

**ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**



BÀI TẬP LỚN

Ứng dụng nhắn tin nội bộ tích hợp Chatbot

Sinh viên thực hiện:

Họ và Tên	MSSV
Nguyễn Trường Sơn	20230097
Nguyễn Thành Bách	20235660
Trần Trung Hiếu	20235716
Nguyễn Đức Nam Khánh	20235755
Phạm Trần Tiên Phong	20235807
Nguyễn Quang Thiện	20235838

Học phần: Lập trình hướng đối tượng

Mã học phần: IT3103

Mã lớp học: 156772

Giảng viên hướng dẫn:

TS. TRẦN THẾ HÙNG

HÀ NỘI, 06/2025

Mục lục

1 Phân chia công việc	7
2 Tổng quan bài toán	8
2.1 Đặt vấn đề	8
2.2 Mục tiêu dự án	8
3 Thiết kế hệ thống	10
3.1 Use Case Diagram	10
3.1.1 Use Case Tổng Quát	10
3.1.2 Use Case Thành Phần	10
3.2 Package Diagram	12
3.3 Class Diagram	13
4 Chương trình	16
4.1 Cấu trúc thư mục	16
4.2 Các chức năng	17
4.2.1 Gửi và nhận tin nhắn giữa nhiều người dùng qua mạng LAN hoặc Internet	17
4.2.2 Tương tác thông minh với Chatbot AI Gemini	18
4.2.3 Quản lý tài khoản người dùng (Đăng ký – Đăng nhập – Cập nhật thông tin)	20
4.2.4 Giao diện hiện đại, hỗ trợ Markdown, Emoji và hiệu ứng sinh động	22
4.2.5 Lưu trữ lịch sử trò chuyện và dữ liệu người dùng bằng SQLite	22
4.2.6 Xử lý ngoại lệ rõ ràng và chuyên biệt	22
5 Các kiến thức OOP áp dụng	24
5.1 Đóng gói (Encapsulation)	24
5.2 Trừu tượng hóa (Abstraction)	24
5.3 Ké thừa (Inheritance)	25
5.4 Đa hình (Polymorphism)	25
5.4.1 Đa hình qua overriding	26
5.5 Đa hình qua Overloading	26
5.5.1 Đa hình qua interface	27
5.6 Java Collections Framework	27
5.6.1 Quản lý Message Listeners	27
5.6.2 Lưu trữ Client Connections	27
5.6.3 Chat History Management	28

5.7 Đánh giá	28
6 Công nghệ sử dụng	28
6.1 JavaFX	28
6.2 API Chatbot	29
6.2.1 API	29
6.2.2 API RESTful và Giao thức HTTP/HTTPS	29
6.2.3 Định dạng JSON	30
6.3 WebView	32
6.3.1 HTML	32
6.3.2 Chuyển đổi Markdown to HTML	33
6.4 Socket TCP & Đa luồng	33
6.4.1 Socket & Giao tiếp mạng	33
6.4.2 Giao tiếp Client – Server qua Socket	33
6.4.3 Đa luồng và xử lý đồng thời	34
7 Tổng kết	35
7.1 Kết quả đã đạt được	35
7.1.1 Ưu điểm	35
7.1.2 Nhược điểm	35

Danh sách hình vẽ

1	Use Case tổng quát	10
2	Use Case quản lý tài khoản.	11
3	Use Case chat với người khác.	11
4	Use Case tương tác với Chatbot.	11
5	Use Case quản lý thông tin User.	12
6	Package Diagram	12
7	Biểu đồ lớp chi tiết cho gói Model.	13
8	Biểu đồ lớp chi tiết cho gói Server.	13
9	Biểu đồ lớp chi tiết cho gói Service.	14
10	Biểu đồ lớp chi tiết cho gói Exception.	14
11	Biểu đồ lớp chi tiết cho gói Controller.	15
12	Giao tiếp nhiều người dùng qua server trung tâm.	18
13	Giao diện trò chuyện với Chatbot Gemini	19
14	Ảnh minh họa	20
16	Quản lý tài khoản người dùng.	21
17	Getter , setter trong User.java	24
18	Ảnh minh họa	25
19	Ảnh minh họa	25
20	Ảnh minh họa	26
21	Ảnh minh họa	26
22	Ảnh minh họa	27
23	Message Listeners	27
24	Clients Connections	27
25	Chat History	28
26	REST API	30
27	HTML Syntax	32
28	ServerSocket	34
29	Socket	34
30	Đa luồng	35

Danh sách bảng

1	Phân chia công việc giữa các thành viên	7
---	---	---

1 Phân chia công việc

Trong quá trình thực hiện đề tài, nhóm chia đều khối lượng công việc giữa sáu thành viên. Mỗi người phụ trách một nhánh quan trọng trong việc tạo nên sản phẩm chung.

Bảng 1: Phân chia công việc giữa các thành viên

Họ và tên	Nhiệm vụ
Nguyễn Trường Sơn	Xây dựng tính năng nhắn tin cục bộ bằng Socket, Vẽ ClassDiagram, làm báo cáo
Nguyễn Thành Bách	Xây dựng tính năng API Chatbot & Markdown to HTML, Vẽ UseCase, làm báo cáo
Trần Trung Hiếu	Code phần About, ProfileSetting, làm slide
Nguyễn Đức Nam Khánh	Code phần Login, Register, làm báo cáo
Phạm Trần Tiến Phong	Code phần ProfileSetting , Code phần UserService để lưu trữ data người dùng
Nguyễn Quang Thiện	Fix các lỗi hiển thị của app, code phần lưu trữ tin nhắn, làm slide

Mỗi thành viên đều hiểu rõ phần việc mình phụ trách, đồng thời nắm được tổng quan toàn bộ quy trình và hệ thống. Nhóm phối hợp tốt trong việc thiết kế logic tổng thể và hỗ trợ nhau trong các bước quan trọng như lập trình ứng dụng và trình bày báo cáo.

2 Tổng quan bài toán

2.1 Đặt vấn đề

Trong quá trình học tập và tiếp cận với các công nghệ hiện đại, nhóm em luôn mong muốn tạo ra một sản phẩm không chỉ ứng dụng kiến thức đã học mà còn có khả năng phục vụ thực tế. Nhận thấy sự phát triển mạnh mẽ của trí tuệ nhân tạo, đặc biệt là các mô hình AI hỗ trợ hội thoại, nhóm em đã nảy ra ý tưởng xây dựng một ứng dụng chatbot thông minh, nơi người dùng có thể vừa trò chuyện cùng nhau, vừa tương tác với một trợ lý ảo được tích hợp AI.

Không dừng lại ở việc trao đổi đơn thuần, ứng dụng của nhóm em còn hướng đến một giao diện thân thiện, đẹp mắt, dễ sử dụng, kết hợp khả năng xử lý thời gian thực qua mạng và lưu trữ thông tin bền vững. Đây cũng là cơ hội để nhóm em rèn luyện kỹ năng lập trình hướng đối tượng, xử lý đa luồng, giao tiếp mạng, tích hợp API và thiết kế giao diện người dùng – những kỹ năng rất quan trọng trong ngành phần mềm.

2.2 Mục tiêu dự án

Dự án này là thành quả của sự kết hợp giữa tinh thần học hỏi và mong muốn ứng dụng công nghệ vào cuộc sống. Nhóm em đã lựa chọn xây dựng một **ứng dụng chatbot desktop** hoàn chỉnh bằng **JavaFX**, sử dụng công cụ **Maven** để quản lý các thư viện cần thiết. Đặc biệt, ứng dụng tích hợp trí tuệ nhân tạo **Gemini** của Google để có thể trò chuyện và phản hồi một cách thông minh, kết hợp với giao thức **Socket TCP** cho phép các người dùng trong mạng nội bộ hoặc Internet kết nối và trò chuyện với nhau theo thời gian thực. Dữ liệu người dùng và hội thoại được lưu trữ bằng **SQLite**, đảm bảo tính nhất quán và dễ truy xuất.

Cụ thể, ứng dụng của nhóm em có thể:

- Cho phép người dùng **đăng ký, đăng nhập** và tùy chỉnh hồ sơ cá nhân như tên, email, hình đại diện.
- Tạo phòng chat nơi người dùng có thể **trò chuyện với nhau trực tiếp** qua mạng LAN hoặc Internet.
- Cung cấp giao diện để người dùng **giao tiếp với chatbot Gemini**, hỗ trợ hiển thị markdown, biểu tượng cảm xúc và đoạn mã lập trình (nếu cần).
- **Ghi lại toàn bộ lịch sử hội thoại**, lưu dưới dạng text và cả trong cơ sở dữ liệu để dễ dàng xem lại.

