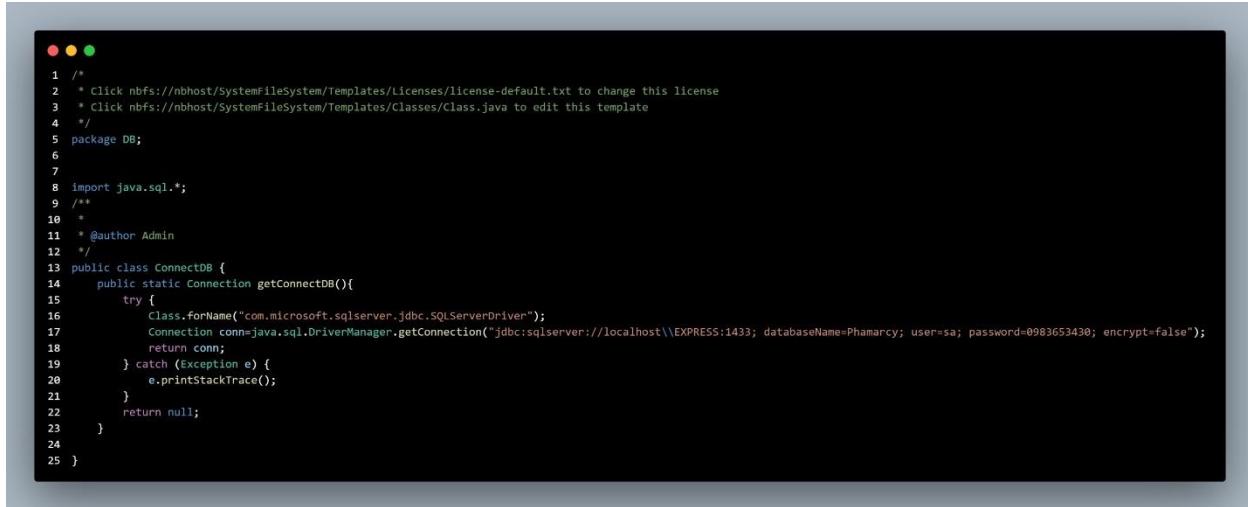
1  public class FXMLUpdatePayrollController implements Initializable {
2
3      @FXML
4      private Button btnUpdatepayroll; // Nút để cập nhật thông tin lương
5
6      @FXML
7      private Button btncancel; // Nút để hủy bỏ và đóng form
8
9      @FXML
10     private DatePicker datePayend; // DatePicker cho ngày kết thúc kỳ trả lương
11
12     @FXML
13     private DatePicker datePaystart; // DatePicker cho ngày bắt đầu kỳ trả lương
14
15     @FXML
16     private TextField txtStaffID; // TextField cho ID nhân viên
17
18     @FXML
19     private TextField txtstandardday; // TextField cho số ngày tiêu chuẩn
20
21     @FXML
22     private TextField txttotalworkday;
23
24     @FXML
25     private TextField txtsalary;
26
27     @FXML
28     private TextField txtovertime;
29
30     public ConnectDB connect = new ConnectDB(); // Kết nối cơ sở dữ liệu
31     private Connection conn; // Kết nối SQL
32     private PreparedStatement pst; // Câu lệnh SQL đã chuẩn bị
33     private ResultSet rs; // Kết quả trả về từ câu lệnh SQL
34
35     @FXML
36     public void handleCancel() {
37         Stage stage = (Stage) btncancel.getScene().getWindow();
38         stage.close();
39     }
40
41     private Table payrollData; // Lưu trữ dữ liệu từ bảng lương
42
43     private FXMLBangluongController parentController; // Tham chiếu đến controller cha để làm mới bảng
44
45     public void setParentController(FXMLBangluongController parentController) {
46         this.parentController = parentController;
47     }
48
49     public void setUpdatePayroll(Table data) {
50         payrollData = data; // Lưu trữ dữ liệu từ dòng được chọn
51         txtStaffID.setText(data.getStaffID());
52         txtStaffID.setEditable(false); // Không cho phép chỉnh sửa trường Staff ID
53
54         java.util.Date payPeriodstartDate = new java.util.Date(data.getPayPeriodStart().getTime());
55         java.util.Date payPeriodendDate = new java.util.Date(data.getPayPeriodEnd().getTime());
56
57         datePaystart.setValue(payPeriodstartDate.toInstant().atZone(ZoneId.systemDefault()).toLocalDate());
58         datePayend.setValue(payPeriodendDate.toInstant().atZone(ZoneId.systemDefault()).toLocalDate());
59
60         txtstandardday.setText(String.valueOf(data.getStandardDays()));
61         txttotalworkday.setText(String.valueOf(data.getTotalWorkDays()));
62         txtsalary.setText(String.valueOf(data.getSalary()));
63         txtovertime.setText(String.valueOf(data.getOvertime()));
64     }
65
66     @FXML
67     public void handleUpdate() {
68         if (txtStaffID.getText().isEmpty() || datePaystart.getValue() == null || datePayend.getValue() == null
69             || txtstandardday.getText().isEmpty() || txttotalworkday.getText().isEmpty() || txtovertime.getText().isEmpty() || txtsalary.getText().isEmpty()) {
70             showAlert("Please enter complete information");
71         } else {
72             try {
73                 conn = connect.getConnectDB();
74                 if (conn == null) {
75                     return;
76                 }
77
78                 String sql = "UPDATE tblPayroll SET PayPeriodStart = ?, PayPeriodEnd = ?, StandardDays = ?, StandardHours = ?, TotalWorkDays = ?, Overtime = ?, TotalPay = ?, Salary = ? WHERE StaffID = ?";
79
80                 pst = conn.prepareStatement(sql);
81
82                 pst.setDate(1, java.sql.Date.valueOf(datePaystart.getValue())); // PayPeriodStart
83                 pst.setDate(2, java.sql.Date.valueOf(datePayend.getValue())); // PayPeriodEnd
84
85                 int standardDays = Integer.parseInt(txtstandardday.getText());
86                 pst.setInt(3, standardDays); // StandardDays
87
88                 int defaultStandardHours = 8;
89                 pst.setInt(4, defaultStandardHours); // StandardHours
90
91                 BigDecimal totalWorkDays = new BigDecimal(txttotalworkday.getText());
92                 pst.setBigDecimal(5, totalWorkDays); // TotalWorkDays
93
94                 BigDecimal overtime = new BigDecimal(txtovertime.getText());
95                 pst.setBigDecimal(6, overtime); // Overtime
96             }
97         }
98     }
99
100 }

```

21.1 Code

```
1      BigDecimal salary = new BigDecimal(txtsalary.getText());
2      pst.setBigDecimal(8, salary); // Salary
3
4      BigDecimal totalPay = calculateTotalPay(salary, standardDays, defaultStandardHours, totalWorkDays, overtime);
5      pst.setBigDecimal(7, totalPay); // TotalPay
6
7      pst.setString(9, txtStaffID.getText()); // StaffID
8
9      int rowsAffected = pst.executeUpdate();
10     if (rowsAffected > 0) {
11         showAlert("Payroll data updated successfully.");
12
13         payrollData.setPayPeriodStart((java.sql.Date.valueOf(datePaystart.getValue())));
14         payrollData.setPayPeriodEnd((java.sql.Date.valueOf(datePayend.getValue())));
15         payrollData.setStandardDays(standardDays);
16         payrollData.setTotalWorkDays(totalWorkDays);
17         payrollData.setOvertime(overtime);
18         payrollData.setSalary(salary);
19         payrollData.setTotalPay(totalPay);
20
21         if (parentController != null) {
22             parentController.showDataBangluong();
23         }
24
25         Stage stage = (Stage) btnUpdatepayroll.getScene().getWindow();
26         stage.close();
27     }
28 } catch (SQLException e) {
29     System.out.println("Error while updating payroll: " + e.getMessage());
30 } finally {
31     closeResources();
32 }
33
34 }
35
36
37 @Override
38 public void initialize(URL url, ResourceBundle rb) {
39     btnUpdatepayroll.setOnAction(event -> handleUpdate());
40     btnCancel.setOnAction(event -> handleCancel());
41 }
42
43 private void showAlert(String message) {
44     Alert alert = new Alert(Alert.AlertType.INFORMATION);
45     alert.setTitle("Information");
46     alert.setHeaderText(null);
47     alert.setContentText(message);
48     alert.showAndWait();
49 }
50
51 private BigDecimal calculateTotalPay(BigDecimal salary, int standardDays, int standardHours, BigDecimal totalWorkDays, BigDecimal overtimeHours) {
52     // Số tiền làm được trong 1 giờ
53     BigDecimal hourlyRate = salary.divide(BigDecimal.valueOf(standardDays), BigDecimal.ROUND_HALF_UP)
54         .divide(BigDecimal.valueOf(standardHours), BigDecimal.ROUND_HALF_UP);
55
56     // Tổng số giờ làm việc trong tháng
57     BigDecimal totalActualHours = totalWorkDays.multiply(BigDecimal.valueOf(standardHours));
58
59     // Tính toán lương từ số giờ làm việc thực tế và số giờ tăng ca
60     BigDecimal totalPay = hourlyRate.multiply(totalActualHours)
61         .add(hourlyRate.multiply(overtimeHours));
62
63     return totalPay;
64 }
65
66 private void closeResources() {
67     try {
68         if (pst != null) {
69             pst.close();
70         }
71         if (conn != null) {
72             conn.close();
73         }
74     } catch (SQLException e) {
75         System.out.println("Error while closing connection: " + e.getMessage());
76     }
77 }
78 }
```

22 Connect Database



