TRƯỜNG ĐẠI HỌC CẦN THƠ
TRƯỜNG CÔNG NGHỆ THÔNG TIN & TRUYỀN THÔNG

NIÊN LUẬN CƠ SỞ
NGÀNH MẠNG MÁY TÍNH & TRUYỀN THÔNG DỮ LIỆU

KHOA MẠNG MÁY TÍNH VÀ TRUYỀN THÔNG

Đề tài

NGHIÊN CỨU XÂY DỰNG ỨNG DỤNG CHAT JAVAFX TÍCH HỢP RABBITMQ

Người hướng dẫn
TS. Ngô Bá Hùng

Sinh viên thực hiện
Nguyễn Trọng Phúc
Mã số: B1908348
Khóa: K45

Cần Thơ, 12/2022

LỜI MỞ ĐẦU

Hiện nay, lĩnh vực công nghệ thông tin ngày càng phát triển và lớn mạnh. Cùng với đó là nhu cầu sử dụng các dịch vụ trên Internet của người dùng trên các thiết bị di động, máy tính để bàn ngày một cao hơn. Đòi hỏi các lập trình viên phải có kĩ năng học tập và làm việc hiệu quả hơn, cũng như hiểu và sử dụng được công nghệ mới để xây dựng và phát triển các phần mềm có giao diện thân thiện với người dùng và hiệu năng được tối ưu. Niên luận cơ sở ngành Mạng máy tính và truyền thông dữ liệu, dưới sự giúp đỡ và hướng dẫn nhiệt tình của Thầy Ngô Bá Hùng đã giúp em hiểu biết thêm về công nghệ mới, để có một cái nhìn tổng quát hơn về lĩnh vực phát triển phần mềm, tổng hợp và vận dụng được những kiến thức đã học để xây dựng một ứng dụng hoàn thiện.

Xin chân thành cảm ơn Thầy.

MỤC LỤC

PHẦN GIỚI THIỆU

1. Đặt vấn đề.

Hệ thống mạng lưới Internet ngày càng phát triển cùng với các công nghệ tiên tiến hiện đại, đã làm cho người sử dụng các dịch vụ Internet trên ứng dụng di động, ứng dụng web, ứng dụng máy tính để bàn ngày một tăng với tốc độ nhanh. Nhu cầu người sử dụng dịch vụ vì vậy mà cũng đòi hỏi cao hơn, mong muốn có được trải nghiệm tốt hơn cùng với một giao diện thu hút và thân thiện với người dùng. Giả sử trong một hệ thống có hàng nghìn người cùng truy cập và sử dụng một dịch vụ nhắn tin trực tuyến, Server đến một lúc nào đó sẽ đạt giới hạn lượt truy cập và dẫn đến quá tải, làm giảm trải nghiệm người dùng. Một trong những giải pháp hiệu quả là sử dụng RabbitMQ.

RabbitMQ là một nhà môi giới tin nhắn mã nguồn mở phổ biến, được sử dụng tại các công ty doanh nghiệp lớn trên toàn thế giới. RabbitMQ rất nhẹ, chạy trên nhiều hệ điều hành, đám mây và cung cấp một một loạt các công cụ dành cho các nhà phát triển với hầu hết các ngôn ngữ lập trình phổ biến. Nó hỗ trợ nhiều giao thức nhắn tin. RabbitMQ có thể triển khai trong các hệ thống phân tán và đáp ứng các yêu cầu về tính sẵn dùng cao, quy mô lớn.

Để có được giao diện thu hút và thân thiện với người dùng, đề tài sử dụng JavaFX để thiết kế giao diện người dùng. JavaFX là một thư viện sử dụng ngôn ngữ Java, dùng để phát triển và phân phối các ứng dụng chạy trên máy tính để bàn và các ứng dụng chạy trên nhiều thiết bị khác nhau. JavaFX bao gồm các gói đồ họa, công cụ hỗ trợ cho người lập trình có thể tạo, kiểm tra, gỡ lỗi và triển khai ứng dụng trên nhiều loại thiết bị như: điện thoại di động, máy tính để bàn,..... Sự ra đời của JavaFX nhằm mục đích thay thế Swing cho các ứng dụng phát triển bằng ngôn ngữ Java như một khung GUI (giao diện đồ họa người dùng). JavaFX có dung lượng nhẹ, tốc độ phần cứng được gia tăng đáng kể, hỗ trợ nhiều hệ điều hành như : Window, MacOS, Linux.

Vì một số lý do đó đề tài xây dựng ứng dụng chat JavaFX với RabbitMQ nhằm minh họa cách thức hoạt động của ứng dụng giao diện đồ họa JavaFX và dịch vụ RabbitMQ để có thể ứng dụng vào việc giải quyết các vấn đề trên trong tương lai.

2. Mục tiêu đề tài.

Xây dựng giao diện đồ họa JavaFX nhằm hiểu được cách thức hoạt động của các thành phần ứng dụng, biết cách thiết kế bố cục giao diện hợp lý.

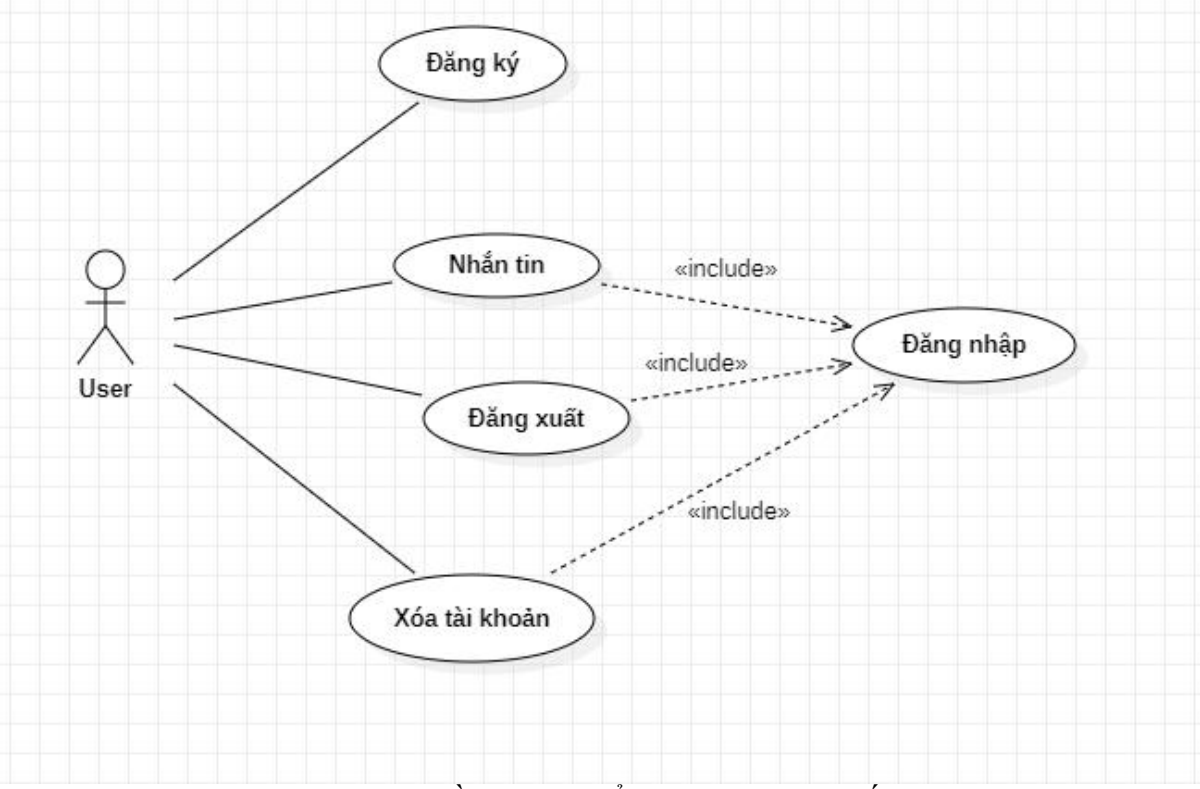
Nghiên cứu cách thức hoạt động của RabbitMQ và tích hợp vào giao diện JavaFX, nhằm xây dựng ứng dụng chat trực tuyến.

PHẦN NỘI DUNG

CHƯƠNG 1. ĐẶC TẢ YÊU CẦU

1.1. Sơ đồ use case tổng quát của ứng dụng Chat.

Sơ đồ use case tổng quát được thể hiện như hình, mô tả cái nhìn tổng quan về toàn bộ ứng dụng chat JavaFX tích hợp RabbitMQ với nhóm người dùng của hệ thống là User.



Hình 1. Sơ đồ use case tổng quát của hệ thống.

1.2. Mô tả các use case trong ứng dụng.

1.2.1. Use case “Đăng ký tài khoản”.

Chức năng “Đăng ký tài khoản” là một trong những chức năng của actor User. Nó cho phép người dùng đăng ký tài khoản. Các thông tin cụ thể về chức năng này bao gồm các kịch bản sử dụng được mô tả cụ thể ở trong bảng 1 phía bên dưới.

Ở chức năng đăng ký, người dùng buộc phải nhập đầy đủ 3 trường email, tên người dùng và mật khẩu. Nếu thiếu hệ thống sẽ thông báo yêu cầu người dùng nhập đủ. Nếu người dùng nhấn nút Back, hệ thống sẽ quay lại cảnh đăng nhập. Nếu email đã tồn tại thì hệ thống sẽ thông báo cho người dùng, nếu email chưa tồn tại và hợp lệ, hệ thống sẽ chuyển người dùng đến cảnh xác thực bằng cách gửi một mã gồm 4 chữ số, yêu

cầu người dùng nhập mã vào, nếu hợp lệ hệ thống thông báo đăng ký thành công và chuyển sang cảnh đăng nhập, ngược lại đưa người dùng về cảnh đăng ký.

Bảng 1. Mô tả use case "Đăng ký tài khoản".

Tên use case	Đăng ký tài khoản
Tóm tắt	Cho phép người xem đăng ký tài khoản.
Actor	User
Kịch bản thường	1. Người dùng vào giao diện đăng nhập và chọn đăng ký tài khoản. 2. Hệ thống hiển thị giao diện đăng ký cho người dùng. 3. Người dùng nhập thông tin email, username và mật khẩu để đăng ký tài khoản. 4. Người dùng chọn nút đăng ký. 5. Hệ thống gửi một mã xác thực có bốn chữ số đến email người dùng vừa nhập. 6. Người dùng nhập đúng mã xác thực. 7. Người dụng chọn nút Submit. 8. Hệ thống thông báo thành công và đưa người dùng đến màn hình đăng nhập.
Kịch bản thay thế	A1 - Người dùng nhập email không đúng định dạng. Chuỗi A1 bắt đầu ở bước 4 của kịch bản thường. 5. Hiển thị thông báo cho biết thông tin người dùng nhập sai định dạng. Quay về bước 2 trong kịch bản thường. A2 - Tài khoản người dùng đăng kí đã được tạo. Chuỗi A2 bắt đầu ở bước 4 của kịch bản thường. 5. Hiển thị thông báo cho người dùng biết tài khoản đã được tạo. Quay về bước 2 trong kịch bản thường. A3 - Người dùng nhập sai mã xác thực. Chuỗi A2 bắt đầu ở bước 7 của kịch bản thường. 8. Hiển thị thông báo cho người dùng biết sai mã xác thực. 9. Quay lại bước 2 trong kịch bản thường.
Kết quả	Người dùng đăng ký tài khoản thành công.

1.2.2. Use case “Đăng nhập tài khoản”.

Chức năng “Đăng nhập tài khoản” là một trong những chức năng của actor User. Nó cho phép người dùng đăng ký tài khoản. Các thông tin cụ thể về chức năng này bao gồm các kịch bản sử dụng được mô tả cụ thể ở trong bảng 2 phía bên dưới.

Ở chức năng đăng nhập, người dùng nhập vào email và mật khẩu, nếu nhấn vào nút clear sẽ xóa email và mật khẩu trong trường nhập. Nếu nhấn nút login, hệ thống sẽ kiểm tra nếu email là hợp lệ và đúng mật khẩu trong cơ sở dữ liệu, hệ thống sẽ chuyển người dùng đến màn hình chính, đồng thời tạo ra một queue là tên người dùng đã đăng ký vào RabbitMQ management. Nếu người dùng nhập không đủ thông tin email hoặc mật khẩu, hệ thống sẽ thông báo yêu cầu người dùng nhập đầy đủ thông tin. Nếu email và mật khẩu không hợp lệ hệ thống sẽ thông báo sai email hoặc mật khẩu và xóa email, mật khẩu trong trường nhập. Nếu người dùng chưa có tài khoản thì nhấn vào nút Register, hệ thống sẽ chuyển người dùng đến giao diện đăng ký.

Bảng 2. Mô tả use case “Đăng nhập tài khoản”.

Tên use case	Đăng nhập tài khoản.
Tóm tắt	Cho phép người dùng đăng nhập tài khoản.
Actor	User
Kịch bản thường	1. Người dùng vào giao diện đăng nhập. 2. Người dùng nhập thông tin email và mật khẩu. 3. Người dùng chọn nút đăng nhập. 4. Hệ thống đưa người dùng vào giao diện nhắn tin.
Kịch bản thay thế	A1 - Người dùng nhập sai email hoặc mật khẩu, Chuỗi A1 bắt đầu ở bước 3 của kịch bản thường. 4. Hiển thị thông báo cho biết thông tin người dùng nhập sai. 5. Quay lại bước 1 của kịch bản thường.
Kết quả	Người dùng đăng nhập tài khoản thành công.