Dự án được tổ chức với các thành phần chính:

- **Controller:** Quản lý giao diện và tương tác người dùng, như LoginController, ChatViewController.
- **Service:** Thực hiện các chức năng xử lý nghiệp vụ như UserService cho đăng nhập và GeminiService cho AI.
- **Model:** Mô tả các đối tượng dữ liệu như User, Message, giúp lưu trữ và xử lý dễ dàng.
- **Network:** Hệ thống mạng sử dụng socket và luồng riêng để các người dùng có thể giao tiếp đồng thời.

Các file cấu hình như pom.xml, config.properties và module-info.java giúp ứng dụng hoạt động ổn định trong môi trường **Java 23**, đảm bảo khả năng mở rộng và tích hợp trong tương lai.

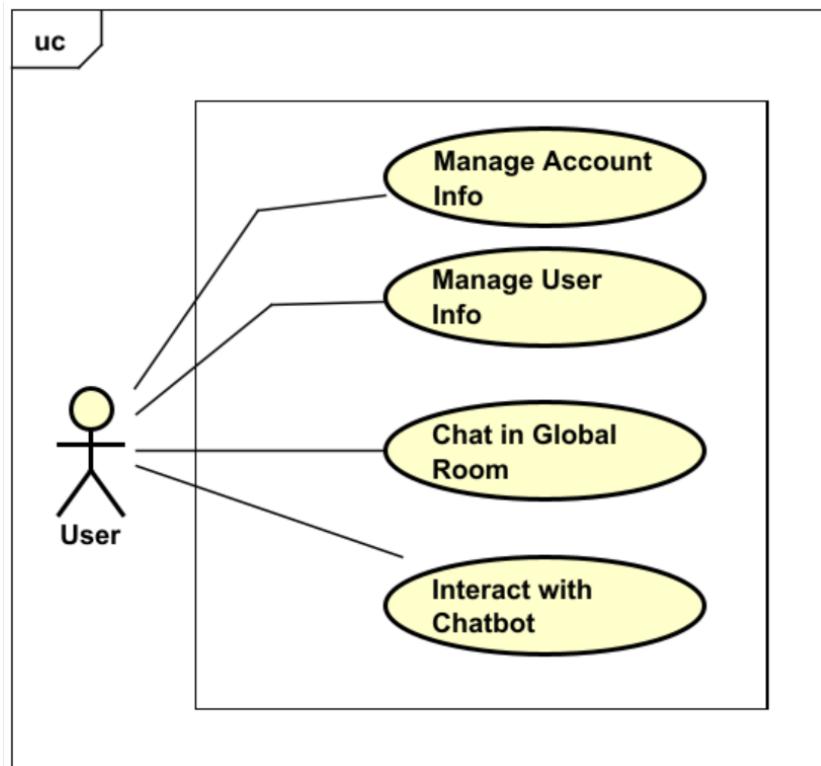
Thông qua dự án này, nhóm em không chỉ ôn tập và củng cố kiến thức từ học phần **Lập trình hướng đối tượng** mà còn có cơ hội tiếp cận với các công nghệ AI tiên tiến – điều mà chúng em tin rằng sẽ rất hữu ích nếu có dịp triển khai trong thực tế sau này.

Chúng em hy vọng rằng sản phẩm nhỏ này sẽ mang lại sự thú vị và cảm nhận được tinh thần nghiêm túc, sáng tạo mà nhóm đã có gắng trong suốt quá trình thực hiện.

3 Thiết kế hệ thống

3.1 Use Case Diagram

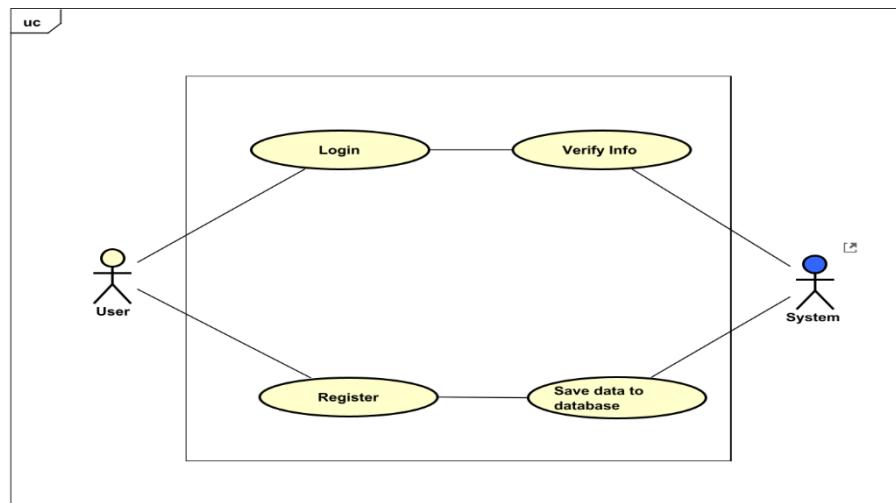
3.1.1 Use Case Tổng Quát



Hình 1: Use Case tổng quát.

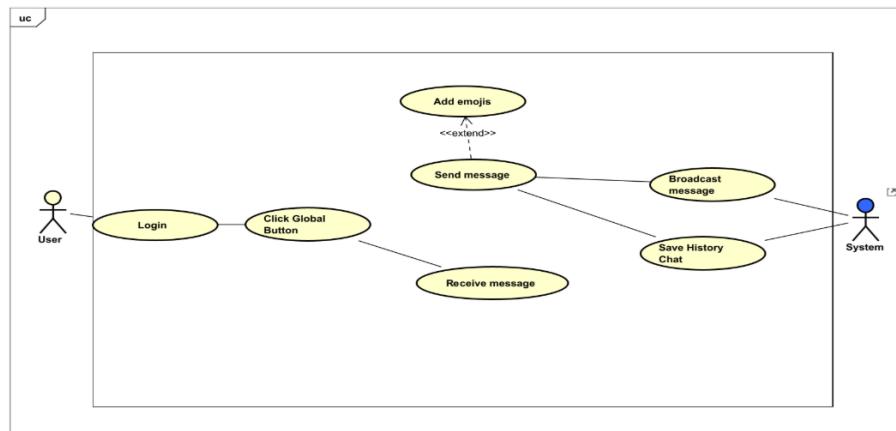
3.1.2 Use Case Thành Phần

Use Case Quản lý tài khoản



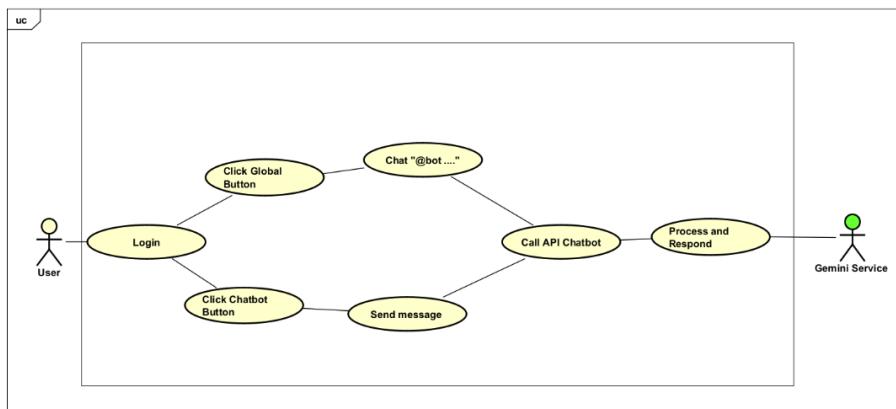
Hình 2: Use Case quản lý tài khoản.

Use Case Chat với người khác



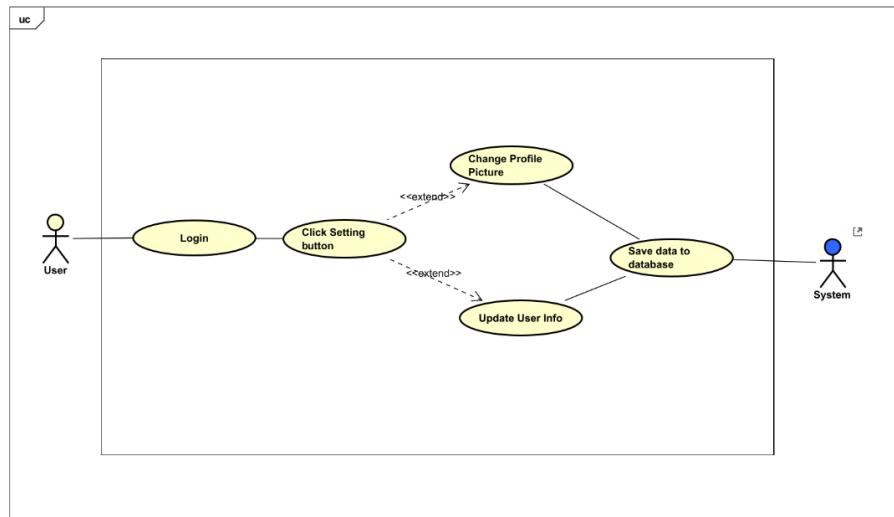
Hình 3: Use Case chat với người khác.