```
1 /*
2  * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3  * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
4  */
5 package DB;
6
7
8 import java.sql.*;
9 /**
10 *
11 * @author Admin
12 */
13 public class ConnectDB {
14     public static Connection getConnectDB(){
15         try {
16             Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
17             Connection conn=java.sql.DriverManager.getConnection("jdbc:sqlserver://localhost\\EXPRESS:1433; databaseName=Pharmacy; user=sa; password=0983653430; encrypt=false");
18             return conn;
19         } catch (Exception e) {
20             e.printStackTrace();
21         }
22         return null;
23     }
24 }
25 }
```

23 Account

```

1  public class AccountInfoController implements Initializable {
2
3      @FXML
4      private TableColumn<AccountData, Void> Acc_Col_AC;
5
6      @FXML
7      private TableColumn<AccountData, String> Acc_Col_Address;
8
9      @FXML
10     private TableColumn<AccountData, LocalDateTime> Acc_Col_DateCreated;
11
12     @FXML
13     private TableColumn<AccountData, String> Acc_Col_Email;
14
15     @FXML
16     private TableColumn<AccountData, String> Acc_Col_FullName;
17
18     @FXML
19     private TableColumn<AccountData, String> Acc_Col_Password;
20
21     @FXML
22     private TableColumn<AccountData, String> Acc_Col_Phone;
23
24     @FXML
25     private TableColumn<AccountData, String> Acc_Col_Role;
26
27     @FXML
28     private Pagination pgtAL;
29
30     @FXML
31     private TextField searchAL;
32
33     @FXML
34     private TableView<AccountData> tableaccoutlist;
35
36     private Connection conn;
37     private PreparedStatement ps;
38     private ResultSet rs;
39     Message alert = new Message();
40
41     public ObservableList<AccountData> DataList() {
42         ObservableList<AccountData> Datalist = FXCollections.observableArrayList();
43         String sql = "SELECT i.ID, i.FullName, i.Phone, i.Email, i.Address, i.Date_Created, r.Role " +
44             "FROM Information i " +
45             "JOIN Login l ON i.Phone = l.username " +
46             "JOIN Role r ON l.username = r.UserName";
47         try {
48             conn = DB.ConnectDB.getConnectDB();
49             ps = conn.prepareStatement(sql);
50             rs = ps.executeQuery();
51             AccountData data;
52             while (rs.next()) {
53                 Timestamp timestamp = rs.getTimestamp("Date_Created");
54                 LocalDateTime datecreated = timestamp.toLocalDateTime();
55                 data = new AccountData(rs.getInt("ID"),
56                     rs.getString("FullName"),
57                     rs.getString("Email"),
58                     rs.getString("Phone"),
59                     rs.getString("Address"),
60                     rs.getString("Password"),
61                     rs.getString("Role"),
62                     datecreated
63
64                 );
65                 Datalist.add(data);
66             }
67         } catch (Exception e) {
68             System.out.println(e.getMessage());
69         } finally {
70             try {
71                 if (rs != null) {
72                     rs.close();
73                 }
74                 if (ps != null) {
75                     ps.close();
76                 }
77                 if (conn != null) {
78                     conn.close();
79                 }
80             } catch (SQLException e) {
81                 System.out.println("Error closing resources: " + e.getMessage());
82             }
83         }
84     }
85     return Datalist;
86 }
87
88     public void showData() {
89         ObservableList<AccountData> showList = DataList();
90
91         Acc_Col_Address.setCellValueFactory(new PropertyValueFactory<>("Address"));
92         Acc_Col_DateCreated.setCellValueFactory(new PropertyValueFactory<>("DateCreated"));
93         Acc_Col_Email.setCellValueFactory(new PropertyValueFactory<>("Email"));
94         Acc_Col_FullName.setCellValueFactory(new PropertyValueFactory<>("FullName"));
95         Acc_Col_Password.setCellValueFactory(new PropertyValueFactory<>("Password"));
96         Acc_Col_Phone.setCellValueFactory(new PropertyValueFactory<>("Phone"));
97         Acc_Col_Role.setCellValueFactory(new PropertyValueFactory<>("Role"));
98         Acc_Col_AC.setCellFactory(new Callback<TableColumn<AccountData, Void>, TableCell<AccountData, Void>>() {
99             @Override
100            public TableCell<AccountData, Void> call(final TableColumn<AccountData, Void> param) {
101                final TableCell<AccountData, Void> cell = new TableCell<AccountData, Void>() {
102
103                    private final Button btnUpdate = new Button("Update");
104                    private final Button btnDelete = new Button("Delete");
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
387
388
389
389
390
391
392
393
394
395
396
397
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
417
418
419
419
420
421
422
423
424
425
426
427
427
428
429
429
430
431
432
433
434
435
436
437
437
438
439
439
440
441
442
443
444
445
446
447
447
448
449
449
450
451
452
453
454
455
456
457
457
458
459
459
460
461
462
463
464
465
466
467
467
468
469
469
470
471
472
473
474
475
476
477
477
478
479
479
480
481
482
483
484
485
486
487
487
488
489
489
490
491
492
493
494
495
496
497
497
498
499
499
500
501
502
503
504
505
506
507
507
508
509
509
510
511
512
513
514
515
516
517
517
518
519
519
520
521
522
523
524
525
526
527
527
528
529
529
530
531
532
533
534
535
536
537
537
538
539
539
540
541
542
543
544
545
546
547
547
548
549
549
550
551
552
553
554
555
556
557
557
558
559
559
560
561
562
563
564
565
566
567
567
568
569
569
570
571
572
573
574
575
576
577
577
578
579
579
580
581
582
583
584
585
586
587
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
607
608
609
609
610
611
612
613
614
615
616
616
617
618
618
619
619
620
621
622
623
624
625
626
626
627
628
628
629
629
630
631
632
633
634
635
636
637
637
638
639
639
640
641
642
643
644
645
646
646
647
648
648
649
649
650
651
652
653
654
655
656
657
657
658
659
659
660
661
662
663
664
665
666
667
667
668
669
669
670
671
672
673
674
675
676
676
677
678
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
707
708
709
709
710
711
712
713
714
715
716
716
717
718
718
719
719
720
721
722
723
724
725
726
726
727
728
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
746
747
748
748
749
749
750
751
752
753
754
755
756
757
757
758
759
759
760
761
762
763
764
765
766
766
767
768
768
769
769
770
771
772
773
774
775
776
776
777
778
778
779
779
780
781
782
783
784
785
786
786
787
788
788
789
789
790
791
792
793
794
795
796
796
797
798
798
799
799
800
801
802
803
804
805
806
806
807
808
808
809
809
810
811
812
813
814
815
815
816
817
817
818
818
819
819
820
821
822
823
824
825
826
826
827
828
828
829
829
830
831
832
833
834
835
836
836
837
838
838
839
839
840
841
842
843
844
845
845
846
847
847
848
848
849
849
850
851
852
853
854
855
855
856
857
857
858
858
859
859
860
861
862
863
864
865
865
866
867
867
868
868
869
869
870
871
872
873
874
875
875
876
877
877
878
878
879
879
880
881
882
883
884
885
885
886
887
887
888
888
889
889
890
891
892
893
894
895
895
896
897
897
898
898
899
900
901
902
903
904
905
906
907
907
908
909
909
910
911
912
913
914
915
915
916
917
917
918
918
919
919
920
921
922
923
924
925
925
926
927
927
928
928
929
929
930
931
932
933
934
935
935
936
937
937
938
938
939
939
940
941
942
943
944
944
945
946
946
947
947
948
948
949
949
950
951
952
953
954
955
955
956
957
957
958
958
959
959
960
961
962
963
964
964
965
966
966
967
967
968
968
969
969
970
971
972
973
974
975
975
976
977
977
978
978
979
979
980
981
982
983
984
984
985
986
986
987
987
988
988
989
989
990
991
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579

```

23.1 Code