1.2.3. Use case “Nhắn tin”.

Chức năng “Nhắn tin” là một trong những chức năng của actor User. Nó cho phép người dùng nhắn tin với những người dùng khác trong hệ thống. Các thông tin cụ thể về chức năng này bao gồm các kịch bản sử dụng được mô tả cụ thể ở trong bảng 3 phía bên dưới.

Ở chức năng nhắn tin, hệ thống hiển thị danh sách những người dùng có trong cơ sở dữ liệu. Người dùng hiện hành có thể nhập tên người muốn gửi tin nhắn hoặc nhấn vào tên người trong danh sách, sau đó nhập nội dung muốn gửi và ấn nút Send để gửi tin nhắn lên queue của RabbitMQ và hiển thị tin nhắn lên màn hình. Khi có tin nhắn từ người dùng khác, hệ thống sẽ nhận tin nhắn từ queue của RabbitMQ và hiển thị lên màn hình. Nếu người dùng nhấn nút Reload hệ thống sẽ tải lại những người dùng có trong cơ sở dữ liệu và hiển thị danh sách người dùng. Nếu người dùng nhấn nút Logout, hệ thống sẽ hiển thị thông báo xác nhận, người dùng nhấn nút OK thì hệ thống đăng xuất tài khoản và đưa người dùng về màn hình đăng nhập.

Bảng 3. Mô tả use case “Nhắn tin”.

Tên use case	Nhắn tin.
Tóm tắt	Cho phép người dùng nhắn tin với những người dùng khác trong hệ thống.
Actor	User.
Kịch bản thường	1. Hiển thị giao diện nhắn tin. 2. Người dùng nhập tên người nhận hoặc chọn vào tên người nhận trên danh sách người dùng. 3. Người dùng nhập tin nhắn. 4. Người dùng nhấn nút gửi. 5. Hệ thống gửi tin nhắn cho người kia và hiển thị tin nhắn lên màn hình.
Kịch bản thay thế	Không có kịch bản thay thế.
Kết quả	Người dùng gửi và nhận tin nhắn thành công.

1.3. Các thành phần trong JavaFX.

1.3.1. Stage (Sân khấu).

Giai đoạn (một cửa sổ) chứa tất cả các đối tượng của ứng dụng JavaFX. Nó được đại diện bởi lớp Stage của gói javafx.stage. Stage chính được tạo bởi chính nền tảng của nó. Đối tượng stage đã tạo sẽ được truyền như một đối số cho phương thức start() của lớp Application.

Mỗi sân khấu có hai tham số xác định vị trí của nó là Width và Height. Nó được chia thành Content Area (Khu vực nội dung) và Decoration (Trang trí).

Có 5 loại sân khấu có sẵn : trang trí, chưa trang trí, trong suốt, thống nhất, tiện ích.

Phải gọi phương thức show() để hiển thị nội dung của một vùng.

1.3.2. Scene (Bối cảnh).

Một bối cảnh tượng trưng cho nội dung vật lý của một ứng dụng JavaFX. Nó chứa tất cả các nội dung của một biểu đồ cảnh. Lớp Scene của gói javafx.scene đại diện cho đối tượng scene. Tại một thể hiện, đối tượng scene chỉ được thêm vào một giai đoạn.

Tạo một cảnh bằng cách khởi tạo lớp Scene, có thể chọn cách thước của cảnh bằng cách truyền kích thước của nó (chiều cao và chiều rộng) cùng với nút gốc tới hàm tạo của nó.

1.3.3. Scene graph và Node.

Một đồ thị cảnh là một cây giống như cấu trúc dữ liệu (thứ bậc) thể hiện nội dung của một cảnh. Ngược lại, một nút là một đối tượng trực quan/ đồ họa của biểu đồ cảnh.

Một nút có thể bao gồm: các đối tượng hình học, điều khiển giao diện người dùng, các vùng chứa, các phần tử media.

Các node của gói javafx.scene đại diện cho một nút trong JavaFX, lớp này là siêu lớp của tất cả các nút.

Một nút có 3 loại :

+ Nút gốc : đồ thị cảnh đầu tiên được gọi là nút gốc.

+ Nút nhánh/Nút cha : nút có các nút con được gọi là nút nhánh/nút cha. Lớp trừu tượng có tên là Parent của gói javafx.scene là lớp cơ sở của tất cả các nút cha và các nút cha đó sẽ thuộc các loại sau:

* Nhóm : nút nhóm là một nút tập hợp có chứa danh sách các nút con. Bất cứ khi nào nút nhóm được hiển thị, tất cả các nút con của nó được hiển thị theo thứ tự. Mọi biến

đổi, trạng thái hiệu ứng được áp dụng trên nhóm sẽ được áp dụng cho tất cả các nút con.

* Vùng : đây là lớp cơ sở của tất cả các điều khiển giao diện người dùng dựa trên Node JavaFX, chẳng hạn như biểu đồ, ngăn và điều khiển.

* WebView : nút này quản lý công cụ web và hiển thị nội dung của nó.

+ Nút lá : nút không có nút con được gọi là nút lá. Ví dụ như Rectangle, Ellipse, Box, ImageView, Media View là các ví dụ về các nút lá.

Bắt buộc phải chuyển nút gốc vào đồ thị cảnh. Nếu nhóm được chuyên làm gốc, tất cả các nút sẽ được cắt bớt vào cảnh và bất kỳ thay đổi nào về kích thước của cảnh sẽ không ảnh hưởng đến bố cục của cảnh.

1.4. Công nghệ sử dụng.

1.4.1. Các khái niệm trong RabbitMQ.

Producer : Ứng dụng gửi tin nhắn
Consumer : Ứng dụng nhận tin nhắn
Queue : Bộ đệm lưu trữ tin nhắn
Message : Thông tin được gửi từ nhà sản xuất đến người tiêu dùng thông qua RabbitMQ
Connection : Một kết nối TCP giữa ứng dụng của bạn và message broker (RabbitMQ)
Channel : Một kết nối ảo bên trong 1 Connection. Khi publishing hoặc consuming messages từ một queue --> tất cả được thực hiện trên một kênh (Channel).
Exchange : Nhận message từ Producer và đẩy chúng vào hàng đợi tùy thuộc vào các quy tắc được xác định bởi loại exchange. Để nhận được message thì một queue phải được binding với ít nhất một exchange.
Binding : Là liên kết giữa queue và exchange
Key routing : Một khóa mà bên exchange xem xét để quyết định các định tuyến thông điệp đến queue. Coi key routing như là địa chỉ cho thư.
AMQP : Giao thức xếp hàng thông điệp nâng cao - là giao thức được Rabbitmq sử dụng để nhắn tin.
Users : Có thể kết nối với Rabbitmq bằng tên người dùng và mật khẩu đã cho. Mọi người đều có thể được chỉ định các quyền như đọc, ghi và cấu hình các đặc quyền cho instance. Người dùng cũng có thể được chỉ định cho các máy chủ ảo cụ thể. (virtual host).
Vhost, virtualhost : cung cấp cách tách biệt các ứng dụng bằng cách sử dụng một instance Rabbitmq. Những người dùng khác nhau đối với các vhost khác nhau và có thể tạo queue và exchange, vì vậy chúng chỉ tồn tại trong một vhost.

1.4.2. Các loại Exchange.

Direct : message được chuyển đến hàng đợi khi có binding key trùng với routing key
Fanout : message được định tuyến đến tất cả các queue liên kết với nó

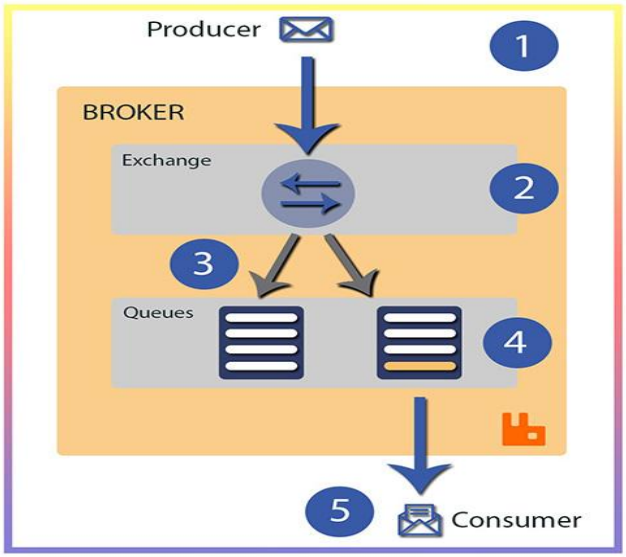
Topic : làm một wildcard (kí tự đại diện) để gắn routing key với một routing pattern được khai báo trong binding. Consumer có thể đăng ký những topic mà nó quan tâm. Cú pháp được sử dụng ở đây là * và #.

Header : Một header exchange sẽ dùng các thuộc tính header của message để định tuyến. Header exchange rất giống với Topic exchange, nhưng nó định tuyến dựa vào các giá trị tiêu đề thay vì các key routing. Một thông điệp được coi là phù hợp nếu giá trị của tiêu đề (header) bằng với giá trị được chỉ định khi ràng buộc.

1.4.3. Cách thức hoạt động của RabbitMQ.

Hình 2. Quy trình làm việc đơn giản của RabbitMQ theo mô hình Publish/Subscribe.

Kiến trúc cơ bản của hàng đợi tin nhắn là khi có các ứng dụng khách được gọi là nhà sản xuất (Producer) tạo tin nhắn và gửi chúng đến nhà môi giới tin nhắn (hàng đợi tin nhắn). Các ứng dụng khác được gọi là người tiêu dùng, kết nối với hàng đợi và đăng ký các tin nhắn sẽ được xử lý. Phần mềm có thể đóng vai trò là nhà sản xuất hoặc người tiêu dùng hoặc cả người sản xuất và người tiêu dùng. Tin nhắn được đặt vào hàng đợi và được lưu trữ cho đến khi người tiêu dùng truy xuất chúng.



Hình 3. Quy trình làm việc đầy đủ của RabbitMQ theo mô hình Publish/Subscribe.

(1) Producer đẩy message vào Exchange. Khi tạo Exchange, phải mô tả nó thuộc loại gì.

(2) Sau khi Exchange nhận Message, nó chịu trách nhiệm định tuyến message. Exchange sẽ chịu trách nhiệm về các thuộc tính của Message, ví dụ routing key, loại Exchange.

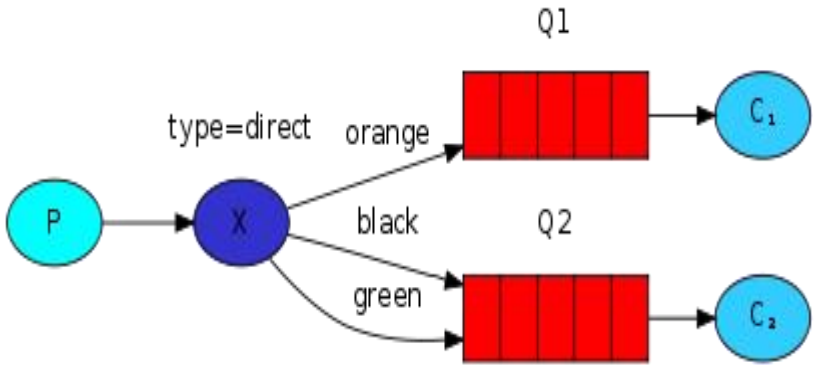
(3) Việc binding phải được tạo từ Exchange đến Queue (hàng đợi). Trong trường hợp này, có hai binding đến hai hàng đợi khác nhau từ một Exchange. Exchange sẽ định tuyến Message vào các hàng đợi dựa trên thuộc tính của của từng Message.

(4) Các Message nằm ở hàng đợi đến khi chúng được xử lý bởi một Consumer.

(5) Consumer xử lý Message nhận từ Queue.

1.4.4. Hoạt động bên trong RabbitMQ thông qua từng loại Exchange.

1.4.4.1. Direct exchange.



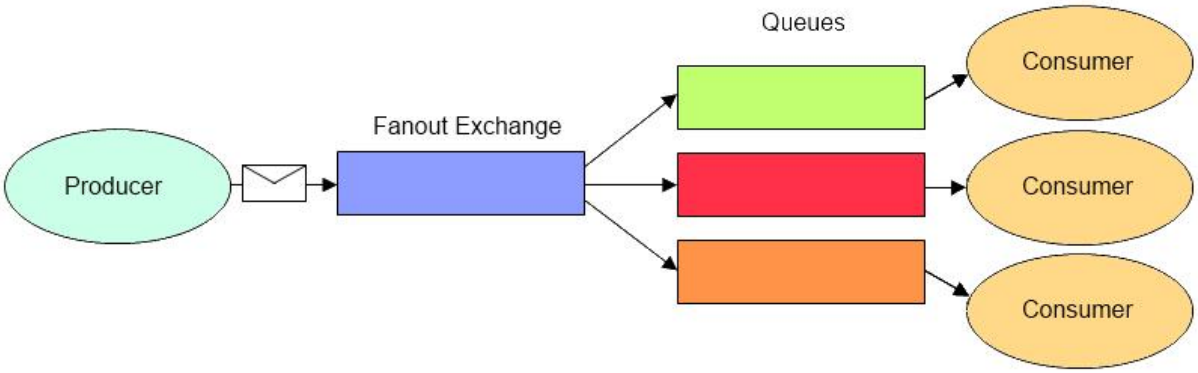
Hình 4. Hoạt động của RabbitMQ thông qua direct exchange.