Use Case Tương tác với Chatbot



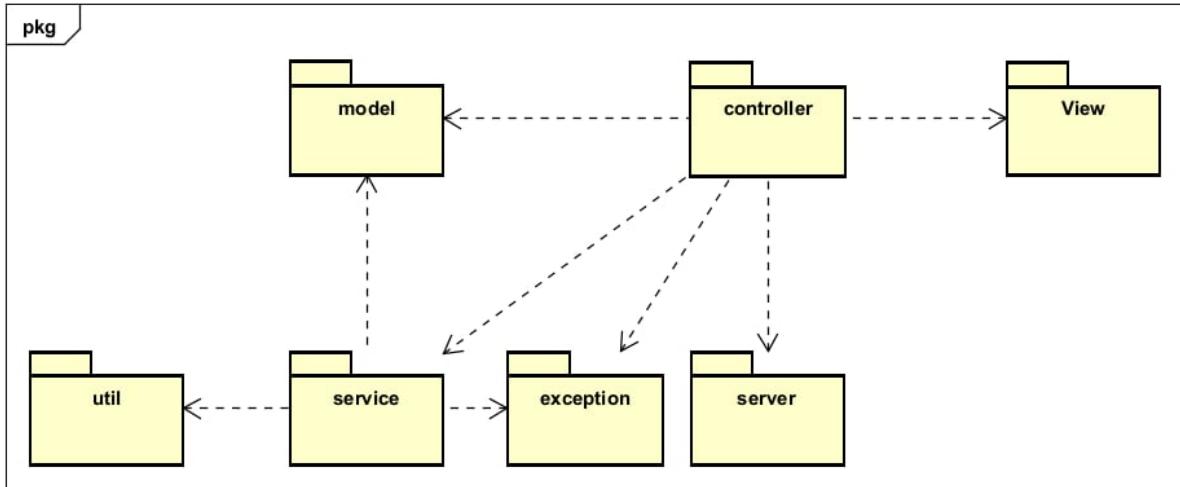
Hình 4: Use Case tương tác với Chatbot.

Use Case Quản lý thông tin User



Hình 5: Use Case quản lý thông tin User.

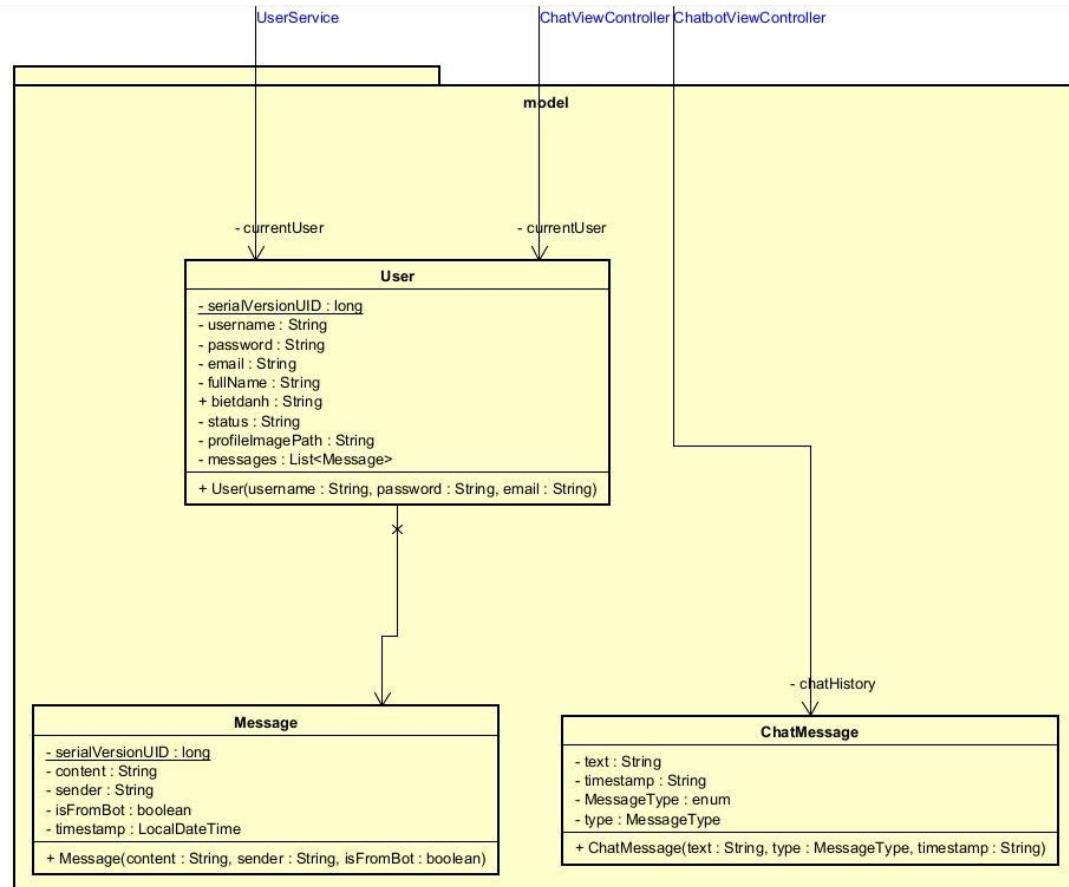
3.2 Package Diagram



Hình 6: Package Diagram

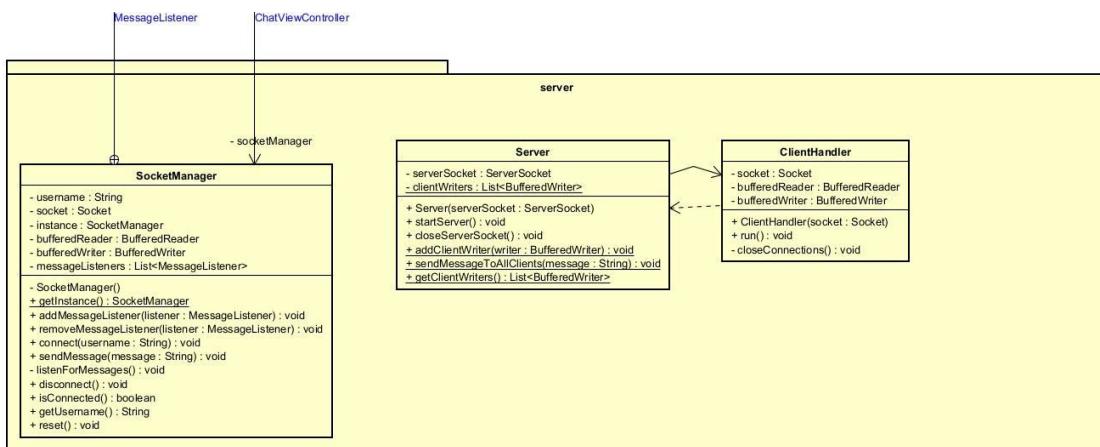
3.3 Class Diagram

Biểu đồ lớp chi tiết cho gói Model:



Hình 7: Biểu đồ lớp chi tiết cho gói Model.