```

1  private void openUpdateForm(AccountData pdata) {
2      try {
3          Data.al_ID = pdata.getId();
4          Data.al_email = pdata.getEmail();
5          Data.al_datecreated = pdata.getDateCreated();
6          Data.al_role = pdata.getRole();
7          Data.al_address = pdata.getAddress();
8          Data.al_password = pdata.getPassword();
9          Data.al_phone = pdata.getPhone();
10         Data.al_fullname = pdata.getFullName();
11
12         FXMLLoader loader = new FXMLLoader(getClass().getResource("/AccountInfo/UpdateInfo.fxml"));
13
14         Parent root = loader.load();
15
16         Stage stage = new Stage();
17         stage.initStyle(StageStyle.UNDECORATED);
18         stage.setScene(new Scene(root));
19         stage.show();
20     } catch (Exception e) {
21         e.printStackTrace();
22     }
23 }
24
25
26 private void filterData(String keyword) {
27     ObservableList<AccountData> filteredList = FXCollections.observableArrayList();
28     String sql = "SELECT i.ID, i.Fullname, i.Email, i.Phone, i.Address, i.Password, r.Role, i.Date_Created " +
29         "FROM Information i " +
30         "INNER JOIN Login l ON i.Phone = l.Username " + // Thêm khoảng trắng sau "l.Username"
31         "INNER JOIN Role r ON l.Username = r.Username " + // Thêm khoảng trắng sau "r.Username"
32         "WHERE i.Fullname LIKE ? OR i.Phone LIKE ? OR i.Email LIKE ?";
33
34     try {
35         conn = DB.ConnectDB.getConnection();
36         ps = conn.prepareStatement(sql);
37         String searchkeyword = "%" + keyword + "%";
38         ps.setString(1, searchkeyword);
39         ps.setString(2, searchkeyword);
40         ps.setString(3, searchkeyword);
41         rs = ps.executeQuery();
42
43         while (rs.next()) {
44             Timestamp timestamp = rs.getTimestamp("Date_Created");
45             LocalDateTime datecreated = timestamp.toLocalDateTime();
46             AccountData data = new AccountData(
47                 rs.getInt("ID"),
48                 rs.getString("FullName"),
49                 rs.getString("Email"),
50                 rs.getString("Phone"),
51                 rs.getString("Address"),
52                 rs.getString("Password"),
53                 rs.getString("Role"),
54                 datecreated
55             );
56             filteredList.add(data);
57         }
58     } catch (Exception e) {
59         e.printStackTrace();
60     } finally {
61         try {
62             if (rs != null) {
63                 rs.close();
64             }
65             if (ps != null) {
66                 ps.close();
67             }
68             if (conn != null) {
69                 conn.close();
70             }
71         } catch (SQLException e) {
72             e.printStackTrace();
73         }
74     }
75     tableaccoutlist.setItems(filteredList);
76 }
77
78 public void Search() {
79     searchAL.textProperty().addListener((observable, oldValue, newValue) -> {
80         filterData(newValue);
81     });
82 }
83
84 private static final int itemsPerPage = 23;
85
86 private int getPageCount() {
87     return (int) Math.ceil((double) DataList().size() / itemsPerPage);
88 }
89
90 private void updateTable(int pageIndex) {
91     int fromIndex = pageIndex * itemsPerPage;
92     int toIndex = Math.min(fromIndex + itemsPerPage, DataList().size());
93     tableaccoutlist.setItems(FXCollections.observableArrayList(DataList().subList(fromIndex, toIndex)));
94 }
95
96 private Node createPage(int pageIndex) {
97     updateTable(pageIndex);
98     return tableaccoutlist;
99 }
100
101 public void Pagination() {
102     pgtAL.setPageCount(getPageCount());
103     pgtAL.setPageFactory(this::createPage);
104 }
105
106 @Override
107 public void initialize(URL url, ResourceBundle rb) {
108     ControllerHolder.getInstance().setAccountinfoController(this);
109     showData();
110     Pagination();
111     Search();
112 }
113
114 }

```

24 Update Account

```

1  public class UpdateInfoController implements Initializable {
2
3      @FXML
4      private Button btnCancelUpdate;
5
6      @FXML
7      private Button btnUpdate;
8
9      @FXML
10     private Button btnUpdateClose;
11
12     @FXML
13     private ComboBox<String> cbRole;
14
15     @FXML
16     private TextField txtAddress;
17
18     @FXML
19     private TextField txtFullName;
20
21     @FXML
22     private TextField txtPassword;
23
24     @FXML
25     private TextField txtPhone;
26
27     @FXML
28     private TextField txtEmail;
29
30
31     @FXML
32     void Cancelbtn(ActionEvent event) {
33         updateDruglistForm();
34         Stage stage = (Stage) btnCancelUpdate.getScene().getWindow();
35         stage.close();
36     }
37
38     @FXML
39     void Closebtn(ActionEvent event) {
40         updateDruglistForm();
41         Stage stage = (Stage) btnUpdateClose.getScene().getWindow();
42         stage.close();
43     }
44
45
46     private Connection conn;
47     private PreparedStatement ps;
48     private ResultSet rs;
49     Message alert = new Message();
50
51     public void setField() {
52         txtFullName.setText(Data.al_fullname);
53         txtAddress.setText(Data.al_address);
54         txtPhone.setText(Data.al_phone);
55         txtPassword.setText(Data.al_password);
56         cbRole.setValue(Data.al_role);
57         txtEmail.setText(Data.al_email);
58     }
59
60     public class ValidateRegister {
61
62         public static boolean isNotEmpty(String input) {
63             return input != null && !input.trim().isEmpty();
64         }
65
66
67         public static boolean isNumeric(String input) {
68             return input != null && input.matches("\\d+");
69         }
70
71         public static boolean isValidEmail(String email) {
72             String emailRegex = "[A-Za-z0-9._%+]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}\\$";
73             Pattern pat = Pattern.compile(emailRegex);
74             return email != null && pat.matcher(email).matches();
75         }
76
77
78         private boolean isPhoneAlreadyExists(String phone, int currentUserid) {
79             try (Connection conn = DB.ConnectDB.getConnectDB()) {
80                 String sqlcheckPhone = "SELECT COUNT(*) FROM Information WHERE Phone = ? AND ID <> ?";
81                 try (PreparedStatement stmt = conn.prepareStatement(sqlcheckPhone)) {
82                     stmt.setString(1, phone);
83                     stmt.setInt(2, currentUserid);
84                     try (ResultSet rs = stmt.executeQuery()) {
85                         if (rs.next()) {
86                             int count = rs.getInt(1);
87                             return count > 0;
88                         }
89                     }
90                 } catch (SQLException e) {
91                     e.printStackTrace();
92                 }
93             }
94             return false;
95         }
96
97         private boolean isEmailAlreadyExists(String email, int currentUserid) {
98             try (Connection conn = DB.ConnectDB.getConnectDB()) {
99                 String sqlcheckEmail = "SELECT COUNT(*) FROM Information WHERE Email = ? AND ID <> ?";
100                try (PreparedStatement stmt = conn.prepareStatement(sqlcheckEmail)) {
101                    stmt.setString(1, email);
102                    stmt.setInt(2, currentUserid);
103                    try (ResultSet rs = stmt.executeQuery()) {
104                        if (rs.next()) {
105                            int count = rs.getInt(1);
106                            return count > 0;
107                        }
108                    }
109                } catch (SQLException e) {
110                    e.printStackTrace();
111                }
112            }
113            return false;
114        }