Direct exchange (trao đổi trực tiếp) định tuyến message đến Queue dựa vào routing key. Một queue được ràng buộc với một direct exchange bởi một routing key K. Khi có một message mới với routing key R đến direct exchange. Message sẽ được chuyển tới queue đó nếu R=K.

Direct exchange hữu ích khi muốn phân biệt các thông báo được publish cho cùng một exchange bằng cách sử dụng một mã định danh chuỗi đơn giản.

Ở ví dụ hình 3, producer P gửi message đến exchange X và queue Q1 gắn với exchange có binding key là orange, message được đẩy vào exchange với routing key là orange thì sẽ được đưa vào hàng đợi này, sau đó consumer C1 nhận message từ hàng đợi đã đăng ký. Tương tự, queue Q2 gắn với exchange có binding key là black và green, message được đẩy vào exchange với routing key là black và green thì sẽ được đẩy vào hàng đợi này, sau đó consumer C2 nhận message từ hàng đợi đã đăng ký.

1.4.4.2. Fanout exchange.



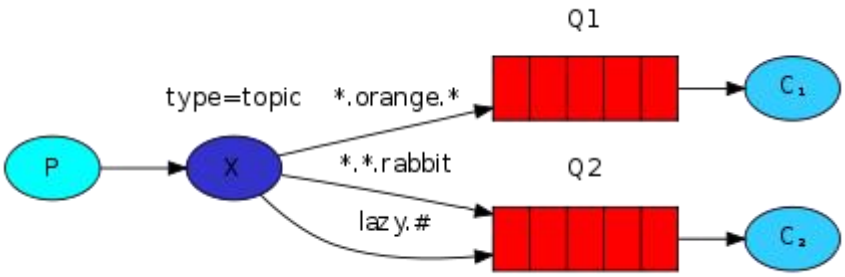
Hình 5. Hoạt động của RabbitMQ thông qua fanout exchange.

Fanout exchange định tuyến message tới tất cả queue mà nó được bind, với bất kể một routing key nào.

Fanout exchange hữu ích với trường hợp ta cần một dữ liệu được gửi tới nhiều ứng dụng khác nhau với cùng một message nhưng cách xử lý ở các ứng dụng là khác nhau.

Ở ví dụ hình 4, Producer gửi message đến fanout exchange, tất cả các queues binding với exchange này đều nhận được message và đưa vào queue. Sau đó các consumer nhận các message từ hàng đợi đã đăng ký tương ứng.

1.4.4.3. Topic exchange.



Hình 6. Hoạt động của RabbitMQ thông qua topic exchange.

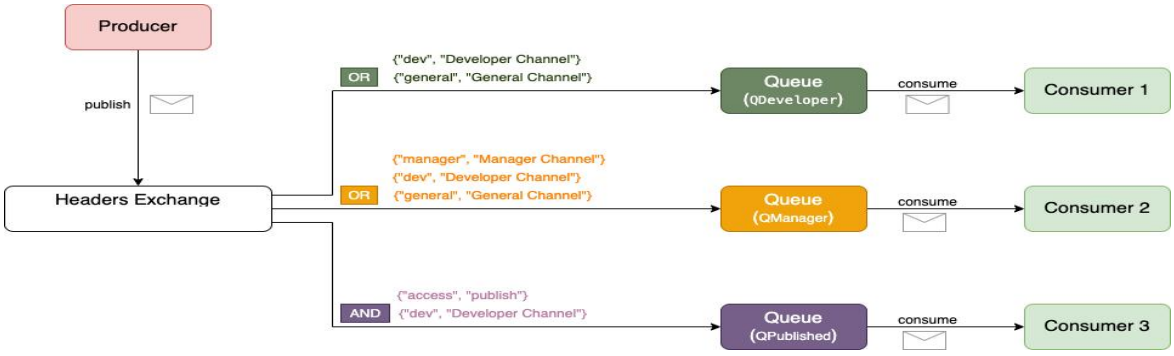
Topic exchange định tuyến message tới một hoặc nhiều queue dựa trên sự trùng khớp giữa routing key và pattern, nó phải là một danh sách các từ, được phân tách bằng dấu chấm.

Topic exchange thường sử dụng để phân phối dữ liệu liên quan đến vị trí địa lý cụ thể, xử lý tác vụ nền được thực hiện bởi nhiều workers, mỗi công việc có khả năng xử lý các nhóm tác vụ cụ thể, cập nhật tin tức liên quan đến một từ khóa hoặc gắn tag.

Trong ví dụ hình 5, producer P gửi message đến topic exchange, queue Q1 binding đến exchange X và nhận các message có binding key bắt đầu bằng từ bất kỳ, kế tiếp là orange và kết thúc bằng từ bất kỳ. Tương tự, queue Q2 binding đến exchange X và

nhận các message hoặc bắt đầu bằng từ bất kỳ và kết thúc bằng từ rabbitmq hoặc chỉ cần bắt đầu với từ lazy sẽ được đưa đến hàng đợi Q2. Sau đó các consumer sẽ nhân các message ở các queue đã đăng ký.

1.4.4.4. Header exchange.



Hình 7. Hoạt động của RabbitMQ thông qua header exchange.

Header exchange được thiết kế để định tuyến với nhiều thuộc tính, để dễ dàng thực hiện dưới dạng header của message hơn là routing key. Header exchange bỏ đi routing key mà thay vào đó định tuyến dựa trên header của message. Trường hợp này, message broker cần quan tâm đến những tin nhắn với tiêu đề nào phù hợp với chúng.

Headers exchange rất giống với Topic Exchange, nhưng nó định tuyến dựa trên các giá trị header thay vì routing key.

Luồng hoạt động của một message trong header exchange như sau :

- Một hoặc nhiều queue được tạo và binding tới một header exchange sử dụng các header property (H).
- Một producer sẽ tạo một message với các header property (MH) và publish tới exchange.
- Một message được exchange chuyển đến queue nếu header H match với header MH.
- Consumer nhận message ở queue đã đăng kí tương ứng.

Có 2 loại matching được sử dụng để kiểm tra một Header của binding queue có match với một header từ message đến :

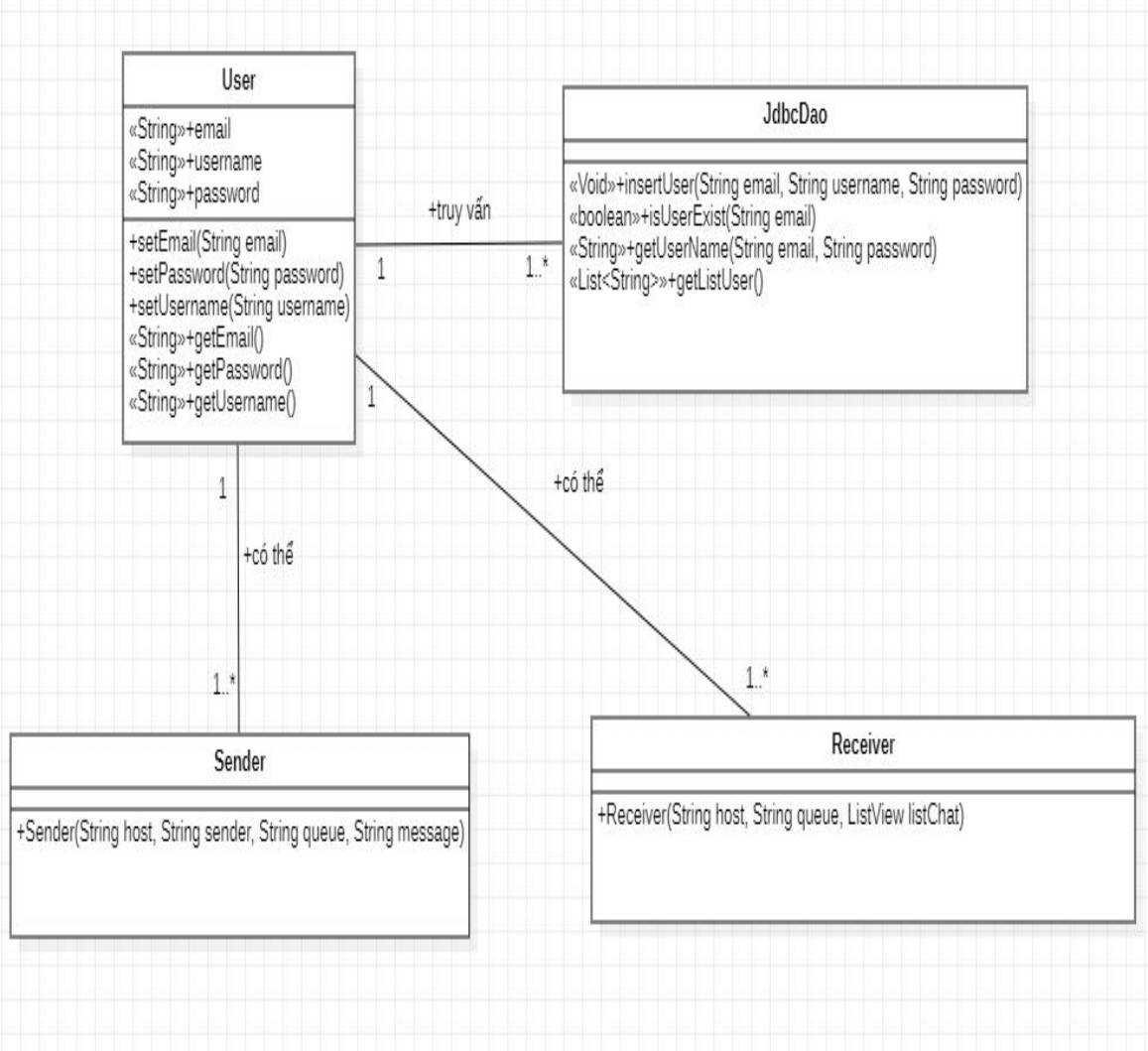
+ any: tương tự như toán tử OR, được biểu diễn trong các ràng buộc header property là {“x-match“, “any“, …} . Nghĩa là, một Message được gửi tới exchange phải chứa ít nhất một trong các header màqueue được liên kết, sau đó message sẽ được chuyển đến queue.

+ all: tương tự như toán tử AND, được biểu diễn trong các ràng buộc header property là {“x-match“, “and“, …} . Nghĩa là, các message có tất cả các header được liệt kê của nó sẽ được chuyển tiếp đến queue.

Trong ví dụ hình 6, queue QDeveloper sẽ nhận tất cả các message có header là {“dev”, “Developer Channel”} hoặc {“general”, “General Channel”}. Queue QManager sẽ nhận tấ cả các message có header là {“manager”, “Manager Channel”} hoặc {“dev”, “Developer Channel”} hoặc {“general”, “General Channel”}. Queue QPublished sẽ nhận tất cả các message có header là {“dev”, “Developer Channel”} và {“access”, “publish”}. Sau đó các consumer sẽ nhận các message ở các queue đã đăng ký tương ứng.