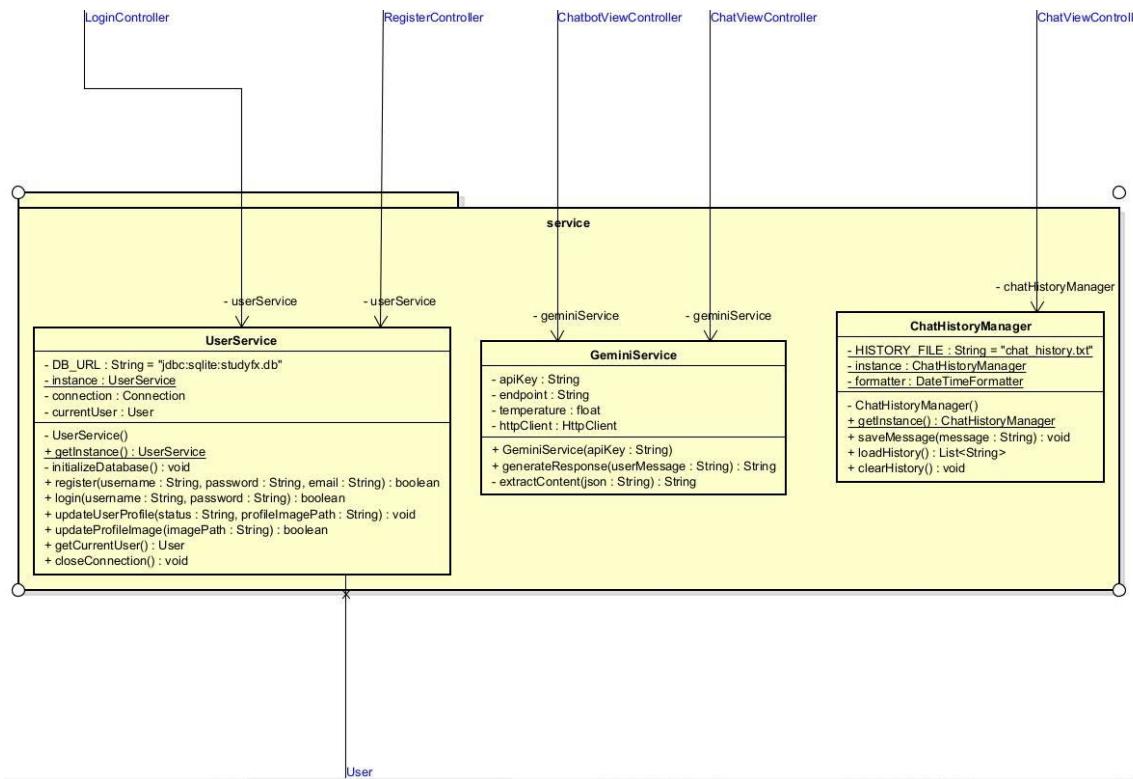
Biểu đồ lớp chi tiết cho gói Server:



Hình 8: Biểu đồ lớp chi tiết cho gói Server.

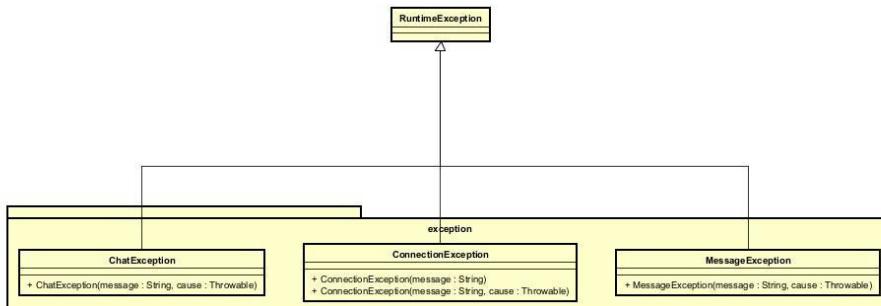
3 Thiết kế hệ thống

Biểu đồ lớp chi tiết cho gói Service:



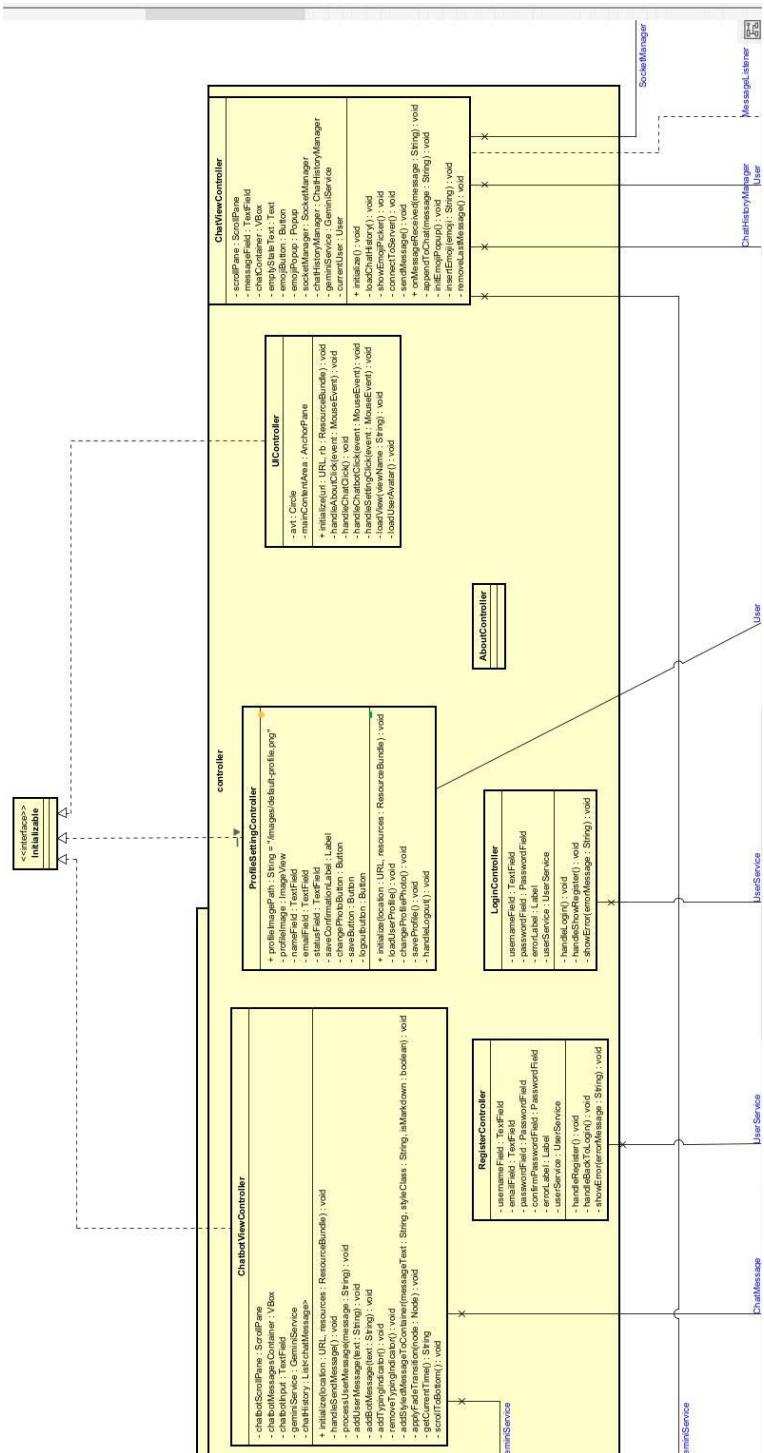
Hình 9: Biểu đồ lớp chi tiết cho gói Service.

Biểu đồ lớp chi tiết cho gói Exception:



Hình 10: Biểu đồ lớp chi tiết cho gói Exception.

Biểu đồ lớp chi tiết cho gói Controller:



Hình 11: Biểu đồ lớp chi tiết cho gói Controller.