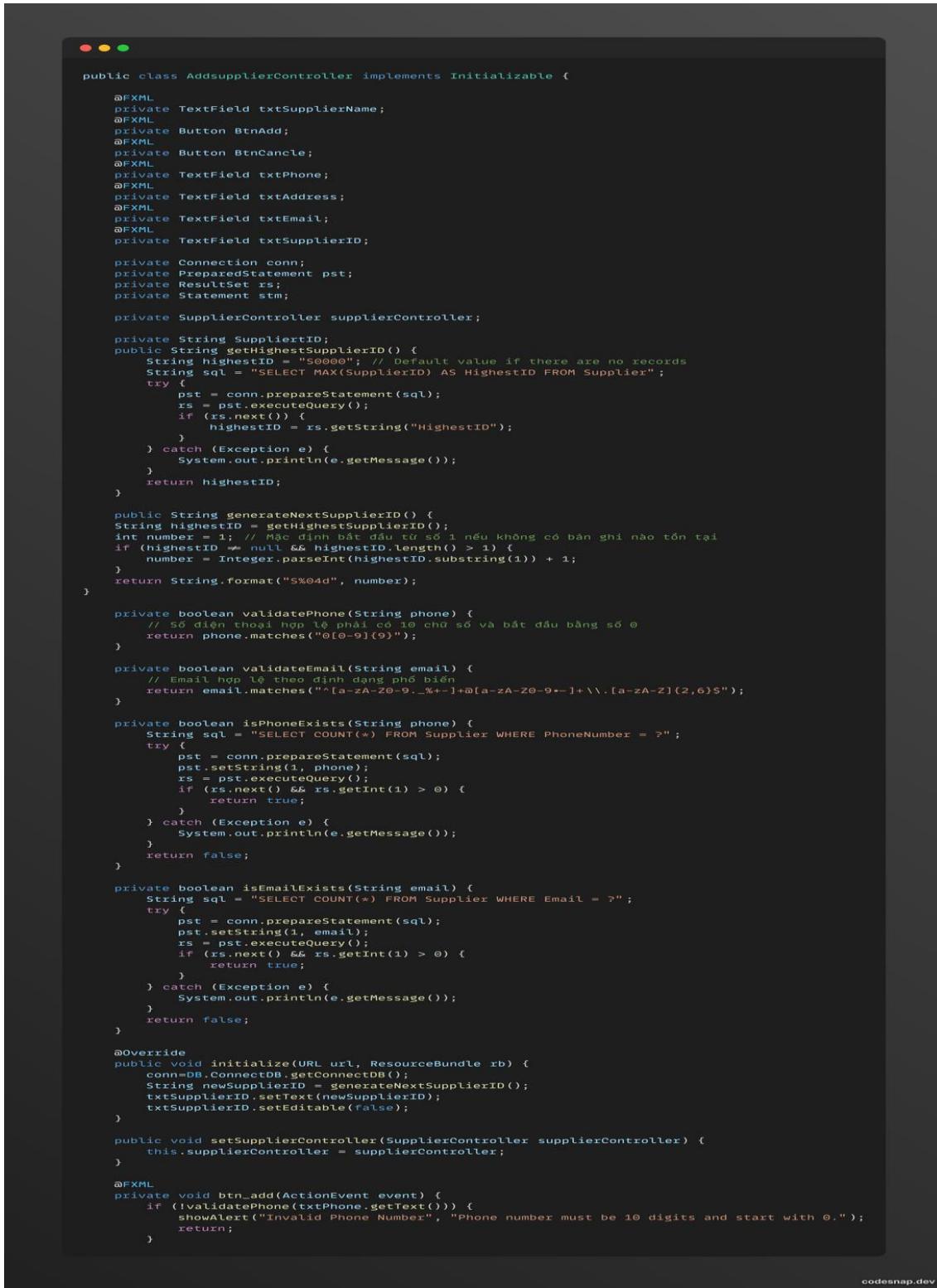
```

```

1  public void updatebtn() {
2      Integer ID = Data.al_ID;
3      if (txtrullName.getText().isEmpty()
4          || txtpassword.getText().isEmpty()
5          || txtphone.getText().isEmpty()
6          || cbRole.getValue() == null
7          || txtEmail.getText().isEmpty()) {
8          alert.errorMessage("Please fill all blank fields");
9      }
10     else if (!ValidateRegister.isNumeric(txtPhone.getText())) {
11         alert.errorMessage("Phone must be a number");
12     }
13     if (isPhoneAlreadyExists(txtPhone.getText(), ID)) {
14         alert.errorMessage("Phone already exists in the database");
15     }
16     if (!ValidateRegister.isValidEmail(txtEmail.getText())) {
17         alert.errorMessage("Email is not valid");
18     }
19     if (isEmailAlreadyExists(txtEmail.getText(), ID)) {
20         alert.errorMessage("Email already exists in the database");
21     }
22     String fullname = txtrullName.getText();
23     String address = txtAddress.getText();
24     String phone = txtPhone.getText();
25     String password = txtpassword.getText();
26     String role = cbRole.getValue();
27
28     try {
29         if (alert.confirmMessage("Are you sure want to update User: " + phone + "?")) {
30             conn = DB.ConnectDB.getConnectDB();
31             String currentRmbPassword = getRmbPassword(phone);
32
33             if (currentRmbPassword != null && !currentRmbPassword.trim().isEmpty()) {
34                 String sqlupdateLogin = "UPDATE Login SET Password = ?, Username = ? , RmbPassword = ? WHERE Username = ?";
35                 try (PreparedStatement pslogin = conn.prepareStatement(sqlupdateLogin)) {
36                     pslogin.setString1(password);
37                     pslogin.setString2(username);
38                     pslogin.setString3(password);
39                     pslogin.setString4(Data.al_phone);
40                     pslogin.executeUpdate();
41                 }
42             } else {
43                 String sqlUpdateLogin = "UPDATE Login SET Password = ?, Username = ? WHERE Username = ?";
44                 try (PreparedStatement pslogin = conn.prepareStatement(sqlUpdateLogin)) {
45                     pslogin.setString1(password);
46                     pslogin.setString2(username);
47                     pslogin.setString3(Data.al_phone);
48                     pslogin.executeUpdate();
49                 }
50                 String sqlUpdateRole = "UPDATE Role SET Role = ?, Username=? WHERE Username = ?";
51                 try (PreparedStatement psRole = conn.prepareStatement(sqlUpdateRole)) {
52                     psRole.setString1(role);
53                     psRole.setString2(phone);
54                     psRole.setString3(Data.al_phone);
55                     psRole.executeUpdate();
56                 }
57                 String sqlUpdateInformation = "UPDATE Information SET FullName = ?, Address = ?, Email = ?, Phone = ? WHERE Phone = ?";
58                 try (PreparedStatement psInfo = conn.prepareStatement(sqlUpdateInformation)) {
59                     ps = conn.prepareStatement(sqlUpdateInformation);
60                     ps.setString1(fullname);
61                     ps.setString2(address);
62                     ps.setString3(email);
63
64                     ps.setString4(phone);
65                     ps.setString5(Data.al_phone);
66
67                     ps.executeUpdate();
68                 }
69
70                 alert.successMessage("Update successfully");
71                 Stage stage = (Stage) btnupdate.getScene().getWindow();
72                 stage.close();
73                 updateDruglistForm();
74             } else {
75                 alert.errorMessage("Failed to update drug");
76             }
77         }
78     } catch (Exception e) {
79         e.printStackTrace();
80         alert.errorMessage("Error adding drug: " + e.getMessage());
81     }
82 }
83
84 private String getRmbPassword(String username) {
85     String rmbPassword = null;
86     String sqlfetchRmbpassword = "SELECT RmbPassword FROM Login WHERE Username = ?";
87     try (Connection conn = DB.ConnectDB.getConnectDB();
88         PreparedStatement ps = conn.prepareStatement(sqlfetchRmbpassword)) {
89         ps.setString1(username);
90         try (ResultSet rs = ps.executeQuery()) {
91             if (rs.next()) {
92                 rmbPassword = rs.getString("rmbPassword");
93             }
94         }
95     } catch (SQLException e) {
96         e.printStackTrace();
97     }
98     return rmbPassword;
99 }
100
101 private void updateDruglistForm() {
102     AccountInfoController accountinfoController = ControllerHolder.getInstance().getAccountinfoController();
103     if (accountinfoController != null) {
104         accountinfoController.showData();
105         accountinfoController.Pagination();
106     }
107 }
108
109
110
111 public void Rolelist() {
112     List<String> List5 = new ArrayList<>();
113     for (String data : Data.role) {
114         List5.add(data);
115     }
116     ObservableList ListData = FXCollections.observableArrayList(List5);
117     cbRole.setItems(ListData);
118 }
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137

```

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 SupplierID;
    public String getHighestSupplierID() {
        String highestID = "50000"; // 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("%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

```
1  if (!validateEmail(txtEmail.getText())) {
2      showAlert("Invalid Email Address", "Please enter a valid email address.");
3      return;
4  }
5
6  if (isPhoneExists(txtPhone.getText())) {
7      showAlert("Phone Number Exists", "The phone number already exists. Please enter a different phone number.");
8      return;
9  }
10
11 if (isEmailExists(txtEmail.getText())) {
12     showAlert("Email Address Exists", "The email address already exists. Please enter a different email address.");
13     return;
14 }
15
16 String sql = "INSERT INTO Supplier (SupplierID, SupplierName, PhoneNumber, Address, Email) VALUES (?, ?, ?, ?, ?)";
17 try {
18     pst = conn.prepareStatement(sql);
19     pst.setString(1, txtSupplierID.getText());
20     pst.setString(2, txtSupplierName.getText());
21     pst.setString(3, txtPhone.getText());
22     pst.setString(4, txtAddress.getText());
23     pst.setString(5, txtEmail.getText());
24     pst.executeUpdate();
25     System.out.println("Supplier added successfully!");
26
27     if (supplierController != null) {
28         supplierController.showSList();
29     }
30
31     Stage stage = (Stage) BtnCancle.getScene().getWindow();
32     stage.close();
33
34 } catch (Exception e) {
35     System.out.println(e.getMessage());
36 }
37 }
38
39 private void showAlert(String title, String message) {
40     Alert alert = new Alert(Alert.AlertType.ERROR);
41     alert.setTitle(title);
42     alert.setHeaderText(null);
43     alert.setContentText(message);
44     alert.showAndWait();
45 }
46
47 @FXML
48 private void btn_cancle(ActionEvent event) {
49     Stage stage = (Stage) BtnCancle.getScene().getWindow();
50     stage.close();
51 }
52
53 }
54 }
```

26 Import Order

```
1 public class ImportOrderReceiptController implements Initializable {
2
3     @FXML
4     private TextField SearchReceipt;
5     @FXML
6     private TableView<ModelReceipt> TableReceipt;
7     @FXML
8     private TableColumn<ModelReceipt, String> col_ReceiptID;
9     @FXML
10    private TableColumn<ModelReceipt, Date> col_orderDate;
11    @FXML
12    private TableColumn<ModelReceipt, Date> col_receiveDate;
13    @FXML
14    private TableColumn<ModelReceipt, String> col_supplier;
15    @FXML
16    private TableColumn<ModelReceipt, Integer> col_Total;
17    @FXML
18    private TableColumn<ModelReceipt, String> col_Status;
19    @FXML
20    private AnchorPane add_receipt;
21    @FXML
22    private AnchorPane edit_receipt;
23    @FXML
24    private AnchorPane delete_receipt;
25    @FXML
26    private Text txtMedicineName;
27    @FXML
28    private TextField Qtye01;
29    @FXML
30    private TextField Qtye02;
31    @FXML
32    private TextField Qtye03;
33    @FXML
34    private Text txtUnitPrice;
35    @FXML
36    private Button btnAddItem;
37    @FXML
38    private Button btnAddReceipt;
39    @FXML
40    private Button btnEditItem;
41    @FXML
42    private Button btnDeleteItem;
43    @FXML
44    private Text txtExpectTotal;
45    @FXML
46    private Button btnFinishOrder;
47    @FXML
48    private Button btnReceiptDelete;
49    @FXML
50    private Button btnOrder;
51    @FXML
52    private ComboBox<String> SelectMedicine;
53    @FXML
54    private ComboBox<String> SelectSupplier;
55    @FXML
56    private Text txtReceiptID;
57    @FXML
58    private TableView<ModelRDetail> TableReceiptDetail;
59    @FXML
60    private TableColumn<ModelRDetail, String> col_RMedicineID;
61    @FXML
62    private TableColumn<ModelRDetail, String> col_RMediName;
63    @FXML
64    private TableColumn<ModelRDetail, Integer> col_Qtye01;
65    @FXML
66    private TableColumn<ModelRDetail, Integer> col_Qtye02;
67    @FXML
68    private TableColumn<ModelRDetail, Integer> col_Qtye03;
69    @FXML
70    private TableColumn<ModelRDetail, Integer> col_ImportPrice;
71    @FXML
72    private Button btnPrint;
73
74    private Connection conn;
75    private PreparedStatement pst;
76    private ResultSet rs;
77    private Statement stm;
78
79    public ObservableList<ModelReceipt> ReceiptList() {
80        //goi ham
81        ObservableList<ModelReceipt> receiptList = FXCollections.observableArrayList();
82        String sql = "select * from ImportOrderReceipt inner join Supplier on ImportOrderReceipt.SupplierID=Supplier.SupplierID";
83        try {
84            pst = conn.prepareStatement(sql);
85            rs = pst.executeQuery();
86            ModelReceipt listR;
87            while (rs.next()) {
88                listR = new ModelReceipt(rs.getString("ReceiptID"),
89                                       rs.getDate("OrderDate"),
90                                       rs.getDate("ReceiveDate"),
91                                       rs.getInt("UserID"),
92                                       rs.getString("SupplierID"),
93                                       rs.getString("SupplierName"),
94                                       rs.getDouble("TotalSpend"),
95                                       rs.getString("Status")
96                );
97                receiptList.add(listR);
98            }
99        } catch (Exception e) {
100            System.out.println(e.getMessage());
101        }
102        return receiptList;
103    }
```

26.1 Code