CHƯƠNG 2. THIẾT KẾ GIẢI PHÁP

2.1. Sơ đồ lớp của ứng dụng Chat.

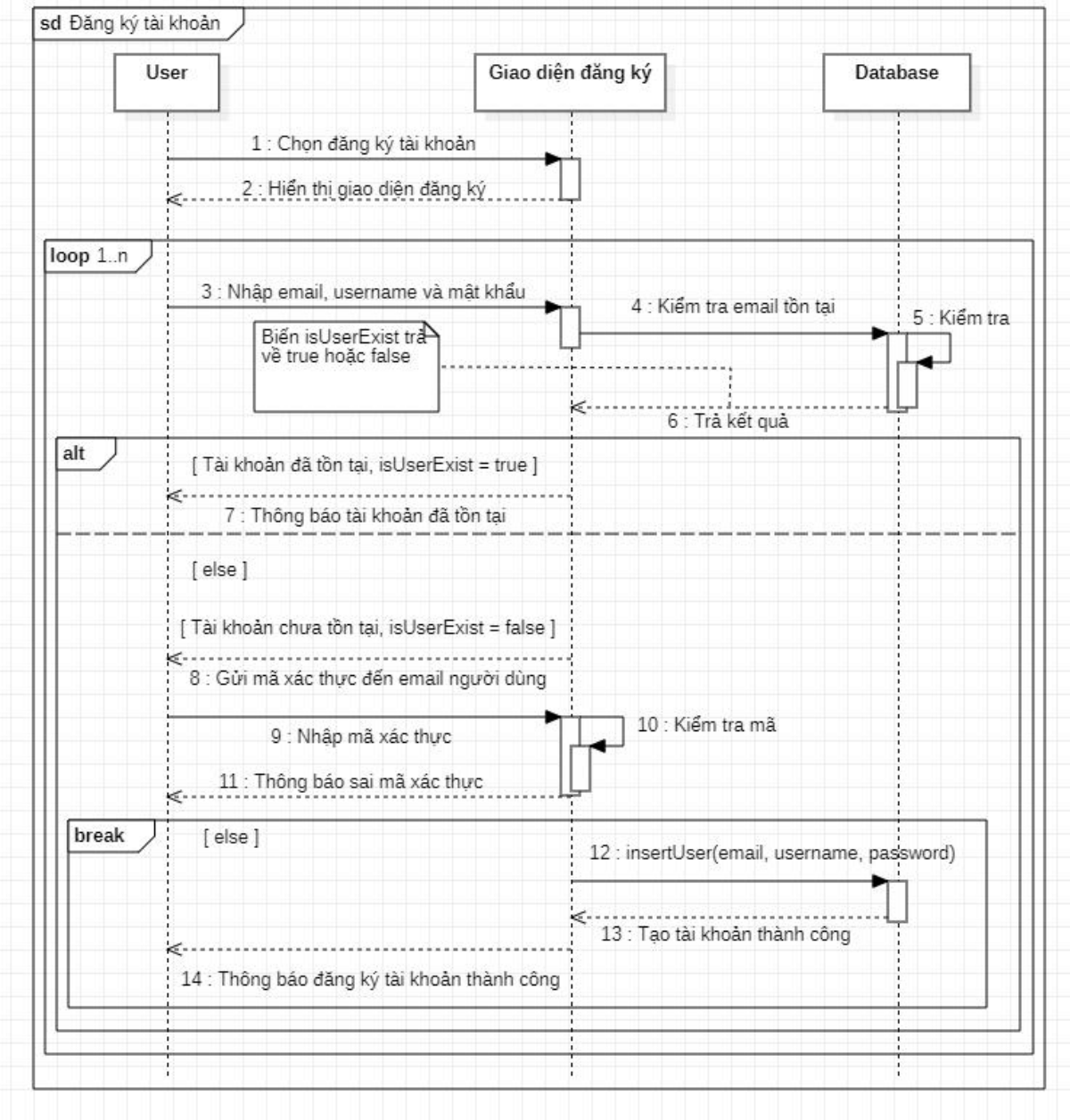


Hình 8. Sơ đồ lớp của ứng dụng Chat.

2.2. Sơ đồ tuần tự của ứng dụng Chat.

2.2.1. Sơ đồ tuần tự "Đăng ký tài khoản".

Chức năng "Đăng ký tài khoản" là một trong những chức năng của actor User. Sơ đồ tuần tự của chức năng này thì được mô tả như trong Hình 9 bên dưới.



Hình 9. Sơ đồ tuần tự của chức năng "Đăng ký tài khoản".

Mô tả chức năng: Cho phép người xem đăng ký tài khoản.

Điều kiện tiên quyết: Không có.

Trình tự thực hiện:

1. Người dùng vào giao diện chính và chọn đăng ký tài khoản.
2. Hệ thống hiển thị giao diện đăng ký cho người dùng.
3. Người dùng nhập thông tin email, username và mật khẩu để đăng kí tài khoản.
4. Hệ thống gọi phương thức để kiểm tra email.
5. Hệ thống kiểm tra email đã tồn tại chưa.
6. Trả kết quả kiểm tra. [Ngoại lệ 1]
7. Kết quả true, thông báo tài khoản tồn tại.

Ngoại lệ 1:

8. Hệ thống gửi mã xác thực đến email vừa nhập.
9. Người dùng nhập mã xác thực.
10. Hệ thống kiểm tra mã xác thực. [Ngoại lệ 2]
11. Kết quả false, hệ thống thông báo sai mã xác thực.

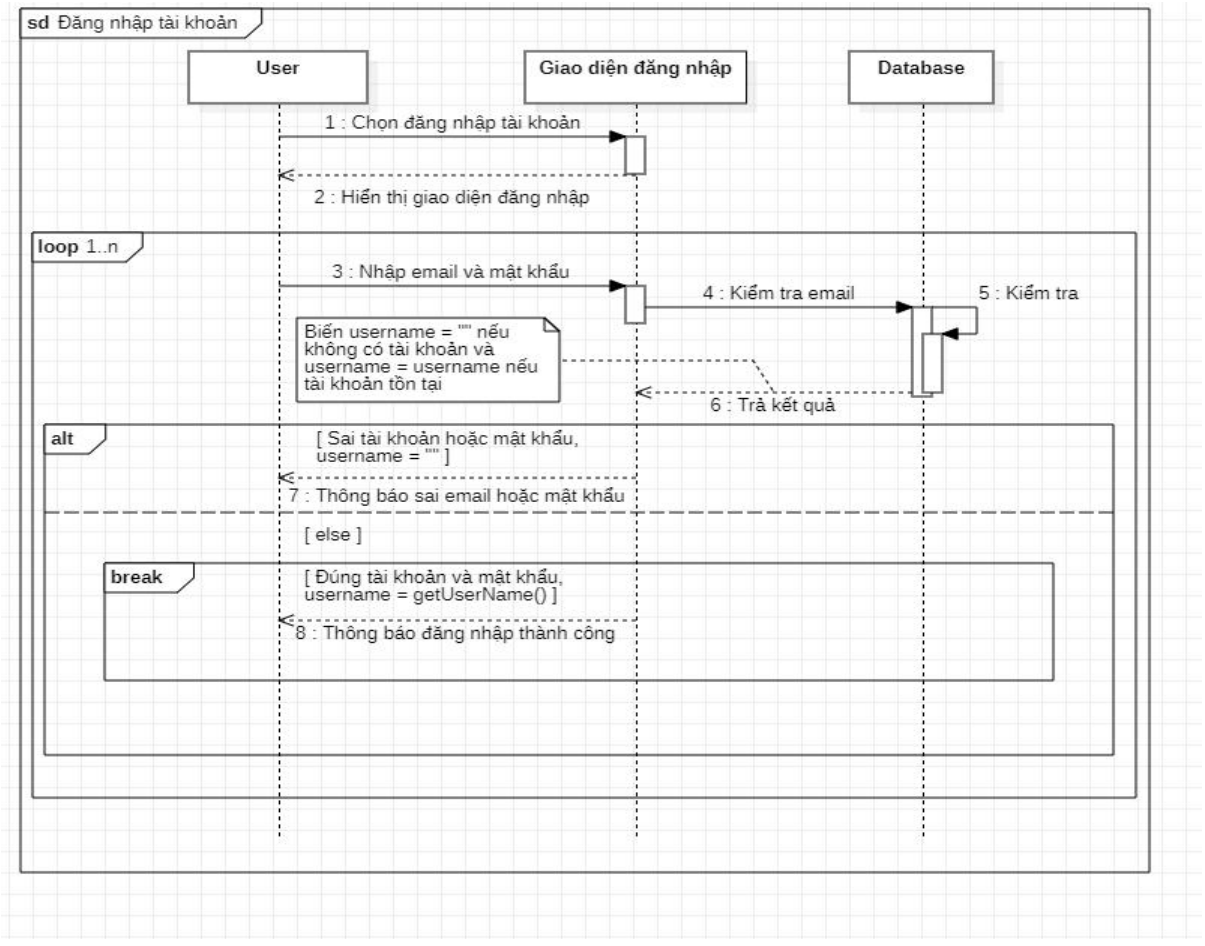
Ngoại lệ 2:

12. Kết quả true, hệ thống gọi phương thức insert(email, username, password) để đăng ký tài khoản.
13. Tạo tài khoản thành công.
14. Thông báo tạo tài khoản thành công

Kết thúc.

2.2.2. Sơ đồ tuần tự "Đăng nhập tài khoản".

Chức năng "Đăng nhập tài khoản" là một trong những chức năng của actor User. Sơ đồ tuần tự của chức năng này được mô tả như trong Hình 10 bên dưới.



Hình 10. Sơ đồ tuần tự của chức năng "Đăng nhập tài khoản".

Mô tả chức năng : cho phép người xem đăng nhập tài khoản.

Điều kiện tiên quyết: Đã đăng ký tài khoản thành công.

Trình tự thực hiện :

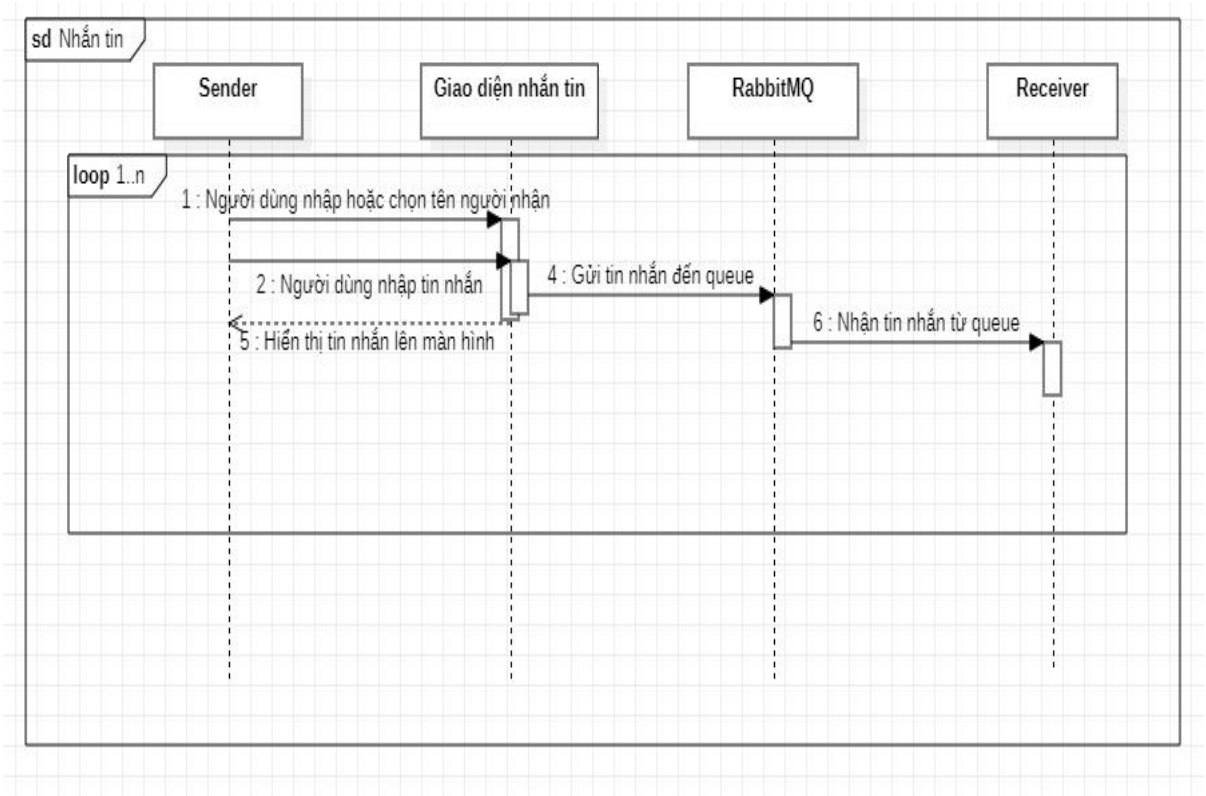
1. Người dùng vào giao diện chính và chọn đăng nhập.
2. Hệ thống hiển thị giao diện đăng nhập cho người dùng.
3. Người dùng nhập vào email và mật khẩu.
4. Hệ thống kiểm tra email.
5. Hệ thống gọi phương thức getUserName(email).
6. Trả kết quả kiểm tra. [Ngoại lệ]
7. Kết quả là chuỗi rỗng, hệ thống thông báo sai email hoặc mật khẩu.

Ngoại lệ:

8. Kết quả là chuỗi tên người dùng, hệ thống thông báo đăng nhập thành công.

2.2.3. Sơ đồ tuần tự “Nhắn tin”.

Chức năng “Nhắn tin” là một trong những chức năng của actor User. Sơ đồ tuần tự của chức năng này được mô tả như trong Hình 11 bên dưới.



Hình 11. Sơ đồ tuần tự của chức năng “Nhắn tin”.

2.3. Cách thức lưu trữ dữ liệu.

2.3.1. Hệ quản trị cơ sở dữ liệu MySQL.

MySQL là hệ quản trị cơ sở dữ liệu tự do nguồn mở phổ biến nhất thế giới và được các nhà phát triển rất ưa chuộng trong quá trình phát triển ứng dụng. Vì MySQL là hệ quản trị cơ sở dữ liệu tốc độ cao, ổn định và dễ sử dụng, có tính khả chuyển, hoạt động trên nhiều hệ điều hành cung cấp một hệ thống lớn các hàm tiện ích rất mạnh. Với tốc độ và tính bảo mật cao, MySQL rất thích hợp cho các ứng dụng có truy cập CSDL trên Internet.

Ứng dụng Chat JavaFX tích hợp RabbitMQ sử dụng hệ quản trị cơ sở dữ liệu MySQL để lưu trữ dữ liệu tài khoản người dùng.

2.3.2. Sử dụng JDBC để truy vấn dữ liệu từ cơ sở dữ liệu.

Java JDBC là một Java API được sử dụng để kết nối và thực hiện truy vấn với cơ sở dữ liệu. JDBC API sử dụng trình điều khiển jdbc để kết nối với cơ sở dữ liệu.

Ứng dụng Chat JavaFX tích hợp RabbitMQ sử dụng JDBC để kết nối với hệ quản trị cơ sở dữ liệu MySQL và truy vấn dữ liệu.