4 Chương trình

4.1 Cấu trúc thư mục

Cấu trúc thư mục dự án

```
OOP_BTL20242/
|
├── pom.xml          # Tệp cấu hình Maven
├── README.md        # Tệp mô tả dự án
├── studyfx.db        # Cơ sở dữ liệu SQLite
├── chat_history.txt  # Lịch sử trò chuyện lưu dưới dạng text
├── structure.txt    # File cây thư mục xuất bằng lệnh `tree /f
|
└── .idea/            # Cấu hình IntelliJ IDEA
    └── *.xml, .gitignore, workspace.xml, v.v.

|
└── src/
    └── main/
        └── java/
            └── com.training.studyfx/
                ├── App.java          # Điểm khởi động ứng dụng
                ├── controller/
                │   ├── AboutController.java
                │   ├── ChatViewController.java
                │   ├── ChatbotViewController.java
                │   ├── LoginController.java
                │   ├── ProfileSettingController.java
                │   ├── RegisterController.java
                │   └── UIController.java
                ├── model/              # Mô hình dữ liệu
                │   ├── User.java
                │   ├── Message.java
                │   └── ChatMessage.java
                ├── service/            # Lớp dịch vụ xử lý logic
                │   ├── GeminiService.java  # Gọi API Gemini
                │   ├── ChatHistoryManager.java # Quản lý lưu lịch sử chat
                │   └── UserService.java    # Xử lý người dùng
                └── server/             # Socket server đa luồng
                    ├── Server.java
                    └── SocketManager.java # Giao tiếp socket client
```

```

    |   |   └── ClientHandler.java
    |   └── exception/          # Định nghĩa ngoại lệ tùy chỉnh
    |       ├── ChatException.java
    |       ├── ConnectionException.java
    |       └── MessageException.java
    └── util/                  # Tiện ích hỗ trợ hệ thống
        └── MarkdownToHtml.java
    └── module-info.java        # Cấu hình module Java (Java 9+)

└── resources/
    ├── config.properties      # Tập cấu hình (API key Gemini)
    ├── fxml/                 # Giao diện JavaFX
    |   ├── AboutView.fxml
    |   ├── ChatView.fxml
    |   ├── ChatbotView.fxml
    |   ├── ListView.fxml
    |   ├── LoginView.fxml
    |   ├── ProfileSettingView.fxml
    |   ├── RegisterView.fxml
    |   └── UI.fxml
    ├── styles/               # Giao diện CSS
    |   └── ui.css
    └── images/               # Tài nguyên hình ảnh giao diện
        └── logo.png, send.png, user.png, ...

└── upload/                # Tập người dùng upload
    ├── images/              # Ảnh chat trong quá trình sử dụng
    └── profile_images/      # Ảnh đại diện người dùng
        ├── bach_* .jpg
        ├── phong_* .jpg
        └── ...

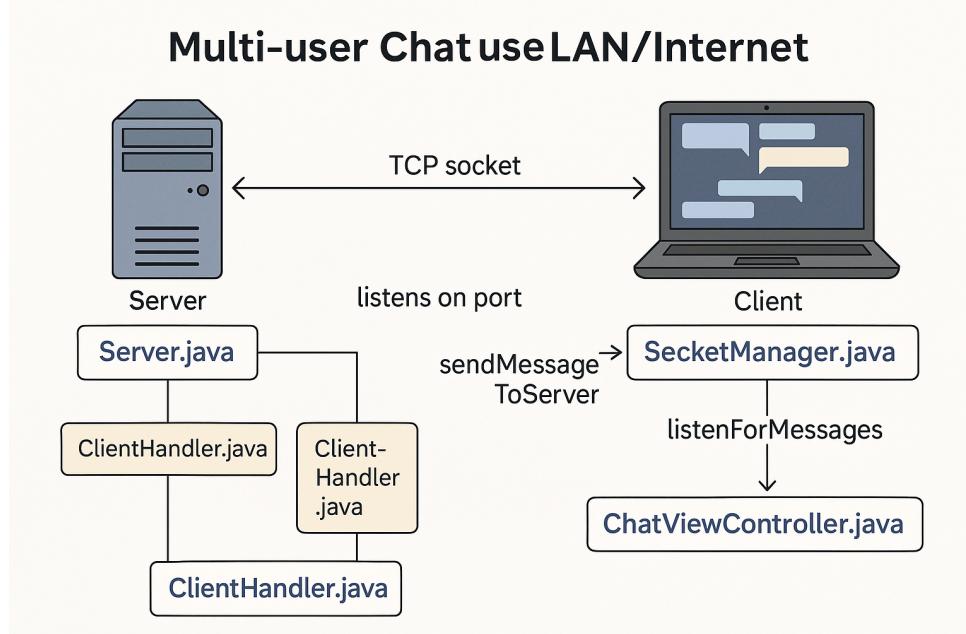
```

4.2 Các chức năng

4.2.1 Gửi và nhận tin nhắn giữa nhiều người dùng qua mạng LAN hoặc Internet

Trong quá trình phát triển ứng dụng, nhóm em đã xây dựng một hệ thống chat thời gian thực cho phép nhiều người dùng kết nối và trò chuyện với nhau qua mạng nội bộ hoặc Internet. Chức năng này được triển khai dựa trên giao thức **socket TCP** và sử dụng kỹ thuật lập trình đa luồng để đảm bảo mọi kết nối đều được xử lý độc lập, không bị chặn hoặc gián đoạn bởi các người dùng khác.

Về phía máy chủ, lớp Server.java có nhiệm vụ lắng nghe các kết nối đến thông qua một cổng cố định. Mỗi khi có một client mới kết nối, một đối tượng ClientHandler sẽ được tạo và chạy trên một luồng riêng biệt để đảm bảo xử lý song song.



Hình 12: Giao tiếp nhiều người dùng qua server trung tâm.

Ở phía người dùng, lớp SocketManager.java đóng vai trò là cầu nối giữa ứng dụng client và server. Nó cho phép gửi tin nhắn thông qua phương thức sendMessageToServer và luôn luôn lắng nghe tin nhắn mới từ server bằng phương thức listenForMessages.

Mọi hoạt động gửi và nhận tin nhắn đều được đồng bộ với giao diện người dùng thông qua lớp ChatViewController.java, nơi các tin nhắn sẽ được hiển thị dưới dạng khung hội thoại (chat bubble). Giao diện được xây dựng bằng các file FXML như ChatView.fxml và UI.fxml, giúp bố trí hợp lý khung chat, ô nhập văn bản và nút gửi.

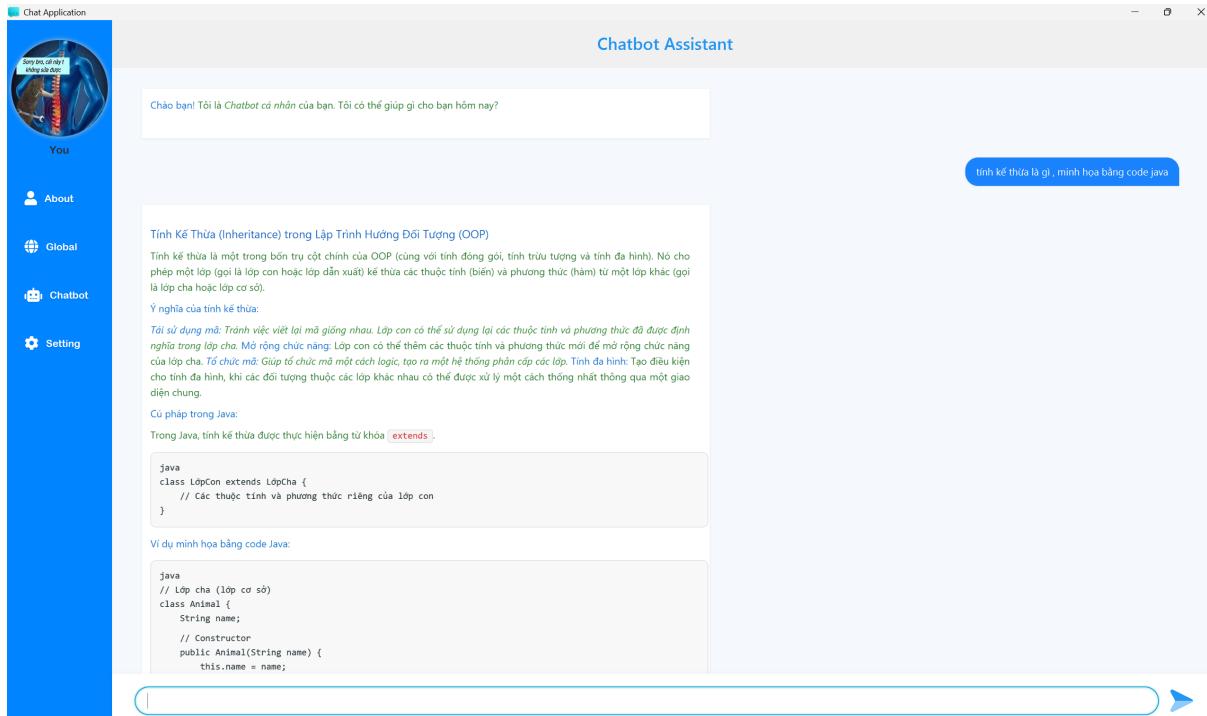
Kết quả đạt được: Tính năng này giúp người dùng dễ dàng trò chuyện với nhau một cách mượt mà, đồng thời cho phép mở rộng thành một hệ thống trò chuyện theo nhóm trong tương lai.

4.2.2 Tương tác thông minh với Chatbot AI Gemini

Ngoài việc trò chuyện với người dùng khác, nhóm em còn tích hợp trí tuệ nhân tạo **Gemini API** từ Google để xây dựng chức năng chatbot thông minh. Điều này cho phép người dùng đặt câu hỏi hoặc trò chuyện tự nhiên với AI và nhận được phản hồi tức thì, giống như đang nói chuyện với một trợ lý ảo.