```

1  public void showRList() {
2      ObservableList<ModelReceipt> showRList = Receiptlist();
3      col_ReceiptID.setCellValueFactory(new PropertyValueFactory<>("ReceiptID"));
4      col_orderDate.setCellValueFactory(new PropertyValueFactory<>("OrderDate"));
5      col_receiveDate.setCellValueFactory(new PropertyValueFactory<>("ReceiveDate"));
6      col_supplier.setCellValueFactory(new PropertyValueFactory<>("SupplierName"));
7      col_Total.setCellValueFactory(new PropertyValueFactory<>("TotalSpend"));
8      col_Status.setCellValueFactory(new PropertyValueFactory<>("Status"));
9      TableReceipt.setItems(showRList);
10 }
11
12 public ObservableList<ModelRDetail> DetailList(String selectReceiptID) {
13     //goi ham
14     ObservableList<ModelDetail> detailList = FXCollections.observableArrayList();
15     String sql = "select * from ReceiptDetail inner join tblProduct on ReceiptDetail.ItemID=tblProduct.MedicineID where ReceiptDetail.ReceiptID=? ";
16     try {
17         pst = conn.prepareStatement(sql);
18         pst.setString(1, selectReceiptID);
19         rs = pst.executeQuery();
20         ModelRDetail data;
21         while (rs.next()) {
22             data = new ModelRDetail(rs.getString("ReceiptID"),
23                                     rs.getString("MedicineID"),
24                                     rs.getString("MedicineName"),
25                                     rs.getInt("Qtyv01"),
26                                     rs.getInt("Qtyv02"),
27                                     rs.getInt("Qtyv03"),
28                                     rs.getInt("ITotalPrice"),
29                                     rs.getInt("CostPrice"),
30                                     rs.getInt("Quantity"),
31                                     rs.getInt("Subunitsperunit"),
32                                     rs.getInt("Itemspersubunit")
33             );
34             detailList.add(data);
35         }
36     } catch (Exception e) {
37         System.out.println(e.getMessage());
38     }
39     return detailList;
40 }
41
42 public ObservableList<ModelRDetail> DetailList() {
43     //goi ham
44     ObservableList<ModelDetail> detailList = FXCollections.observableArrayList();
45     String sql = "select * from ReceiptDetail inner join tblProduct on ReceiptDetail.ItemID=tblProduct.MedicineID ";
46     try {
47         pst = conn.prepareStatement(sql);
48         rs = pst.executeQuery();
49         ModelRDetail data;
50         while (rs.next()) {
51             data = new ModelRDetail(rs.getString("ReceiptID"),
52                                     rs.getString("MedicineID"),
53                                     rs.getString("MedicineName"),
54                                     rs.getInt("Qtyv01"),
55                                     rs.getInt("Qtyv02"),
56                                     rs.getInt("Qtyv03"),
57                                     rs.getInt("ITotalPrice"),
58                                     rs.getInt("CostPrice"),
59                                     rs.getInt("Quantity"),
60                                     rs.getInt("Subunitsperunit"),
61                                     rs.getInt("Itemspersubunit")
62             );
63             detailList.add(data);
64         }
65     } catch (Exception e) {
66         System.out.println(e.getMessage());
67     }
68     return detailList;
69 }
70
71 public void showDetail(String selectReceiptID) {
72     ObservableList<ModelDetail> showDetail = DetailList(selectReceiptID);
73     col_RMedicineID.setCellValueFactory(new PropertyValueFactory<>("MedicineID"));
74     col_RMedicineName.setCellValueFactory(new PropertyValueFactory<>("MedicineName"));
75     col_Qtyv01.setCellValueFactory(new PropertyValueFactory<>("Qtyv01"));
76     col_Qtyv02.setCellValueFactory(new PropertyValueFactory<>("Qtyv02"));
77     col_Qtyv03.setCellValueFactory(new PropertyValueFactory<>("Qtyv03"));
78     col_ImportPrice.setCellValueFactory(new PropertyValueFactory<>("ITotalPrice"));
79     TableReceiptDetail.setItems(showRDetail);
80
81     purchasedDisplayTotal();
82 }
83
84 public void selectMedicine() {
85     ObservableList<String> listMedicine = FXCollections.observableArrayList();
86     String sql = "select MedicineName from tblProduct";
87     try {
88         pst = conn.prepareStatement(sql);
89         rs = pst.executeQuery();
90         while (rs.next()) {
91             listMedicine.add(rs.getString("MedicineName"));
92         }
93         SelectMedicine.setItems(listMedicine);
94     } catch (Exception e) {
95         System.out.println(e.getMessage());
96     }
97 }
98
99 public void selectSupplier() {
100    ObservableList<String> listSupplier = FXCollections.observableArrayList();
101    String sql = "select SupplierName from Supplier";
102    try {
103        pst = conn.prepareStatement(sql);
104        rs = pst.executeQuery();
105        while (rs.next()) {
106            listSupplier.add(rs.getString("SupplierName"));
107        }
108        SelectSupplier.setItems(listSupplier);
109    } catch (Exception e) {
110        System.out.println(e.getMessage());
111    }
112 }

```

```

1  @Override
2  public void initialize(URL url, ResourceBundle rb) {
3      conn = DB.ConnectDB.getConnectDB();
4      showRList();
5      selectMedicine();
6      selectSupplier();
7      System.out.println("ok");
8  }
9
10 public ObservableList<ModelReceipt> SearchReceiptList(String keyword) {
11     ObservableList<ModelReceipt> receiptlist = FXCollections.observableArrayList();
12     String sql = "select * from ImportOrderReceipt inner join Supplier on ImportOrderReceipt.SupplierID=Supplier.SupplierID";
13     if (keyword != null && !keyword.trim().isEmpty()) {
14         sql += " where lower(ReceiptID) like ? or lower(SupplierName) like ? or lower(Status) like ?";
15     }
16     try {
17         pst = conn.prepareStatement(sql);
18         if (keyword != null && !keyword.trim().isEmpty()) {
19             String searchPattern = "%" + keyword.toLowerCase() + "%";
20             pst.setString(1, searchPattern);
21             pst.setString(2, searchPattern);
22             pst.setString(3, searchPattern);
23         }
24         rs = pst.executeQuery();
25         ModelReceipt listR;
26         while (rs.next()) {
27             listR = new ModelReceipt(rs.getString("ReceiptID"),
28                                     rs.getDate("OrderDate"),
29                                     rs.getDate("ReceiveDate"),
30                                     rs.getInt("UserID"),
31                                     rs.getString("SupplierID"),
32                                     rs.getString("SupplierName"),
33                                     rs.getInt("TotalSpend"),
34                                     rs.getString("Status")
35             );
36             receiptlist.add(listR);
37         }
38     } catch (Exception e) {
39         System.out.println(e.getMessage());
40     }
41     return receiptlist;
42 }
43
44 public void showsearchRList(String keyword) {
45     ObservableList<ModelReceipt> showsearchRList;
46     if (keyword == null || keyword.trim().isEmpty()) {
47         showsearchRList = Receiptlist; // Lấy danh sách đầy đủ nếu không có từ khóa
48     } else {
49         showsearchRList = SearchReceiptList(keyword); // Lấy danh sách dựa trên từ khóa
50     }
51
52     col_ReceiptID.setCellValueFactory(new PropertyValueFactory<>("ReceiptID"));
53     col_orderDate.setCellValueFactory(new PropertyValueFactory<>("OrderDate"));
54     col_receiveDate.setCellValueFactory(new PropertyValueFactory<>("ReceiveDate"));
55     col_supplier.setCellValueFactory(new PropertyValueFactory<>("SupplierName"));
56     col_Total.setCellValueFactory(new PropertyValueFactory<>("TotalSpend"));
57     col_Status.setCellValueFactory(new PropertyValueFactory<>("Status"));
58
59     TableReceipt.setItems(showsearchRList);
60 }
61
62 @FXML
63 public void search_receipt(ActionEvent event) {
64     SearchReceipt.textProperty().addListener((observable, oldValue, newValue) -> {
65         showsearchRList(newValue); // Gọi lại showList với từ khóa tìm kiếm mới
66     });
67 }
68
69 @FXML
70 void selectReceipt(MouseEvent event) {
71     if (event.getClickCount() == 2) { // Chỉ xử lý khi người dùng click lần
72         ModelReceipt data = TableReceipt.getSelectionModel().getSelectedItem();
73         if (data != null) {
74             txtReceiptID.setText(data.getReceiptID());
75             showRDetail(data.getReceiptID());
76         }
77     }
78 }
79
80 @FXML
81 void selectItem(MouseEvent event) {
82     if (event.getClickCount() == 2) { // Chỉ xử lý khi người dùng click lần
83         ModelReceiptDetail data = TableReceiptDetail.getSelectionModel().getSelectedItem();
84         if (data != null) {
85             txtMedicineName.setText(data.getMedicineName());
86             txtUnitPrice.setText(data.getCostPrice().toString() + " đ");
87             Qtye1.setText(data.getQtye1().toString());
88             Qtye2.setText(data.getQtye2().toString());
89             Qtye3.setText(data.getQtye3().toString());
90         }
91     }
92 }
93
94 @FXML
95 public void search(ActionEvent event) {
96 }
97
98 private String ReceiptID;
99
100 public String getHighestReceiptID() {
101     String highestID = "R0000"; // Default value if there are no records
102     String sql = "SELECT MAX(ReceiptID) AS HighestID FROM ImportOrderReceipt";
103     try {
104         pst = conn.prepareStatement(sql);
105         rs = pst.executeQuery();
106         if (rs.next()) {
107             highestID = rs.getString("HighestID");
108             if (highestID == null) {
109                 highestID = "00000";
110             }
111         }
112     } catch (Exception e) {
113         System.out.println(e.getMessage());
114     }
115     return highestID;
116 }

```