2.4. Cách thức gửi mail xác thực.

Jakarta Mail (trước đây là JavaMail) là API Jakarta EE được sử dụng để gửi và nhận email qua SMTP , POP3 và IMAP . Jakarta Mail được tích hợp vào nền tảng Java EE , nhưng cũng cung cấp một gói tùy chọn để sử dụng trong Java SE.

Ứng dụng JavaFX tích hợp RabbitMQ sử dụng Jakarta Mail để gửi mail xác thực đến người dùng.

CHƯƠNG 3. CÀI ĐẶT GIẢI PHÁP

3.1. Cài đặt chức năng "Đăng ký tài khoản".

3.1.1. Cài đặt giao diện đăng ký.

Giao diện đăng ký gồm 3 trường nhập : email, username và password. Giao diện đơn giản, các background được lấy từ url trên Internet.

```
 * layout register
 * */
TextField tfEmailRegis = new TextField();
tfEmailRegis.setStyle("-fx-focus-color: blue;" +
        "-fx-border-width: 1px;" +
        "-fx-corner-radius: 2px");
tfEmailRegis.setPromptText("Enter your email");
tfEmailRegis.setMaxSize( v: 200,  v1: 100);
TextField tfUsernameRegis = new TextField();
tfUsernameRegis.setStyle("-fx-focus-color: blue;" +
        "-fx-border-width: 1px;" +
        "-fx-corner-radius: 2px");
tfUsernameRegis.setPromptText("Enter your username");
tfUsernameRegis.setMaxSize( v: 200,  v1: 100);
PasswordField tfPasswordRegis = new PasswordField();
tfPasswordRegis.setStyle("-fx-focus-color: blue;" +
        "-fx-border-width: 1px;" +
        "-fx-corner-radius: 2px");
tfPasswordRegis.setPromptText("Enter your password");
tfPasswordRegis.setMaxSize( v: 200,  v1: 100);
Button btnRegister = new Button( s: "Register");
btnRegister.setEffect(new DropShadow());
btnRegister.setStyle("-fx-background-color: #0014FF");
btnRegister.setTextFill(Color.WHITE);
Button btnBack = new Button( s: "Back");
btnBack.setEffect(new DropShadow());
btnBack.setStyle("-fx-background-color: #0014FF");
btnBack.setTextFill(Color.WHITE);
HBox hBoxReBack = new HBox();
hBoxReBack.setAlignment(Pos.CENTER);
hBoxReBack.getChildren().addAll(btnRegister, btnBack);
hBoxReBack.setSpacing(10);
layoutRegister.getChildren().addAll(tfEmailRegis, tfUsernameRegis, tfPasswordRegis, hBoxReBack);
layoutRegister.setAlignment(Pos.CENTER);
layoutRegister.setSpacing(10);
layoutRegister.setMaxWidth(400);
layoutRegister.setMaxHeight(300);
layoutRegister.setStyle("-fx-background-image: url(https://img.freepik.com/free-vector/talk-show" +
        "-studio-interior_1284-9411.jpg);" +
        "-fx-background-repeat: no-repeat;" +
        "-fx-background-position: center center;");
```

Hình 12. Đoạn mã tạo giao diện đăng ký.

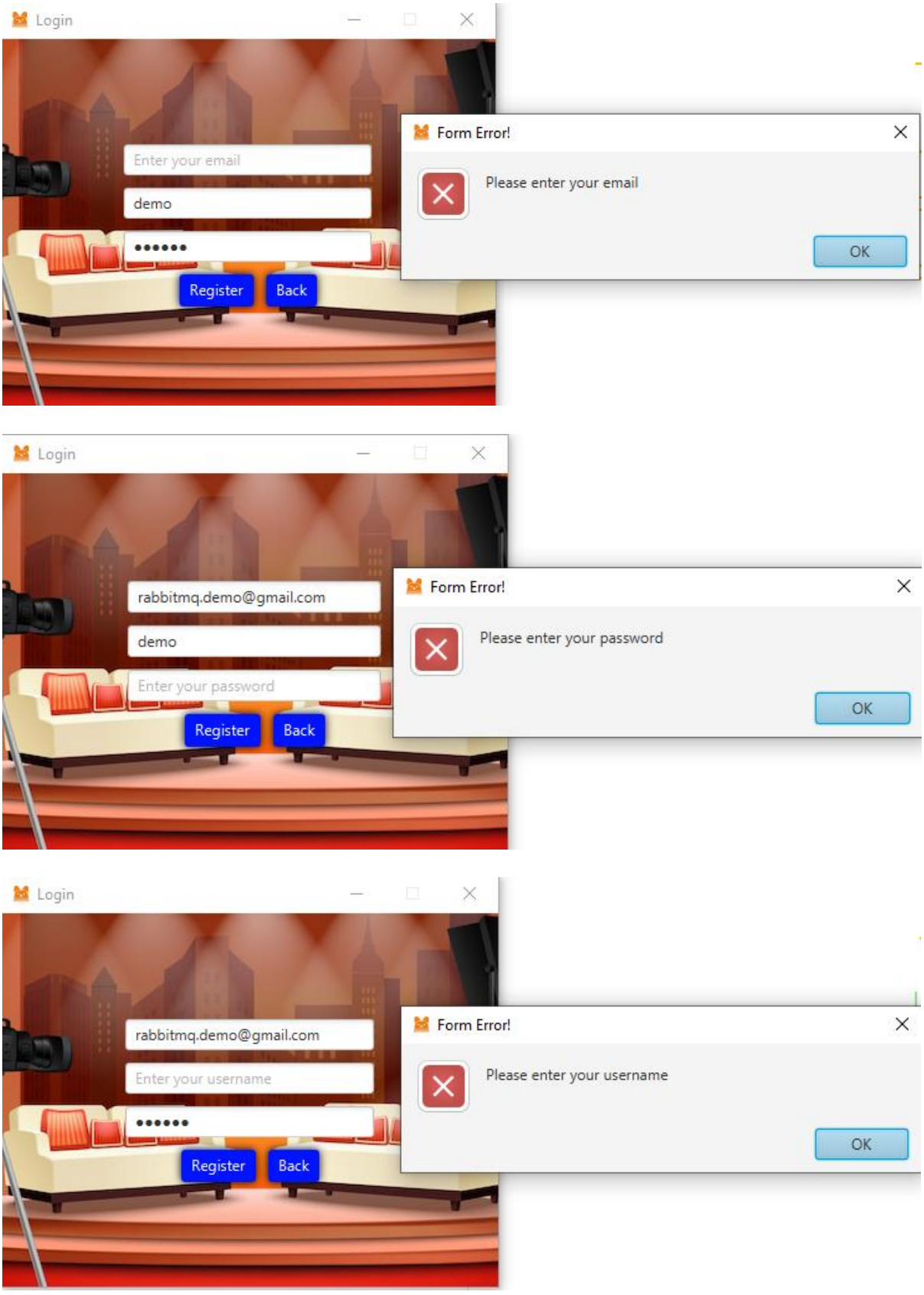
Hình 13. Kết quả của giao diện đăng ký.

3.1.2. Xử lý các sự kiện khi nhấn nút Register.

3.1.2.1. Các sự kiện người dùng nhập thiếu hoặc sai thông tin.

```
1 usage   PhucEnterdev *
private void onClickButtonRegister(Button btnRegister, TextField tfEmailRegis, TextField tfUsernameRegis,
                                   TextField tfPasswordRegis, Stage stageMain, Scene sceneLogin) {
    btnRegister.setOnAction(actionEvent -> {

        Window owner = btnRegister.getScene().getWindow();

        if (tfEmailRegis.getText().isEmpty()) {
            showAlert(Alert.AlertType.ERROR, owner,  title: "Form Error!",
                    notification: "Please enter your email");
            return;
        }

        if (tfUsernameRegis.getText().isEmpty()) {
            showAlert(Alert.AlertType.ERROR, owner,  title: "Form Error!",
                    notification: "Please enter your username");
            return;
        }

        if (tfPasswordRegis.getText().isEmpty()) {
            showAlert(Alert.AlertType.ERROR, owner,  title: "Form Error!",
                    notification: "Please enter your password");
            return;
        }
```

Hình 14. Đoạn mã thông báo các sự kiện cho người dùng biết khi nhấn nút Register.



Hình 15. Kết quả của các sự kiện thiếu hoặc sai thông tin sau khi nhấn nút Register.

3.1.2.2. Các sự kiện người dùng nhập đủ thông tin.

3.1.2.2.1. Tài khoản đã tồn tại.

```
if(isEmailValid(tfEmailRegis.getText())){
    String emailRegister = tfEmailRegis.getText();
    String usernameRegister = tfUsernameRegis.getText();
    String passwordRegister = tfPasswordRegis.getText();


    JdbcDao jdbcDao = new JdbcDao();
    boolean isUserExist = jdbcDao.isUserExist(emailRegister);
    if (isUserExist) {
        showAlert(Alert.AlertType.ERROR, owner,  title: "Account is existed",
                notification: "Account is existed!");

        tfEmailRegis.clear();
        tfUsernameRegis.clear();
        tfPasswordRegis.clear();
        stageMain.setScene(sceneRegister);
```

Hình 16. Đoạn mã xử lý khi tài khoản đã tồn tại.

Sử dụng lớp JdbcDao để kiểm tra tài khoản có tồn tại hay không. Nếu email là hợp lệ và đã tồn tại trong cơ sở dữ liệu thì thông báo cho người dùng.

```
1 usage   PhucEnterdev
public boolean isUserExist(String email){
    boolean isUserExist = false;
    // Establish a connection and auto close the connection
    try (Connection connection = DriverManager
            .getConnection(DATABASE_URL, DB_USERNAME, DB_PASSWORD);

         // Create a statement using connection object
         PreparedStatement preparedStatement = connection.prepareStatement(SELECT_CHECK_USER_QUERY)) {
        preparedStatement.setString( parameterIndex: 1, email);

        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()){
            isUserExist = true;
        }
    } catch (SQLException sqlEx) {
        System.out.println("Loi truy van");
    }
    return isUserExist;
}
```

Hình 17. Đoạn mã kiểm tra email đã tồn tại trong cơ sở dữ liệu của lớp JdbcDao.

3.1.2.2.2. Tài khoản chưa tồn tại.

```
} else {
    /** Authentication email **/
    new*
    Platform.runLater(new Runnable() {
        new*
        @Override
        public void run() { EmailUtil.checkEmail(emailRegister); }
    });

    stageMain.setScene(sceneAuthentication);
```

Hình 18. Đoạn mã xử lý khi tài khoản chưa tồn tại..

3.1.3. Xử lý gửi mail xác thực người dùng.

3.1.3.1. Tạo và kiểm tra mã xác thực.

```
1 usage
static int random = (int) ((Math.random()*(9999-1000))+1000);
2 usages
static String codeRandom = String.valueOf(random);
2 usages   ⚠ PhucEnterdev
public static boolean checkCode(String code){
    boolean isValid = false;
    if (code.equals(codeRandom)){
        isValid = true;
    }
    return isValid;
}
```

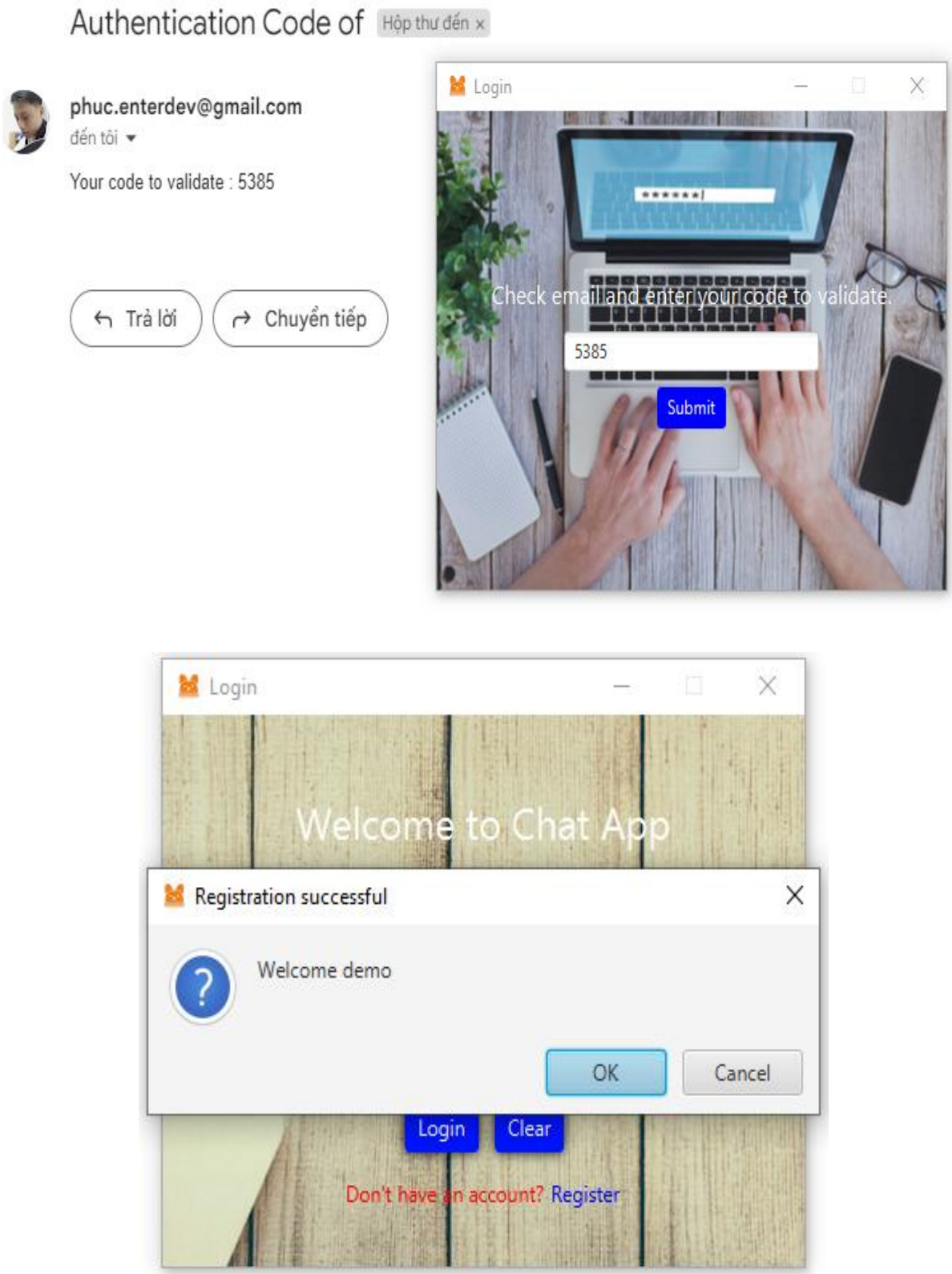
Hình 19. Đoạn mã tạo mã xác thực và kiểm tra.