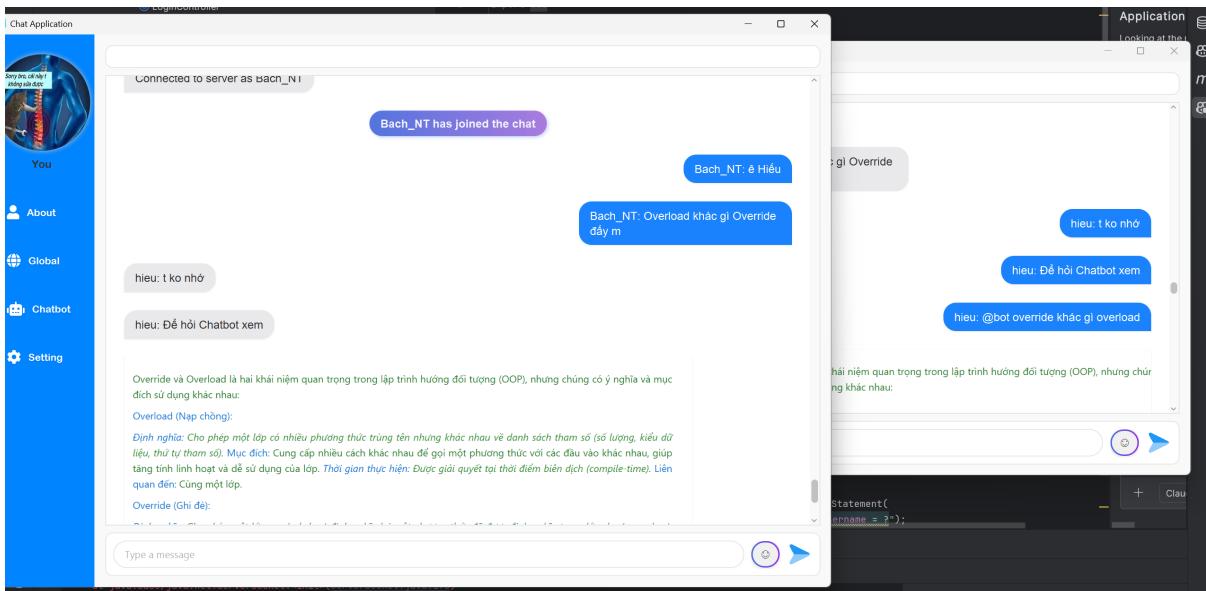
Lớp GeminiService.java đóng vai trò chính trong chức năng này. Tại đây, chúng em sử dụng thư viện java.net.http để gửi truy vấn (prompt) của người dùng lên Gemini thông

qua phương thức sendMessageToGemini. API key được lấy từ file cấu hình config.properties, đảm bảo bảo mật và linh hoạt khi triển khai.



Hình 13: Giao diện trò chuyện với Chatbot Gemini

Một tính năng đáng chú ý là trong Global thì người dùng có thể chat ”@bot ”để có thể tương tác với Chatbot , tính năng này tương tự với Meta AI trên Messenger bản Mobile.



Hình 14: Ảnh minh họa

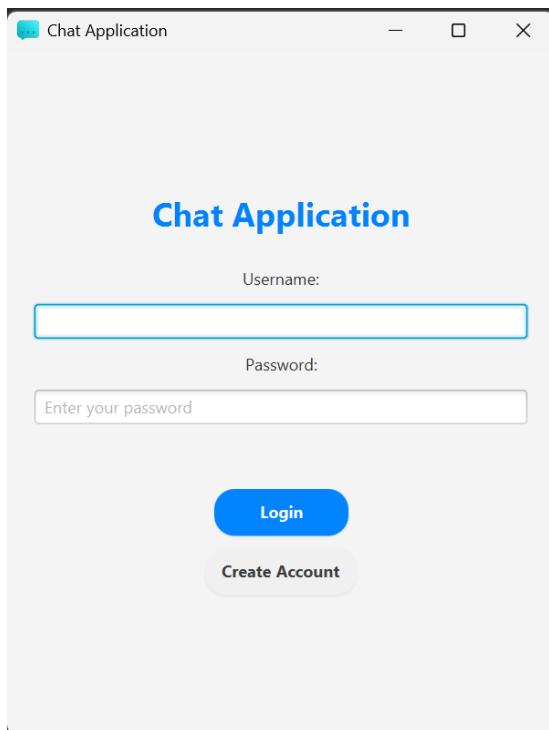
Một điểm đặc biệt là phản hồi từ Gemini thường có định dạng **markdown**, bao gồm chữ in đậm, danh sách, đoạn code... Do đó, nhóm em sử dụng lớp MarkdownToHtml.java để chuyển markdown sang HTML và hiển thị nội dung đó thông qua WebView trong JavaFX, mang lại trải nghiệm trực quan và đẹp mắt.

Kết quả đạt được: Người dùng có thể vừa trò chuyện với bạn bè, vừa tận hưởng sự thông minh và hỗ trợ từ chatbot AI một cách tự nhiên và linh hoạt.

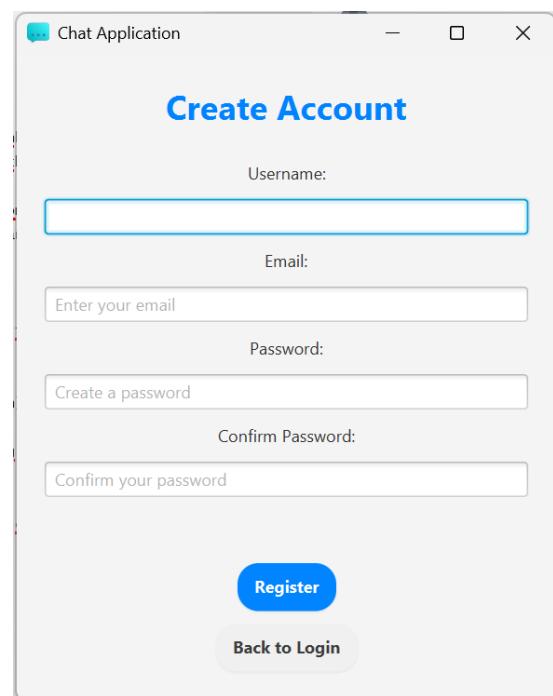
4.2.3 Quản lý tài khoản người dùng (Đăng ký – Đăng nhập – Cập nhật thông tin)

Nhằm đảm bảo cá nhân hóa trải nghiệm người dùng, ứng dụng của nhóm em hỗ trợ hệ thống tài khoản đầy đủ từ đăng ký, đăng nhập đến cập nhật hồ sơ cá nhân.

Trong phần đăng nhập, lớp LoginController.java thực hiện xác minh tài khoản dựa trên dữ liệu được lưu trong cơ sở dữ liệu SQLite. Nếu thành công, người dùng sẽ được chuyển đến giao diện chính. Đối với người dùng mới, lớp RegisterController.java hỗ trợ tạo tài khoản mới, đồng thời kiểm tra tính hợp lệ của thông tin nhập vào (email, xác thực mật khẩu...).

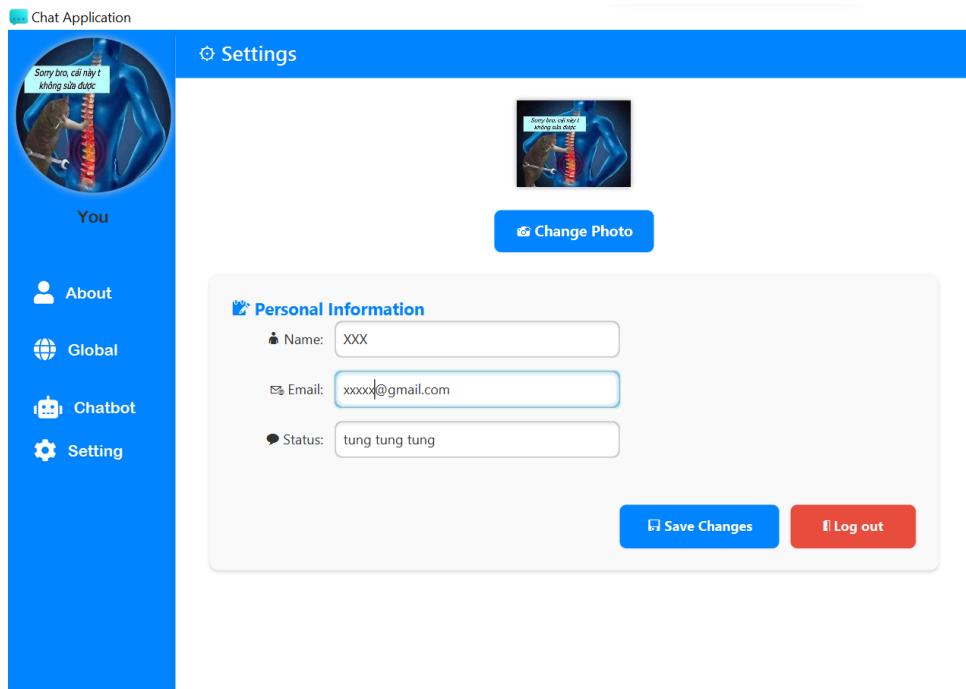


a Giao diện phần login



b Giao diện phần đăng ký

Người dùng cũng có thể cập nhật thông tin cá nhân như tên, địa chỉ email hoặc hình đại diện thông qua lớp ProfileSettingController.java. Tất cả các thao tác truy xuất hoặc cập nhật dữ liệu đều được xử lý bởi lớp UserService.java, tương tác trực tiếp với cơ sở dữ liệu.