```

1  @FXML
2  public void btn_edit_item(ActionEvent event) {
3
4      try {
5          Alert alert;
6
7          if (Qty01.getText().isEmpty() || Qty02.getText().isEmpty() || Qty03.getText().isEmpty()) {
8              alert = new Alert(AlertType.ERROR);
9              alert.setTitle("Error Message");
10             alert.setHeaderText(null);
11             alert.setContentText("Please fill all blank fields");
12             alert.showAndWait();
13             return;
14         }
15
16         String checkINName = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
17         try (PreparedStatement pst = conn.prepareStatement(checkINName)) {
18             pst.setString(1, txtMedicineName.getText());
19             try (ResultSet rs = pst.executeQuery()) {
20                 if (rs.next()) {
21                     ItemID = rs.getString("MedicineID");
22                 } else {
23                     alert = new Alert(AlertType.ERROR);
24                     alert.setTitle("Error Message");
25                     alert.setHeaderText(null);
26                     alert.setContentText("Item not found");
27                     alert.showAndWait();
28                     return;
29                 }
30             }
31         }
32
33         String checkData = "SELECT * FROM tblProduct WHERE MedicinesName = ?";
34         int priceD = 0;
35         int RQty02 = 0;
36         int RQty03 = 0;
37         int OQty01 = Integer.parseInt(Qty01.getText());
38         int OQty02 = Integer.parseInt(Qty02.getText());
39         int OQty03 = Integer.parseInt(Qty03.getText());
40
41         try (PreparedStatement pst = conn.prepareStatement(checkData)) {
42             pst.setString(1, txtMedicineName.getText());
43             try (ResultSet rs = pst.executeQuery()) {
44                 if (rs.next()) {
45                     priceD = rs.getInt("CostPrice");
46                     RQty02 = rs.getInt("Subunitsperunit");
47                     RQty03 = rs.getInt("Itemspersubunit");
48                 }
49             }
50         }
51
52         if (OQty01 == 0) {
53             OQty01 = 1;
54             RQty02 = 1;
55         }
56         if (OQty02 == 0) {
57             OQty02 = 1;
58             RQty03 = 1;
59         }
60         if (OQty03 == 0) {
61             OQty03 = 1;
62         }
63
64         ITotal = priceD * RQty02 * RQty03 * OQty01 * OQty02 * OQty03;
65
66         alert = new Alert(AlertType.CONFIRMATION);
67         alert.setTitle("Confirmation Message");
68         alert.setHeaderText(null);
69         alert.setContentText("Are you sure you want to UPDATE Medicine: " + txtMedicineName.getText() + "?");
70         Optional<ButtonType> option = alert.showAndWait();
71
72         if (option.isPresent() && option.get() == ButtonType.OK) {
73             String sql = "UPDATE ReceiptDetail SET OQty01 = ?, OQty02 = ?, OQty03 = ?, ITTotalPrice = ? WHERE ReceiptID = ? AND ItemID = ?";
74             try (PreparedStatement pst = conn.prepareStatement(sql)) {
75                 pst.setString(1, Qty01.getText());
76                 pst.setString(2, Qty02.getText());
77                 pst.setString(3, Qty03.getText());
78                 pst.setInt(4, ITtotal);
79                 pst.setString(5, txtReceiptID.getText());
80                 pst.setString(6, ItemID);
81
82                 int affectedRows = pst.executeUpdate();
83                 if (affectedRows > 0) {
84                     alert = new Alert(AlertType.INFORMATION);
85                     alert.setTitle("Information Message");
86                     alert.setHeaderText(null);
87                     alert.setContentText("Successfully Updated!");
88                     alert.showAndWait();
89
90                     showRList();
91                     showDetail(txtReceiptID.getText());
92                     purchaseDisplayTotal();
93                 } else {
94                     alert = new Alert(AlertType.ERROR);
95                     alert.setTitle("Error Message");
96                     alert.setHeaderText(null);
97                     alert.setContentText("Update failed, no rows affected");
98                     alert.showAndWait();
99                 }
100            }
101        }
102    } catch (Exception e) {
103        e.printStackTrace();
104    }
105}

```

```

1  @FXML
2  void btn_delete_item(ActionEvent event) {
3      try {
4          Alert alert;
5
6          if (txtReceiptID.getText().isEmpty()) {
7              alert = new Alert(AlertType.ERROR);
8              alert.setTitle("Error Message");
9              alert.setHeaderText(null);
10             alert.setContentText("Please select an item and ensure the ReceiptID is filled");
11             alert.showAndWait();
12             return;
13         }
14
15         String checkIName = "SELECT MedicineID FROM tblProduct WHERE MedicineName = ?";
16         try (PreparedStatement pst = conn.prepareStatement(checkIName)) {
17             pst.setString(1, txtMedicineName.getText());
18             try (ResultSet rs = pst.executeQuery()) {
19                 if (rs.next()) {
20                     ItemID = rs.getString("MedicineID");
21                 } else {
22                     alert = new Alert(AlertType.ERROR);
23                     alert.setTitle("Error Message");
24                     alert.setHeaderText(null);
25                     alert.setContentText("Item not found");
26                     alert.showAndWait();
27                     return;
28                 }
29             }
30         }
31
32         alert = new Alert(AlertType.CONFIRMATION);
33         alert.setTitle("Confirmation Message");
34         alert.setHeaderText(null);
35         alert.setContentText("Are you sure you want to DELETE the selected item?");
36         Optional<ButtonType> option = alert.showAndWait();
37
38         if (option.isPresent() && option.get() == ButtonType.OK) {
39             String sql = "DELETE FROM ReceiptDetail WHERE ReceiptID = ? AND ItemID = ?";
40             try (PreparedStatement pst = conn.prepareStatement(sql)) {
41                 pst.setString(1, txtReceiptID.getText());
42                 pst.setString(2, ItemID);
43
44                 int affectedRows = pst.executeUpdate();
45                 if (affectedRows > 0) {
46                     alert = new Alert(AlertType.INFORMATION);
47                     alert.setTitle("Information Message");
48                     alert.setHeaderText(null);
49                     alert.setContentText("Successfully Deleted!");
50                     alert.showAndWait();
51
52                     showRList();
53                     showDetail(txtReceiptID.getText());
54                     purchaseDisplayTotal();
55                 } else {
56                     alert = new Alert(AlertType.ERROR);
57                     alert.setTitle("Error Message");
58                     alert.setHeaderText(null);
59                     alert.setContentText("Delete failed, no rows affected");
60                     alert.showAndWait();
61                 }
62             }
63         } catch (Exception e) {
64             e.printStackTrace();
65         }
66     }
67 }
68
69 @FXML
70 void showMedicine(MouseEvent event) {
71     if (event.getClickCount() == 1) { // Chỉ xử lý khi người dùng chọn một lần
72         SelectMedicine.getSelectionModel().selectedItemProperty().addListener((options, oldValue, newValue) -> {
73             if (newValue != null) {
74                 String sql = "SELECT * FROM tblProduct WHERE MedicineName = ?";
75                 try {
76                     pst = conn.prepareStatement(sql);
77                     pst.setString(1, newValue);
78                     rs = pst.executeQuery();
79                     if (rs.next()) {
80                         String ItemName = rs.getString("MedicineName");
81                         txtMedicineName.setText(ItemName);
82                         String ImportPrice = rs.getString("CostPrice");
83                         txtUnitPrice.setText(ImportPrice.toString() + " đ");
84                     }
85                     Qty01.setText("0");
86                     Qty02.setText("0");
87                     Qty03.setText("0");
88                 } catch (Exception e) {
89                     System.out.println(e.getMessage());
90                 }
91             }
92         });
93     }
94 }

```