3.1.3.2. Gửi mã xác thực đến email của người dùng.

Sử dụng thư viện Jakarta mail của gói jakarta.mail để gửi mã xác thực đến emaill của người dùng.

```
1 usage   PhucEnterdev *
public static void checkEmail(String emailToSend) {
    final String username = "phuc.enterdev@gmail.com";
    final String password = "asetugiqbzetumni";

    Properties prop = new Properties();
    prop.put("mail.smtp.host", "smtp.gmail.com");
    prop.put("mail.smtp.port", "587");
    prop.put("mail.smtp.auth", "true");
    prop.put("mail.smtp.starttls.enable", "true"); //TLS

    Session session = Session.getInstance(prop,
            PhucEnterdev
            new Authenticator() {
                no usages   PhucEnterdev
                @Override
                protected jakarta.mail.PasswordAuthentication getPasswordAuthentication() {
                    return new jakarta.mail.PasswordAuthentication(username, password);
                }
            });
    try {

        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress("phuc.enterdev@gmail.com"));
        message.setRecipients(
                Message.RecipientType.TO,
                InternetAddress.parse(emailToSend)
        );
        message.setSubject("Authentication Code of ");
        message.setText("Your code to validate : "+codeRandom);

        Platform.runLater(() -> {
            try {
                Transport.send(message);
            } catch (MessagingException e) {
                throw new RuntimeException(e);
            }
        });

        System.out.println("Done");
```

Hình 20. Đoạn mã gửi mail xác thực người dùng.



Hình 21. Kết quả hiển thị khi xác thực thành công.

3.2. Cài đặt chức năng “Đăng nhập tài khoản”.

3.2.1. Cài đặt giao diện đăng nhập.

```
private void setLayoutLogin(Stage stageMain, HBox hBoxRegister) {
    // Login
    labelWelcome.setFont(Font.font( v: 24));
    labelWelcome.setTextFill(Color.WHITE);
    labelWelcome.setPadding(new Insets( v: 10,  v1: 10,  v2: 50,  v3: 10));

    tfEmailLogin.setMaxSize( v: 200,  v1: 100);
    tfEmailLogin.setStyle("-fx-focus-color: blue;" +
            "-fx-border-width: 1px;" +
            "-fx-corner-radius: 2px");
    tfEmailLogin.setPromptText("Enter your email");

    tfPasswordLogin.setMaxSize( v: 200,  v1: 100);
    tfPasswordLogin.setStyle("-fx-focus-color: blue;" +
            "-fx-border-width: 1px;" +
            "-fx-corner-radius: 2px");
    tfPasswordLogin.setPromptText("Enter your password");

    HBox hBoxLogin = new HBox();
    btnLogin.setStyle("-fx-background-color: #0014FF");
    btnLogin.setTextFill(Color.WHITE);
    btnLogin.setEffect(new DropShadow());
    btnClear.setStyle("-fx-background-color: #0014FF");
    btnClear.setTextFill(Color.WHITE);
    btnClear.setEffect(new DropShadow());
    hBoxLogin.setAlignment(Pos.CENTER);
    hBoxLogin.getChildren().addAll(btnLogin, btnClear);
    hBoxLogin.setPadding(new Insets( v: 5,  v1: 5,  v2: 5,  v3: 5));
    hBoxLogin.setSpacing(10);

    layoutLogin.setStyle("-fx-background-image: url(https://images.unsplash.com/photo-1432821596592-e2c" +
            "18b78144f?ixlib=rb-4.0.3&ixid=Mnw" +
            "xMjA3fDB8MHxzZWFyY2h8M3x8bG9naW58ZW58MHx8MHx8&w=1000&q=80);" +
            "-fx-background-repeat: stretch;" +
            "-fx-background-position: bottom center;");
    layoutLogin.getChildren().addAll(labelWelcome, tfEmailLogin, tfPasswordLogin, hBoxLogin, hBoxRegister);
    layoutLogin.setAlignment(Pos.CENTER);
    layoutLogin.setSpacing(10);
    layoutLogin.setMaxWidth(400);
    layoutLogin.setMaxHeight(300);
```

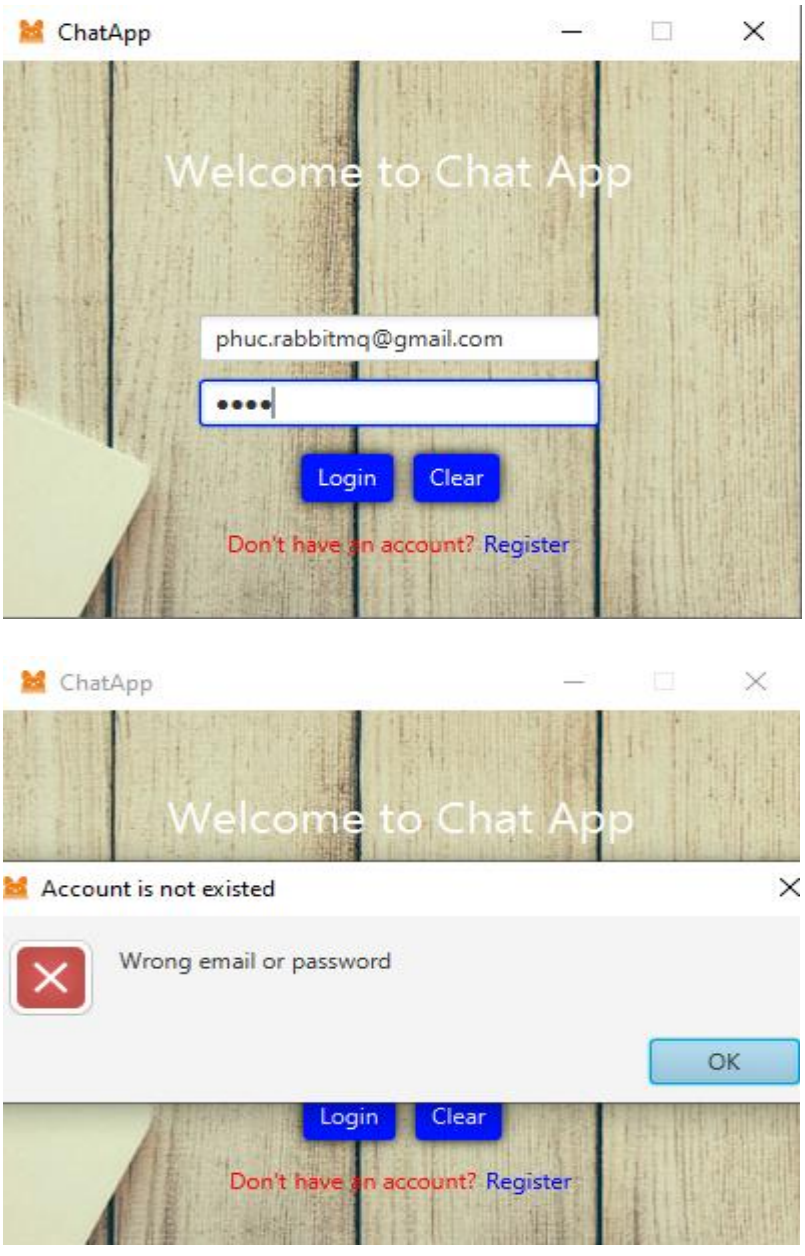
Hình 22. Đoạn mã tạo giao diện đăng nhập.

3.2.2. Xử lý các sự kiện khi nhấn nút Login.

3.2.2.1. Sai email hoặc mật khẩu.

```
}else {
    showAlert(Alert.AlertType.ERROR, tfEmail.getScene().getWindow() ,
            title: "Account is not existed",  notification: "Wrong email or password");
    tfEmail.clear();
    tfPassword.clear();
    stageMain.setScene(sceneLogin);
}
return username;
```

Hình 23. Đoạn mã kiểm tra email hoặc password sai.



Hình 24. Kết quả sự kiện sai email hoặc mật khẩu khi nhấn nút Login.

3.2.2.2. Đúng email và mật khẩu.

```
// event button login
1 usage   PhucEnterdev
private void onClickButtonLogin(Button btnLogin, Stage stageMain) {
    btnLogin.setOnAction(event -> {
        if (!tfEmailLogin.getText().isEmpty()) {
            tfReceiver.clear();
            stageMain.setTitle("ChatApp");
            String userName = setLayoutMain(stageMain, sceneMain, tfEmailLogin, tfPasswordLogin);
            Receiver receive = new Receiver(host, userName, listChat);
        }
    });
}
```

Hình 25. Đoạn mã kiểm tra đúng email và password.

3.3. Cài đặt chức năng “Nhắn tin”.

3.3.1. Cài đặt giao diện nhắn tin.

```
// top layout main
HBox hBoxTop = new HBox();
Label labelUserName = new Label( s: "Welcome " + username);
labelUserName.setTextFill(Color.WHITE);
labelUserName.setFont(Font.font( v: 24));
hBoxTop.getChildren().add(labelUserName);
hBoxTop.setAlignment(Pos.CENTER);
hBoxTop.setPadding(new Insets( v: 5,  v1: 5,  v2: 5,  v3: 5));
layoutMain.setTop(hBoxTop);


// left layout main
VBox vBoxListUser = new VBox();
vBoxListUser.setMaxHeight(1000);
vBoxListUser.setAlignment(Pos.CENTER_LEFT);
vBoxListUser.setPadding(new Insets( v: 5, v1: 10, v2: 5, v3: 5));
vBoxListUser.setSpacing(5);
Label lbListUser = new Label( s: "List Users");
lbListUser.setTextFill(Color.WHITE);
lbListUser.setFont(Font.font( v: 20));
Button btnReload = new Button( s: "Reload");
btnReload.setStyle("-fx-background-color: #0014FF");
btnReload.setTextFill(Color.WHITE);
btnReload.setEffect(new DropShadow());
btnReload.setOnAction(actionEvent -> {
    List<String> listUsers = jdbcDao.getListUser();
    lsUsers.getItems().clear();
    for (int i = 0; i <= listUsers.size() - 1 ; i++) {
        lsUsers.getItems().add(listUsers.get(i));
        lsUsers.refresh();
    }
});
HBox hBox = new HBox();
hBox.setSpacing(5);
hBox.getChildren().addAll(lbListUser, btnReload);
```

```
lsUsers.setMaxSize( v: 150,  v1: 1000);

List<String> listUsers = jdbcDao.getListUser();
lsUsers.getItems().clear();
for (int i = 0; i <= listUsers.size() - 1 ; i++) {
    lsUsers.getItems().add(listUsers.get(i));
    lsUsers.refresh();
}

vBoxListUser.getChildren().addAll(hBox, lsUsers);
layoutMain.setLeft(vBoxListUser);
layoutMain.setStyle("-fx-background-color: #009EFF");


// center layout main
listChat.setMaxSize( v: 500,  v1: 600);
layoutMain.setCenter(listChat);

// bottom layout main
HBox hBoxBottom = new HBox();
hBoxBottom.setAlignment(Pos.CENTER);

tfReceiver.setPromptText("Receiver name");
tfReceiver.setStyle("-fx-focus-color: blue;" +
        "-fx-border-width: 1px;" +
        "-fx-corner-radius: 2px");
tfReceiver.setMaxWidth(100);
tfReceiver.setMaxHeight(100);

tfMessage.setPromptText("Enter your message");
tfMessage.setStyle("-fx-focus-color: blue;" +
        "-fx-border-width: 1px;" +
        "-fx-corner-radius: 2px");
tfMessage.setMaxWidth(200);
tfMessage.setMaxHeight(100);
```

```
tfMessage.setPromptText("Enter your message");
tfMessage.setStyle("-fx-focus-color: blue;" +
        "-fx-border-width: 1px;" +
        "-fx-corner-radius: 2px");
tfMessage.setMaxWidth(200);
tfMessage.setMaxHeight(100);

btnSend.setStyle("-fx-background-color: #0014FF");
btnSend.setTextFill(Color.WHITE);
btnSend.setEffect(new DropShadow());
btnLogout.setStyle("-fx-background-color: #0014FF");
btnLogout.setTextFill(Color.WHITE);
btnLogout.setEffect(new DropShadow());
btnDeleteAccount.setStyle("-fx-background-color: #0014FF");
btnDeleteAccount.setTextFill(Color.WHITE);
btnDeleteAccount.setEffect(new DropShadow());
hBoxBottom.getChildren().addAll(tfMessage, tfReceiver, btnSend, btnLogout, btnDeleteAccount);
hBoxBottom.setSpacing(20);
hBoxBottom.setPadding(new Insets( v: 10,  v1: 10,  v2: 10,  v3: 10));
layoutMain.setBottom(hBoxBottom);
stageMain.setScene(sceneMain);
```

Hình 26. Đoạn mã tạo giao diện nhắn tin.