Hình 16: Quản lý tài khoản người dùng.

Thông tin của mỗi người dùng được định nghĩa và quản lý bởi lớp User.java, với các thuộc tính như id, tên, email, mật khẩu và đường dẫn ảnh đại diện. Toàn bộ dữ liệu được lưu trữ trong file studyfx.db và quản lý qua thư viện sqlite-jdbc đã được khai báo trong pom.xml.

Kết quả đạt được: Người dùng có thể tự do tạo, đăng nhập và tùy chỉnh hồ sơ của mình một cách dễ dàng, đồng thời dữ liệu được bảo vệ và quản lý cẩn thận.

4.2.4 Giao diện hiện đại, hỗ trợ Markdown, Emoji và hiệu ứng sinh động

Giao diện người dùng đóng vai trò rất quan trọng đối với trải nghiệm tổng thể của ứng dụng. Vì vậy, nhóm em đã xây dựng giao diện hiện đại, rõ ràng và thân thiện dựa trên JavaFX kết hợp với FXML và CSS tùy biến.

File ui.css là nơi định nghĩa toàn bộ kiểu dáng giao diện như màu sắc, hiệu ứng bóng đổ, khung chat phân biệt user và bot, nút bấm với gradient động, hiệu ứng hover mượt mà... Nhờ đó, ứng dụng không chỉ đẹp mắt mà còn dễ sử dụng.

Các bộ cục FXML như LoginView.fxml, RegisterView.fxml, ChatView.fxml, ProfileSettingView.fxml giúp tách biệt hoàn toàn phần giao diện với phần xử lý logic, từ đó dễ bảo trì và mở rộng.

Ngoài ra, người dùng có thể sử dụng emoji thông qua các nút chức năng, đồng thời chuyển đổi giữa các phần giao diện như hồ sơ, bot, chat... bằng thanh sidebar tiện lợi.

Kết quả đạt được: Ứng dụng mang lại cảm giác sử dụng chuyên nghiệp, hiện đại và dễ tiếp cận cho mọi đối tượng người dùng.

4.2.5 Lưu trữ lịch sử trò chuyện và dữ liệu người dùng bằng SQLite

Để đảm bảo tính liên tục và bảo mật cho cuộc trò chuyện, nhóm em sử dụng cơ sở dữ liệu **SQLite** để lưu trữ thông tin người dùng và lịch sử chat.

Lớp UserService.java thực hiện các thao tác thêm, cập nhật và xác minh người dùng. Trong khi đó, lớp ChatHistoryManager.java có nhiệm vụ ghi và đọc dữ liệu chat từ file chat_history.txt, giúp người dùng có thể truy xuất lại hội thoại nếu cần.

Ngoài file studyfx.db là nơi lưu trữ chính, nhóm còn sử dụng file dataSources.xml và pom.xml để cấu hình kết nối đến CSDL và thư viện JDBC. File config.properties cũng được dùng để lưu API key cho Gemini một cách an toàn.

Kết quả đạt được: Ứng dụng có khả năng ghi nhớ và lưu trữ thông tin người dùng cũng như các cuộc trò chuyện một cách đầy đủ và đáng tin cậy.

4.2.6 Xử lý ngoại lệ rõ ràng và chuyên biệt

Trong quá trình phát triển phần mềm, việc xử lý lỗi rõ ràng và có tổ chức là cực kỳ quan trọng. Do đó, nhóm em đã định nghĩa các lớp ngoại lệ riêng biệt như ChatException,

ConnectionException và MessageException, mỗi lớp phụ trách một loại lỗi cụ thể.

Những lớp ngoại lệ này kế thừa từ Exception và được sử dụng trong các khối try-catch để giúp phát hiện và phân loại lỗi dễ dàng hơn. Nhờ đó, ứng dụng có thể cung cấp thông báo lỗi chính xác, giúp người dùng hiểu rõ nguyên nhân và nhà phát triển dễ dàng kiểm tra lại khi cần.

Kết quả đạt được: Tăng độ tin cậy cho hệ thống, đồng thời hỗ trợ người dùng và lập trình viên trong việc phát hiện và khắc phục lỗi một cách hiệu quả.

5 Các kiến thức OOP áp dụng

Trong quá trình xây dựng ứng dụng này, nhóm em đã vận dụng linh hoạt và hợp lý các kiến thức nền tảng của lập trình hướng đối tượng. Các kỹ thuật này không chỉ giúp mã nguồn trở nên rõ ràng, mà còn hỗ trợ việc bảo trì và phát triển ứng dụng một cách thuận lợi trong tương lai.

5.1 Đóng gói (Encapsulation)

Tính đóng gói là một trong những nguyên lý quan trọng nhất mà nhóm em đã triển khai một cách nhất quán trong toàn bộ dự án. Nguyên lý này được thể hiện qua việc ẩn giấu thông tin bên trong đối tượng và chỉ cho phép truy cập thông qua các phương thức được định nghĩa sẵn.

Các lớp dữ liệu như **User**, **Message**, và **ChatMessage** đều được thiết kế với các thuộc tính riêng tư (private), truy cập gián tiếp thông qua các hàm getter và setter. Chính điều này giúp đảm bảo rằng dữ liệu luôn ở trạng thái hợp lệ, đồng thời hạn chế tối đa các sai sót do truy cập trái phép từ bên ngoài. Sử dụng các thuộc tính private kết hợp với getter/setter để kiểm soát truy cập và thay đổi dữ liệu, ví dụ trong lớp User.java và Message.java.

```
public String getFullName() { 4 usages • NeoCyber05
    return fullName;
}

public void setFullName(String fullName) { 2 usages • NeoCyber05
    this.fullName = fullName;
}
```

Hình 17: Getter , setter trong User.java

Ưu điểm nổi bật nhất của việc đóng gói là giúp cô lập và kiểm soát chặt chẽ dữ liệu, qua đó nâng cao khả năng tái sử dụng và dễ dàng bảo trì mã nguồn. Tuy nhiên, nhóm em cũng hiểu rằng nếu không kiểm soát cẩn thận, việc tạo quá nhiều phương thức setter có thể làm mất đi tính toàn vẹn dữ liệu.

5.2 Trừu tượng hóa (Abstraction)

Trong dự án này, nhóm em đã sử dụng nguyên lý trừu tượng hóa nhằm giấu đi những chi tiết phức tạp và chỉ giữ lại phần cốt lõi cần thiết nhất cho người sử dụng.

Lớp **SocketManager** trừu tượng hóa việc giao tiếp mạng, các controller không cần quan tâm chi tiết kỹ thuật, chỉ cần sử dụng các phương thức đơn giản để gửi và nhận dữ liệu.

```
public interface MessageListener { 1 implementation • NeoCyber05
    void onMessageReceived(String message); 1 usage 1 implementation
}
```

Hình 18: Ảnh minh họa

Việc sử dụng trừu tượng hóa hiệu quả giúp mã nguồn trở nên đơn giản hơn, rõ ràng hơn và giảm bớt những phụ thuộc không cần thiết. Dẫu vậy, nhóm em cũng nhận thức rõ rằng nếu che giấu quá nhiều chi tiết kỹ thuật, việc tìm ra lỗi khi debug có thể sẽ khó khăn hơn.