```

1  @FXML
2  void showSupplier(ActionEvent event) {
3
4  }
5
6  @FXML
7  public void btn_finish(ActionEvent event) {
8    try {
9      Alert alert;
10
11      if (TableReceiptDetail.getItems().isEmpty()) {
12        alert = new Alert(AlertType.ERROR);
13        alert.setTitle("Error Message");
14        alert.setHeaderText(null);
15        alert.setContentText("This order item first");
16        alert.showAndWait();
17        return;
18      }
19
20      String checkStatus = "SELECT Status FROM ImportOrderReceipt WHERE ReceiptID = ?";
21      try (PreparedStatement pst = conn.prepareStatement(checkStatus)) {
22        pst.setString(1, txtReceiptID.getText());
23        try (ResultSet rs = pst.executeQuery()) {
24          if (rs.next()) {
25            String status = rs.getString("Status");
26            if ("Finishing".equals(status)) {
27              alert = new Alert(Alert.AlertType.ERROR);
28              alert.setTitle("Error Message");
29              alert.setHeaderText(null);
30              alert.setContentText("This Order is already finished");
31              alert.showAndWait();
32              return;
33            }
34          }
35        } catch (SQLException e) {
36          e.printStackTrace();
37          alert = new Alert(Alert.AlertType.ERROR);
38          alert.setTitle("Error Message");
39          alert.setHeaderText(null);
40          alert.setContentText("An error occurred: " + e.getMessage());
41          alert.showAndWait();
42        }
43
44        alert = new Alert(AlertType.CONFIRMATION);
45        alert.setTitle("Confirmation Message");
46        alert.setHeaderText(null);
47        alert.setContentText("Are you sure you want to finish this ORDER: " + txtReceiptID.getText() + "?");
48        Optional<ButtonType> option = alert.showAndWait();
49
50        if (option.isPresent() && option.get() == ButtonType.OK) {
51          String sql = "UPDATE ImportOrderReceipt SET ReceiveDate = ?, TotalSpend = ?, Status= ? WHERE ReceiptID = ?";
52          try (PreparedStatement pst = conn.prepareStatement(sql)) {
53            LocalDate currentDate = LocalDate.now();
54            Date sqlDate = Date.valueOf(currentDate);
55            pst.setDate(1, sqlDate);
56            pst.setString(2, txtExpectTotal.getText());
57            pst.setString(3, "Finishing");
58            pst.setString(4, txtReceiptID.getText());
59
60            int affectedRows = pst.executeUpdate();
61            if (affectedRows > 0) {
62              alert = new Alert(AlertType.INFORMATION);
63              alert.setTitle("Information Message");
64              alert.setHeaderText(null);
65              alert.setContentText("Successfully Updated!");
66              alert.showAndWait();
67
68              showRList();
69              showRDetail(txtReceiptID.getText());
70              purchaseDisplayTotal();
71            } else {
72              alert = new Alert(AlertType.ERROR);
73              alert.setTitle("Error Message");
74              alert.setHeaderText(null);
75              alert.setContentText("Update failed, no rows affected");
76              alert.showAndWait();
77            }
78          }
79        }
80      } catch (Exception e) {
81        e.printStackTrace();
82      }
83    }
84  }
85
86  @FXML
87  public void btn_Rdelete(ActionEvent event) {
88    try {
89      Alert alert;
90
91      String checkStatus = "SELECT Status FROM ImportOrderReceipt WHERE ReceiptID = ?";
92      try (PreparedStatement pst = conn.prepareStatement(checkStatus)) {
93        pst.setString(1, txtReceiptID.getText());
94        try (ResultSet rs = pst.executeQuery()) {
95          if (rs.next()) {
96            String status = rs.getString("Status");
97            if ("Finishing".equals(status)) {
98              alert = new Alert(Alert.AlertType.ERROR);
99              alert.setTitle("Error Message");
100              alert.setHeaderText(null);
101              alert.setContentText("This Order is already finished");
102              alert.showAndWait();
103              return;
104            }
105          }
106        } catch (SQLException e) {
107          e.printStackTrace();
108          alert = new Alert(Alert.AlertType.ERROR);
109          alert.setTitle("Error Message");
110          alert.setHeaderText(null);
111          alert.setContentText("An error occurred: " + e.getMessage());
112          alert.showAndWait();
113        }
114
115        alert = new Alert(AlertType.CONFIRMATION);
116        alert.setTitle("Confirmation Message");
117        alert.setHeaderText(null);
118        alert.setContentText("Are you sure you want to delete this ORDER: " + txtReceiptID.getText() + "?");
119        Optional<ButtonType> option = alert.showAndWait();
120
121      }
122    }
123  }

```

```

1  if (option.isPresent() && option.get() == ButtonType.OK) {
2      String sql1 = "DELETE FROM ReceiptDetail WHERE ReceiptID = ?";
3      String sql2 = "DELETE FROM ImportOrderReceipt WHERE ReceiptID = ?";
4
5      try (PreparedStatement pst1 = conn.prepareStatement(sql1); PreparedStatement pst2 = conn.prepareStatement(sql2)) {
6          conn.setAutoCommit(false); // Start transaction
7
8          String receiptID = txtReceiptID.getText();
9          pst1.setString(1, receiptID);
10         pst2.setString(1, receiptID);
11
12         int affectedRow1 = pst1.executeUpdate();
13         int affectedRow2 = pst2.executeUpdate();
14         if (affectedRow1 > 0 && affectedRow2 > 0) {
15             conn.commit();
16             alert.setTitle(AlertType.INFORMATION);
17             alert.setHeaderText("Information Message");
18             alert.setHeaderText(null);
19             alert.setContentText("Successfully Deleted!");
20             alert.showAndWait();
21
22             showList();
23             showDetail(txtReceiptID.getText());
24             purchaseDisplayTotal();
25
26         } else {
27             Alert alert = new Alert(AlertType.ERROR);
28             alert.setTitle("Error Message");
29             alert.setHeaderText(null);
30             alert.setContentText("Delete failed, no rows affected");
31             alert.showAndWait();
32         }
33     }
34
35     } catch (Exception e) {
36         e.printStackTrace();
37     }
38 }
39
40 @FXML
41 public void btn_order(ActionEvent event) {
42     try {
43         Alert alert;
44
45         if (TableReceiptDetail.getItems().isEmpty() || SelectSupplier.getSelectionModel().getSelectedItem() == null) {
46             alert = new Alert(AlertType.ERROR);
47             alert.setTitle("Error Message");
48             alert.setHeaderText(null);
49             alert.setContentText("Please add Item or Supplier");
50             alert.showAndWait();
51             return;
52         }
53
54         String checkSupplier = "SELECT SupplierID FROM Supplier WHERE SupplierName = ?";
55         try (PreparedStatement pst = conn.prepareStatement(checkSupplier)) {
56             pst.setString(1, SelectSupplier.getSelectionModel().getSelectedItem());
57             try (ResultSet rs = pst.executeQuery()) {
58                 if (rs.next()) {
59                     SupplierID = rs.getString("SupplierID");
60                 } else {
61                     alert = new Alert(AlertType.ERROR);
62                     alert.setTitle("Error Message");
63                     alert.setHeaderText(null);
64                     alert.setContentText("Supplier not found");
65                     alert.showAndWait();
66                     return;
67                 }
68             }
69         }
70
71         String checkStatus = "SELECT Status FROM ImportOrderReceipt WHERE ReceiptID = ?";
72         try (PreparedStatement pst = conn.prepareStatement(checkStatus)) {
73             pst.setString(1, txtReceiptID.getText());
74             try (ResultSet rs = pst.executeQuery()) {
75                 if (rs.next()) {
76                     String status = rs.getString("Status");
77                     if ("Finishing".equals(status)) {
78                         alert = new Alert(AlertType.ERROR);
79                         alert.setTitle("Error Message");
80                         alert.setHeaderText(null);
81                         alert.setContentText("This Order is already finished");
82                         alert.showAndWait();
83                         return;
84                     }
85                 }
86             } catch (SQLException e) {
87                 e.printStackTrace();
88                 alert = new Alert(AlertType.ERROR);
89                 alert.setTitle("Error Message");
90                 alert.setHeaderText(null);
91                 alert.setContentText("An error occurred: " + e.getMessage());
92                 alert.showAndWait();
93             }
94         }
95
96         alert = new Alert(AlertType.CONFIRMATION);
97         alert.setTitle("Confirmation Message");
98         alert.setHeaderText(null);
99         alert.setContentText("Are you sure you want to ORDER: " + txtReceiptID.getText() + "?");
100        OptionType option = alert.showAndWait();
101
102        if (option.isPresent() && option.get() == ButtonType.OK) {
103            String sql = "UPDATE ImportOrderReceipt SET OrderDate = ?, SupplierID = ?, TotalSpend = ?, Status= ? WHERE ReceiptID = ?";
104            try (PreparedStatement pst = conn.prepareStatement(sql)) {
105                LocalDate currentDate = LocalDate.now();
106                Date date = Date.valueOf(currentDate);
107                pst.setDate(1, date);
108                pst.setString(2, SupplierID);
109                pst.setString(3, txtImportTotal.getText());
110                pst.setString(4, "Ordering");
111                pst.setString(5, txtReceiptID.getText());
112
113                int affectedRow = pst.executeUpdate();
114                if (affectedRow > 0) {
115                    alert = new Alert(AlertType.INFORMATION);
116                    alert.setTitle("Information Message");
117                    alert.setHeaderText(null);
118                    alert.setContentText("Successfully Updated!");
119                    alert.showAndWait();
120
121                    showList();
122                    showDetail(txtReceiptID.getText());
123                    purchaseDisplayTotal();
124
125                } else {
126                    Alert alert = new Alert(AlertType.ERROR);
127                    alert.setTitle("Error Message");
128                    alert.setHeaderText(null);
129                    alert.setContentText("Update failed, no rows affected");
130                    alert.showAndWait();
131                }
132            }
133        } catch (Exception e) {
134            e.printStackTrace();
135        }
136    }
137
138    @FXML
139    public void btn_print(ActionEvent event) {
140
141
142}
143

```

27 Add Supplier