3.3.2. Xử lý sự kiện gửi và nhận tin nhắn.

3.3.2.1. Gửi tin nhắn.

```
3 usages  PhucEnterdev
public class Sender {

    1 usage  PhucEnterdev
    public Sender(String host, String sender, String queue, String message) {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost(host);
        factory.setConnectionTimeout(30000);
        try (Connection connection = factory.newConnection();
             Channel channel = connection.createChannel()) {
            channel.queueDeclare(queue, b: false, b1: false, b2: false, map: null);
            channel.basicPublish( s: "", queue, basicProperties: null, (sender+">>>"+message).getBytes(StandardCharsets.UTF_8));
        } catch (IOException | TimeoutException e) {
            throw new RuntimeException(e);
        }
    }
}
```

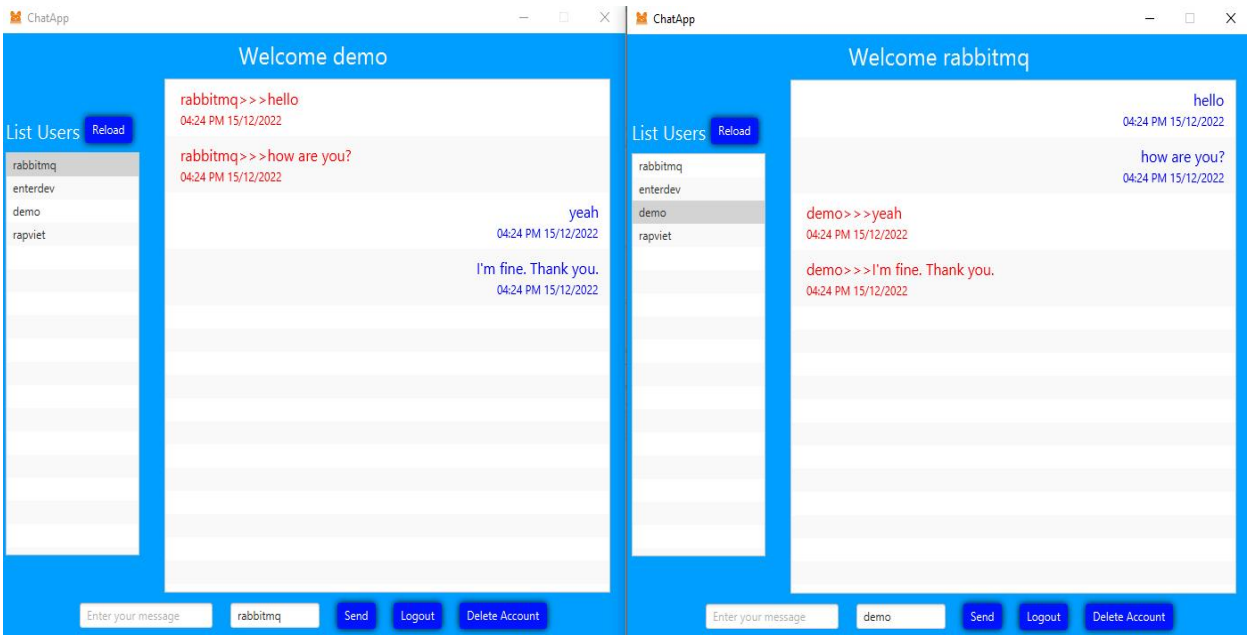
Hình 27. Đoạn mã gửi tin nhắn đến queue của RabbitMQ.

3.3.2.2. Nhận tin nhắn.

```
3 usages  PhucEnterdev
public class Receiver {
    1 usage  PhucEnterdev
    public Receiver(String host, String queue, ListView<VBox> listChat) {
        ConnectionFactory factory = new ConnectionFactory();
        Connection connection;
        factory.setHost(host);

        try {
            connection = factory.newConnection();
            Channel channel = connection.createChannel();
            channel.queueDeclare(queue, b: false, b1: false, b2: false, map: null);
            DeliverCallback deliverCallback = (s, delivery) -> {
                // Get date and format
                Date date = new Date();
                SimpleDateFormat dateFormat = new SimpleDateFormat( pattern: "hh:mm aa dd/MM/yyyy");
                String strDate = dateFormat.format(date);

                String msgRcv = new String(delivery.getBody(), StandardCharsets.UTF_8);
                Text textMsg = new Text(msgRcv);
                textMsg.setFill(Color.RED);
                textMsg.setFont(new Font( v: 16));
                Text textTime = new Text(strDate);textTime.setFill(Color.RED);
                VBox vBox = new VBox();vBox.getChildren().addAll(textMsg, textTime);
                vBox.setAlignment(Pos.CENTER_LEFT);vBox.setSpacing(2);
                vBox.setPadding(new Insets( v: 5, v1: 5, v2: 5, v3: 10));
                listChat.setOrientation(Orientation.VERTICAL);
                listChat.refresh();autoScrollMessageList(listChat);
                Platform.runLater(() -> listChat.getItems().add(vBox));
                listChat.refresh();
            };
            channel.basicConsume(queue, b: true, deliverCallback, consumerTag -> {
            });
        } catch (IOException | TimeoutException | IllegalStateException e) {
            throw new RuntimeException(e);
        }
    }
```

Hình 28. Đoạn mã nhận tin nhắn từ queue của RabbitMQ.

3.3.2.3. Kết quả của quá trình gửi và nhận tin nhắn.

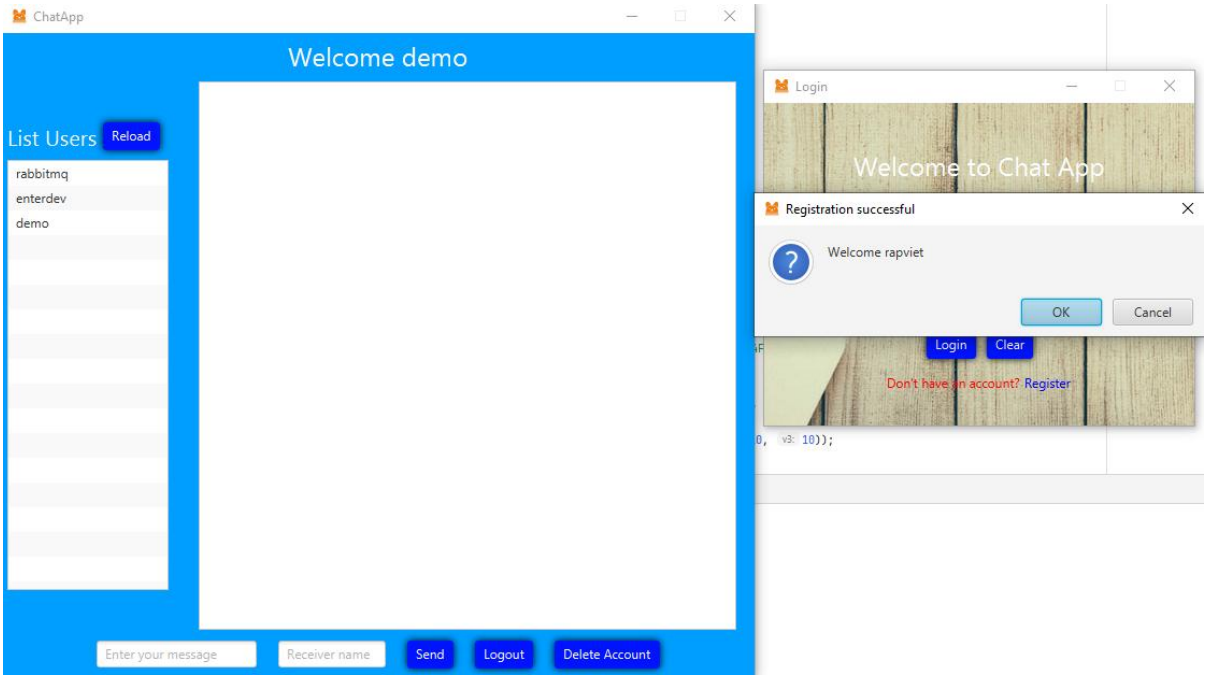


Hình 29. Kết quả quá trình gửi và nhận tin nhắn.

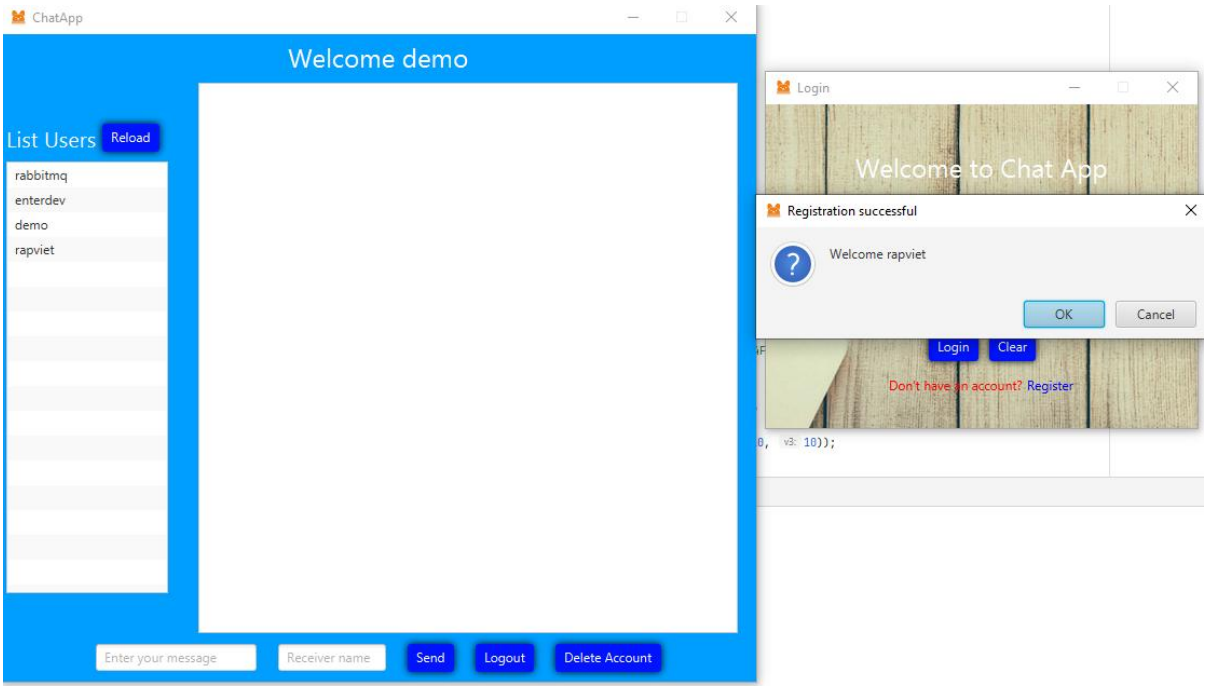
3.3.3. Sự kiện của các chức năng bổ trợ.

3.3.3.1. Sự kiện nhấn nút Reload.

Khi nhấn nút Reload, hệ thống sẽ lấy tất cả các tài khoản có trong cơ sở dữ liệu để hiển thị lên danh sách User. Được sử dụng nhằm mục đích khi có người dùng mới đăng ký tài khoản hoặc người dùng xóa tài khoản.