```
1  public class SupplierController implements Initializable {
2
3      @FXML
4      private TextField SearchSupplier;
5      @FXML
6      private Button btnAddSupplier;
7      @FXML
8      private TableView<ModelSupplier> TableSupplier;
9      @FXML
10     private TableColumn<ModelSupplier, String> col_SupplierID;
11    @FXML
12    private TableColumn<ModelSupplier, String> col_SupplierName;
13    @FXML
14    private TableColumn<ModelSupplier, String> col_Address;
15    @FXML
16    private TableColumn<ModelSupplier, String> col_Phone;
17    @FXML
18    private TableColumn<ModelSupplier, String> col_Email;
19
20    private Connection conn;
21    private PreparedStatement pst;
22    private ResultSet rs;
23    private Statement stm;
24
25    public ObservableList<ModelSupplier> SuppliertList(){
26        //goi ham
27        ObservableList<ModelSupplier> suppliertList = FXCollections.observableArrayList();
28        String sql = "select * from Supplier";
29        try{
30            pst = conn.prepareStatement(sql);
31            rs = pst.executeQuery();
32            ModelSupplier lists;
33            while(rs.next()){
34                lists = new ModelSupplier(rs.getString("SupplierID"),
35                                         rs.getString("SupplierName"),
36                                         rs.getString("Address"),
37                                         rs.getString("PhoneNumber"),
38                                         rs.getString("Email"))
39                );
40                suppliertList.add(lists);
41            }
42        }catch(Exception e){
43            System.out.println(e.getMessage());
44        }
45        return suppliertList;
46    }
47
48
49    public void showSList(){
50        ObservableList<ModelSupplier> showSList = SuppliertList();
51        col_SupplierID.setCellValueFactory(new PropertyValueFactory<>"SupplierID"));
52        col_SupplierName.setCellValueFactory(new PropertyValueFactory<>"SupplierName"));
53        col_Address.setCellValueFactory(new PropertyValueFactory<>"Address"));
54        col_Phone.setCellValueFactory(new PropertyValueFactory<>"PhoneNumber"));
55        col_Email.setCellValueFactory(new PropertyValueFactory<>"Email"));
56
57        TableSupplier.setItems(showSList);
58    }
59
60    @Override
61    public void initialize(URL url, ResourceBundle rb) {
62        conn=DB.ConnectDB.getConnectDB();
63        showSList();
64    }
}
```

```
1
2     @FXML
3     private void btn_add_supplier(MouseEvent event) throws IOException {
4         try {
5             FXMLLoader loader = new FXMLLoader(getClass().getResource("Addsupplier.fxml"));
6             Parent root = loader.load();
7
8             AddsupplierController controller = loader.getController();
9             controller.setSupplierController(this);
10
11            Stage stage = new Stage();
12            stage.initModality(Modality.APPLICATION_MODAL);
13            stage.setTitle("Add Supplier");
14            stage.setScene(new Scene(root));
15            stage.showAndWait();
16        } catch (Exception e) {
17            System.out.println(e.getMessage());
18        }
19    }
20
21    @FXML
22    private void selectCustomer(MouseEvent event) {
23        if (event.getClickCount() == 2) {
24            openUpdateSupplierWindow();
25        }
26    }
27
28    @FXML
29    private void openUpdateSupplierWindow() {
30        ModelSupplier selectedSupplier = TableSupplier.getSelectionModel().getSelectedItem();
31
32        try {
33            FXMLLoader loader = new FXMLLoader(getClass().getResource("Updatesupplier.fxml"));
34            Parent root = loader.load();
35
36            UpdateSupplierController controller = loader.getController();
37            controller.setSupplierController(this);
38            controller.setSupplierData(selectedSupplier.getSupplierID(),
39                                      selectedSupplier.getSupplierName(),
40                                      selectedSupplier.getPhoneNumber(),
41                                      selectedSupplier.getAddress(),
42                                      selectedSupplier.getEmail());
43
44            Stage stage = new Stage();
45            stage.initModality(Modality.APPLICATION_MODAL);
46            stage.setTitle("Update Supplier");
47            stage.setScene(new Scene(root));
48            stage.showAndWait();
49        } catch (Exception e) {
50            System.out.println(e.getMessage());
51        }
52    }
53 }
54 }
```

28 Update Supplier

```

1  public class UpdateSupplierController implements Initializable {
2
3      @FXML
4      private TextField txtSupplierName;
5      @FXML
6      private Button btnCancel;
7      @FXML
8      private TextField txtPhone;
9      @FXML
10     private TextField txtAddress;
11    @FXML
12    private TextField txtEmail;
13    @FXML
14    private Button btnUpdate;
15    @FXML
16    private Button btnDelete;
17    @FXML
18    private TextField txtSupplierID;
19
20    private Connection conn;
21    private PreparedStatement pst;
22    private ResultSet rs;
23    private SupplierController supplierController;
24    private String supplierID;
25
26    public void setSupplierController(SupplierController supplierController) {
27        this.supplierController = supplierController;
28    }
29
30    public void setSupplierData(String supplierID, String supplierName, String phone, String address, String email) {
31        this.supplierID = supplierID;
32        txtSupplierID.setText(supplierID);
33        txtSupplierID.setEditable(false);
34        txtSupplierName.setText(supplierName);
35        txtPhone.setText(phone);
36        txtAddress.setText(address);
37        txtEmail.setText(email);
38    }
39
40    @Override
41    public void initialize(URL url, ResourceBundle rb) {
42        conn=DB.ConnectDB.getConnection();
43    }
44
45    @FXML
46    private void btn_Cancle(ActionEvent event) {
47        Stage stage = (Stage) btnCancel.getScene().getWindow();
48        stage.close();
49    }
50
51    private boolean validatePhone(String phone) {
52        // Số điện thoại hợp lệ phải có 10 chữ số và bắt đầu bằng số 0
53        return phone.matches("0[0-9]{9}");
54    }
55
56    private boolean validateEmail(String email) {
57        // Email hợp lệ theo định dạng phổ biến
58        return email.matches("^[_a-zA-Z0-9-_.%+-]+@[a-zA-Z0-9-_.%+-]+\\.[a-zA-Z]{2,6}$");
59    }
60
61    private void showAlert(String title, String message) {
62        Alert alert = new Alert(AlertType.ERROR);
63        alert.setTitle(title);
64        alert.setHeaderText(null);
65        alert.setContentText(message);
66        alert.showAndWait();
67    }
68
69    @FXML
70    private void btn_update(ActionEvent event) {
71        String supplierName = txtSupplierName.getText();
72        String phone = txtPhone.getText();
73        String address = txtAddress.getText();
74        String email = txtEmail.getText();
75
76        if (!validatePhone(phone)) {
77            showAlert("Invalid Phone Number", "Phone number must be 10 digits and start with 0.");
78            return;
79        }
80        if (!validateEmail(email)) {
81            showAlert("Invalid Email Address", "Please enter a valid email address.");
82            return;
83        }
84
85        String sql = "UPDATE Supplier SET SupplierName=?, PhoneNumber=?, Address=?, Email=? WHERE SupplierID=?";
86        try {
87            pst = conn.prepareStatement(sql);
88            pst.setString(1, supplierName);
89            pst.setString(2, phone);
90            pst.setString(3, address);
91            pst.setString(4, email);
92            pst.setString(5, supplierID);
93            pst.executeUpdate();
94            System.out.println("Supplier updated successfully!");
95
96            // Cập nhật bảng trong SupplierController
97            if (supplierController != null) {
98                supplierController.showList();
99            }
100
101            // Đóng cửa sổ hiện tại
102            Stage stage = (Stage) btnUpdate.getScene().getWindow();
103            stage.close();
104        } catch (Exception e) {
105            System.out.println(e.getMessage());
106        }
107    }
108
109
110    @FXML
111    private void btn_delete(ActionEvent event) {
112        String sql = "DELETE FROM Supplier WHERE SupplierID=?";
113        try {
114            pst = conn.prepareStatement(sql);
115            pst.setString(1, supplierID);
116            pst.executeUpdate();
117            System.out.println("Supplier deleted successfully!");
118
119            // Cập nhật bảng trong SupplierController
120            if (supplierController != null) {
121                supplierController.showList();
122            }
123
124            // Đóng cửa sổ hiện tại
125            Stage stage = (Stage) btnDelete.getScene().getWindow();
126            stage.close();
127        } catch (Exception e) {
128            System.out.println(e.getMessage());
129        }
130    }
131
132 }
133

```

TASK SHEET CODING REVIEW 1, 2, 3

Project: Pharmacy			Date of preparation of activity plan:
#	Task	Prepared by	Status
1	Coding Review 1	Hoang Gia Huy	Completed
2	Coding Review 2	Hoang Gia Huy	
3	Coding Review 3	Hoang Gia Huy	
4	Document	Hoang Gia Huy & Tran Nhat Linh	
5	PowerPoint	Nguyen Anh Minh	
6	Video	Nguyen Anh Quan	
Review			Signature of instructor
			Mr. Pham Cong Danh

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		