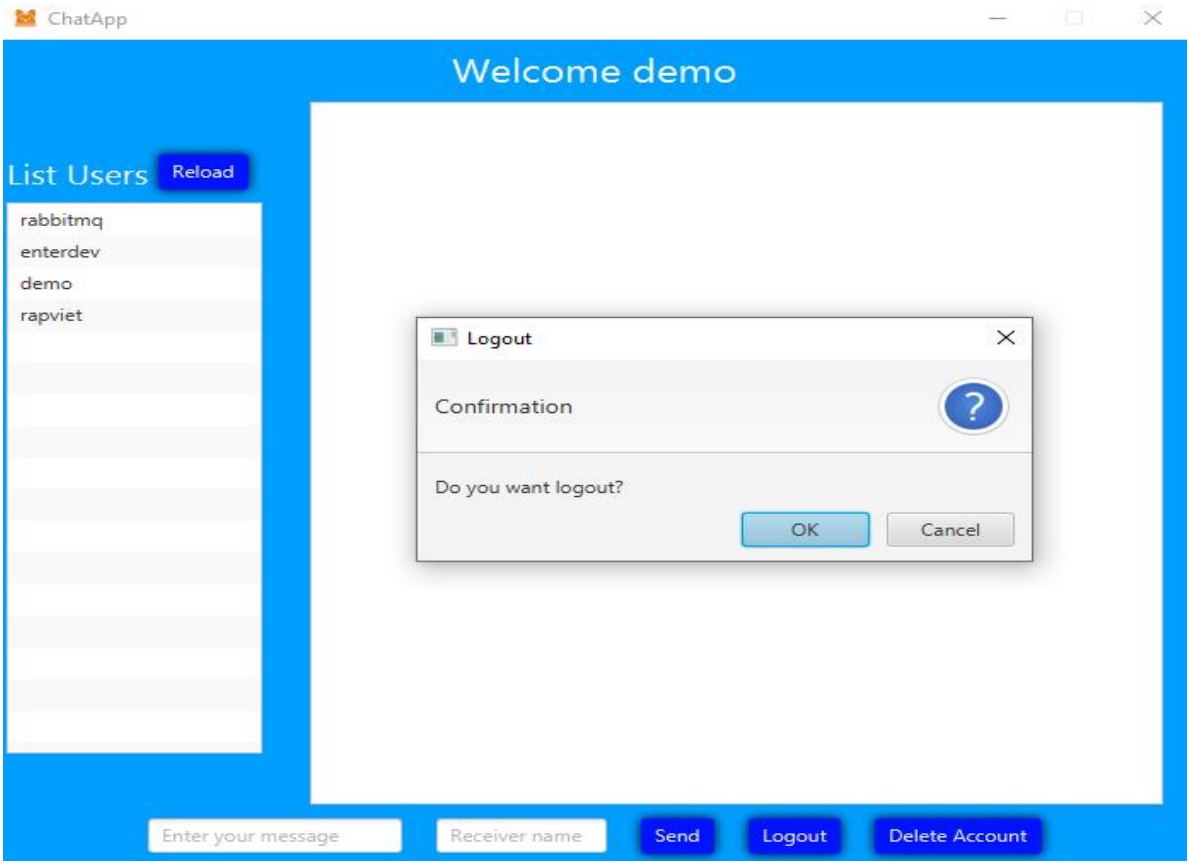
Hình 30. Kết quả trước khi người dùng nhấn Reload.



Hình 31. Kết quả sau khi người dùng nhấn Reload.

3.3.3.2. Sự kiện nhấn nút Logout.

Khi nhấn nút Logout, hệ thống sẽ hiển thị thông báo xác nhận. Tài khoản được đăng xuất khi người dùng nhấn nút OK.



Hình 32. Kết quả của sự kiện nhấn nút Logout.

CHƯƠNG 4. HƯỚNG DẪN CÀI ĐẶT, ĐÁNH GIÁ, KIỂM THỬ

4.1. Hướng dẫn cài đặt.

4.1.1. Cài đặt môi trường phát triển.

- Download và cài đặt IntelliJ IDEA.

Link download : https://www.jetbrains.com/idea/download/#section=windows

- Download và cài đặt JDK 19.

Link download : https://www.oracle.com/java/technologies/downloads/#jdk19-windows

- Download và giải nén JavaFX SDK.

Link download : https://gluonhq.com/products/javafx/

- Download và cài đặt MySQL Server và MySQL Workbench.

Link download MySQL Server : https://dev.mysql.com/downloads/windows/installer/8.0.html

Link download MySQL Workbench : https://dev.mysql.com/downloads/workbench/

- Download và cài đặt Erlang.

Link download : https://www.erlang.org/downloads

- Download và cài đặt RabbitMQ.

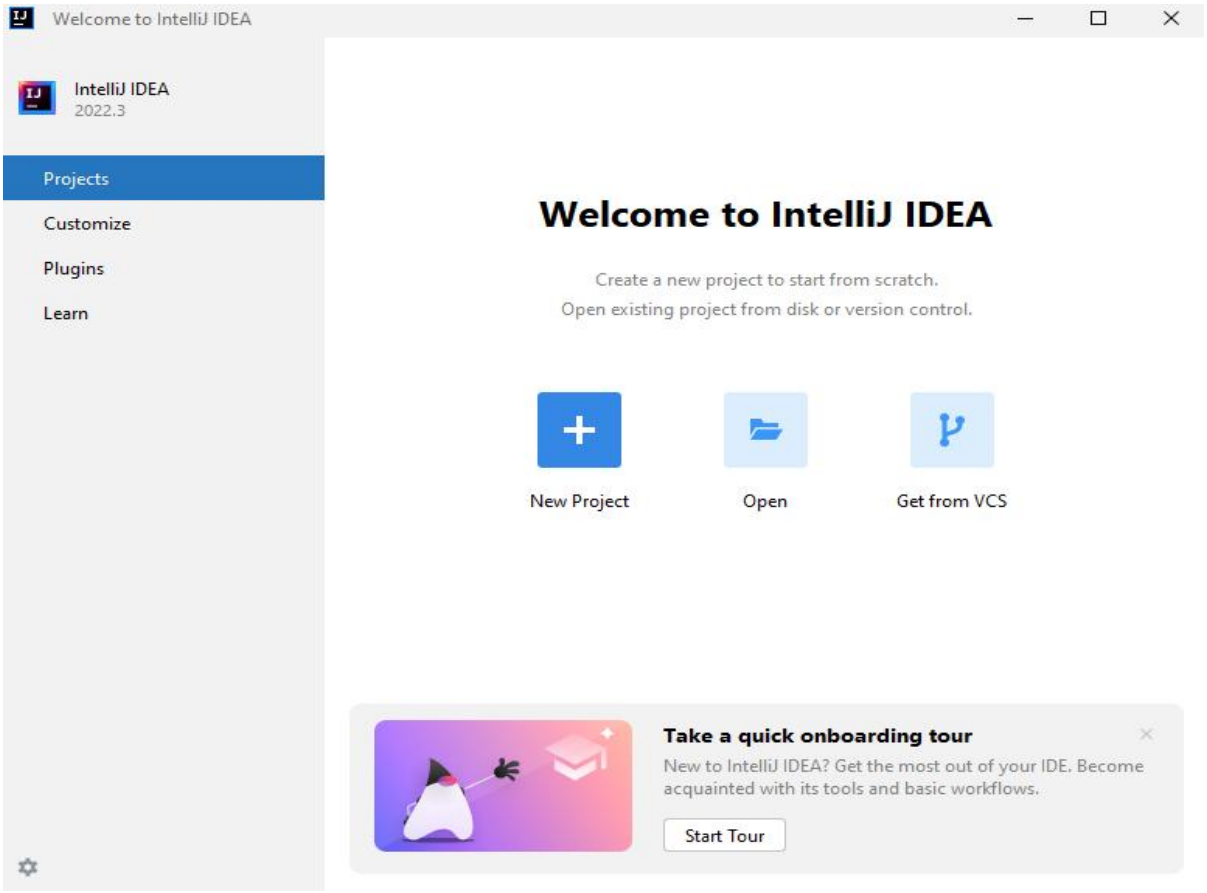
Link download : https://github.com/rabbitmq/rabbitmq-server/releases

4.1.2. Chạy chương trình.

Bước 1. Mở MySQL Workbench và tạo cơ sở dữ liệu như hình bên dưới.

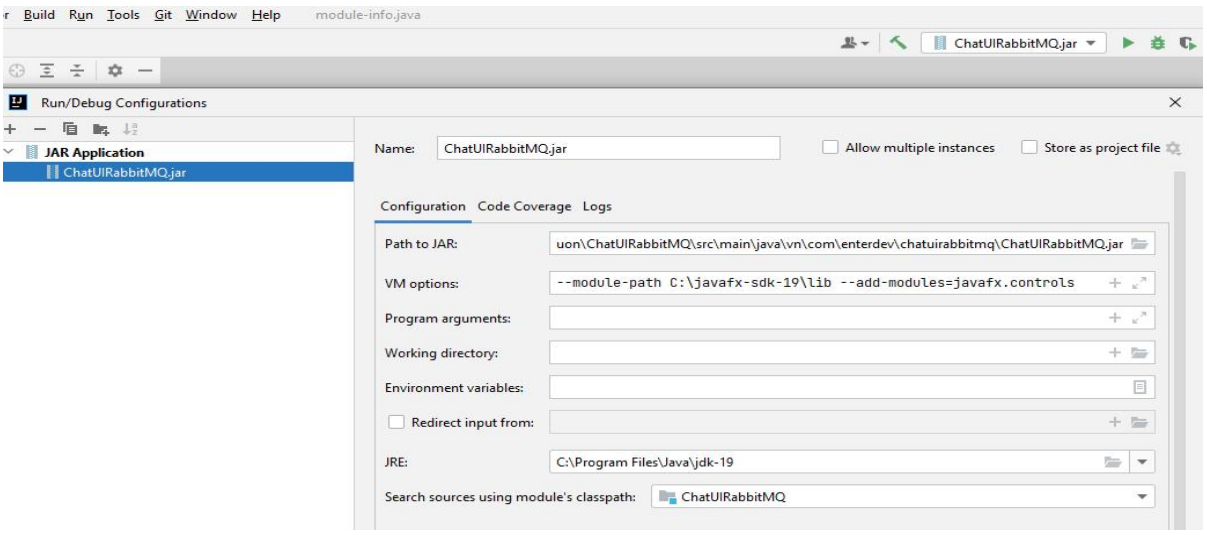
Hình 33. Tạo cơ sở dữ liệu cho ứng dụng.

Bước 2. Mở IntelliJ IDEA chọn Open và chọn vào thư mục chứa mã nguồn.



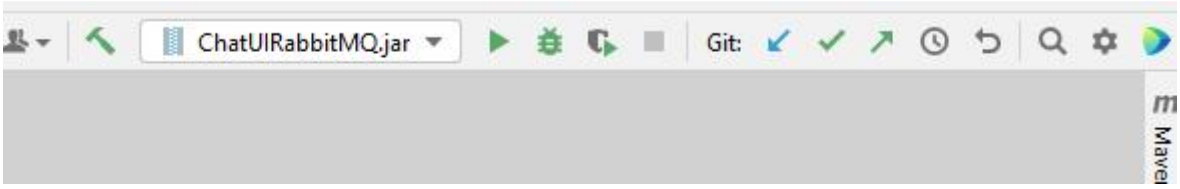
Hình 33. Giao diện IntelliJ IDEA.

Bước 3. Trên thanh Toolbar chọn Run, chọn Edit configurations, sau đó copy đường dẫn đến thư mục lib của JavaFX SDK vừa tải xuống thay thế đường dẫn sau --module-path.



Hình 34. Cấu hình cho file ChatUIRabbitMQ.jar.

Bước 4. Thực thi chương trình bằng cách nhấn vào nút Run.



Hình 35. Thực thi chương trình ứng dụng.

4.2. Đánh giá, kiểm thử.

Kiểm thử tính khả dụng là kiểm tra ứng dụng có thân thiện với người dùng hay không. Người dùng có thể hiểu ứng dụng dễ dàng hay không:

- Nội dung chính xác, không có bất kỳ lỗi ngữ pháp nào.

- Người dùng dễ dàng sử dụng.

Kiểm thử chức năng là để xác minh sản phẩm có đáp ứng các đặc điểm chức năng, nghiệp vụ được đề cập trong tài liệu đặc tả hay không:

- Sản phẩm đáp ứng các chức năng đã được đề cập trong đặt tả.

Kiểm thử cơ sở dữ liệu là việc kiểm tra dữ liệu được hiển thị trong ứng dụng có khớp với dữ liệu được lưu trữ trong cơ sở dữ liệu hay không. Dữ liệu thao tác trên ứng dụng có được thêm vào cơ sở dữ liệu một cách chính xác hay không:

- Dữ liệu hiển thị trong ứng dụng khớp với dữ liệu được lưu trữ trong cơ sở dữ liệu.

- Dữ liệu thao tác trên ứng dụng được thêm vào cơ sở dữ liệu là chính xác.

Kiểm thử tính bảo mật là kiểm nghiệm để xác định bất kỳ sai sót và lỗ hỏng bảo mật nào:

- Có tính bảo mật bằng cách gửi xác thực bằng email.

PHẦN KẾT LUẬN

*** KẾT QUẢ ĐẠT ĐƯỢC.**

- Hiểu được cách thức hoạt động của các thành phần ứng dụng trong JavaFX

- Hiểu được cách thức hoạt động của RabbitMQ và tích hợp vào giao diện JavaFX.

- Xây dựng được ứng dụng chat với các tính năng cơ bản, hoạt động tốt, không có lỗi.

*** HƯỚNG PHÁT TRIỂN.**

- Phát triển thêm tính năng Chat group.

- Phát triển thêm tính năng kết bạn.

- Phát triển thêm tính năng lưu lịch sử tin nhắn.

TÀI LIỆU THAM KHẢO

Tài liệu về RabbitMQ:

Link 1: https://www.rabbitmq.com/getstarted.html

Link 2: https://viblo.asia/p/gioi-thieu-ve-rabbitmq-gDVK2G30ZLj

Tài liệu về JavaFX:

Link 1: https://openjfx.io/openjfx-docs/

Link 2: https://fxdocs.github.io/docs/html5/

Link 3: https://jenkov.com/tutorials/javafx/index.html