5.3 Kế thừa (Inheritance)

Kế thừa là một tính chất mạnh mẽ của lập trình hướng đối tượng mà nhóm em tận dụng rất hiệu quả. Các lớp exception như **ChatException**, **ConnectionException**, **MessageException** đều kế thừa từ lớp **RuntimeException**. Cho phép các lớp con tái sử dụng code và mở rộng chức năng của lớp cha. Điều này giúp xử lý linh hoạt và cụ thể từng tình huống lỗi xảy ra, từ lỗi kết nối mạng đến lỗi xử lý tin nhắn.

```
public class ConnectionException extends RuntimeException {
```

Hình 19: Ảnh minh họa

Nhờ kế thừa, nhóm em đã tận dụng tối đa mã nguồn, giảm thiểu những đoạn mã trùng lặp và dễ dàng mở rộng các tính năng mới trong tương lai. Tuy nhiên, nhóm cũng lưu ý rằng việc sử dụng kế thừa cần tổ chức hợp lý, nếu không sẽ dễ gây rối loạn cấu trúc mã nguồn và giảm tính linh hoạt của hệ thống.

5.4 Đa hình (Polymorphism)

Cuối cùng, đa hình là một tính chất quan trọng khác đã được nhóm em ứng dụng vào dự án. Đa hình cho phép chương trình xử lý các đối tượng khác nhau thông qua cùng một giao diện chung, nhờ đó, chương trình trở nên rất linh hoạt và dễ dàng thay đổi theo từng tình huống.

5.4.1 Đa hình qua overriding

Ví dụ cụ thể, nhóm em áp dụng đa hình trong việc xử lý đa luồng. Các đối tượng ClientHandler đều được khởi tạo từ giao diện Runnable, cho phép chạy song song nhiều kết nối một cách hiệu quả.

```
public class ClientHandler implements Runnable {
    private Socket socket;
    private BufferedReader bufferedReader;
    private BufferedWriter bufferedWriter;
}

public ClientHandler(Socket socket) { ... }

@Override
public void run() {
    ...
}
```

Hình 20: Ảnh minh họa

5.5 Đa hình qua Overloading

```
public GeminiService(String apiKey) { 2 usages  ↳ NeoCyber05
    this.apiKey = apiKey;
    if (apiKey == null || apiKey.isEmpty()) {
        throw new IllegalArgumentException("API key không được để trống. Vui lòng kiểm tra file config.properties");
    }
    this.endpoint = "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent?key=" + a
    this.httpClient = HttpClient.newBuilder()
        .connectTimeout(Duration.ofSeconds(15))
        .build();
}

public GeminiService() { ↳ NeoCyber05
    this(loadApiKeyFromConfig());
}
```

Hình 21: Ảnh minh họa

5.5.1 Đa hình qua interface

```
public class ChatViewController implements SocketManager.MessageListener {
    @Override
    public void onMessageReceived(String message) {
        appendToChat(message); // Implementation cụ thể
    }
}
```

Hình 22: Ảnh minh họa

Ưu điểm lớn nhất của đa hình là làm tăng đáng kể tính linh hoạt và khả năng mở rộng của ứng dụng. Tuy nhiên, nếu không đặt tên và tổ chức hợp lý, đa hình có thể làm cho mã nguồn trở nên khó hiểu và gây ra những lỗi không mong muốn khi thực thi.

5.6 Java Collections Framework

Nhóm em đã sử dụng rộng rãi Java Collection Framework để quản lý dữ liệu một cách hiệu quả.

5.6.1 Quản lý Message Listeners

```
private List<MessageListener> messageListeners = new ArrayList<>(); 4 usages
```

Hình 23: Message Listeners

5.6.2 Lưu trữ Client Connections

```
private static List<BufferedWriter> clientWriters = new ArrayList<>();
```

Hình 24: Clients Connections

5.6.3 Chat History Management

```
public List<String> loadHistory() { 1 usage  ↳ NeoCyber05
    List<String> history = new ArrayList<>();
    try (BufferedReader reader = new BufferedReader(new FileReader(HISTORY_FILE))) {
        String line;
        while ((line = reader.readLine()) != null) {
            history.add(line);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    return history;
}
```

Hình 25: Chat History

5.7 Đánh giá

Qua việc áp dụng các kiến thức quan trọng của lập trình hướng đối tượng một cách hợp lý và linh hoạt, nhóm em không chỉ tạo ra một ứng dụng đáp ứng tốt các yêu cầu kỹ thuật mà còn xây dựng được một mã nguồn sạch, rõ ràng, dễ bảo trì và mở rộng. Nhóm em tin rằng đây sẽ là nền tảng vững chắc cho những dự án tương lai của mình.

6 Công nghệ sử dụng

6.1 JavaFX

- Là nền tảng chính để xây dựng giao diện người dùng đồ họa (GUI) với hiệu năng tốt và hỗ trợ đa nền tảng. Đây cũng là một trong những kiến thức trọng tâm được học tại học phần này.
- JavaFX cho phép kết hợp giữa FXML (định nghĩa giao diện dạng XML) và controller Java, giúp tách biệt rõ layout và logic.
- Hỗ trợ sự kiện, animation, binding dữ liệu và tích hợp CSS trực tiếp để làm đẹp UI mà không cần thư viện ngoài.

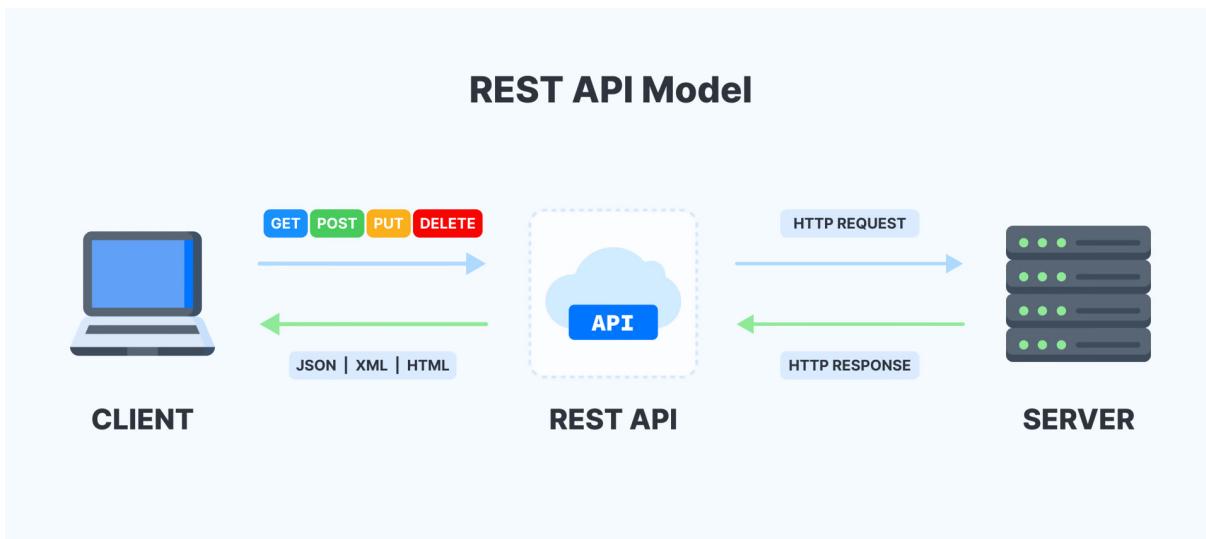
6.2 API Chatbot

6.2.1 API

- API (Application Programming Interface) là một tập hợp các quy tắc và giao thức cho phép các ứng dụng phần mềm khác nhau giao tiếp với nhau. Trong bối cảnh của Project này, Gemini API sẽ đóng vai trò là một cầu nối giữa Client (Ứng dụng JavaFX của bạn em) và bên hệ thống máy chủ Google.
- Thông qua API, Project có thể gửi dữ liệu (câu hỏi của người dùng) và nhận về kết quả (câu trả lời do AI tạo ra) mà không cần biết chi tiết về cách mô hình AI được xây dựng hay hoạt động động bên trong.

6.2.2 API RESTful và Giao thức HTTP/HTTPS

- Ứng dụng tương tác với Gemini API thông qua kiến trúc REST (Representational State Transfer). Đây là một kiểu kiến trúc phần mềm phổ biến để thiết kế các ứng dụng mạng.
- Giao thức HTTP/HTTPS: Mọi giao tiếp đều dựa trên giao thức HTTP, nền tảng của World Wide Web.
 - Phương thức POST: Được sử dụng để gửi dữ liệu đến máy chủ. Trong trường hợp này, ứng dụng gửi tin nhắn của người dùng và các tham số cấu hình đến Gemini API.
 - URL (Uniform Resource Locator): Mỗi dịch vụ của API được xác định bởi một địa chỉ duy nhất, gọi là **endpoint**.
 - HTTPS: Là phiên bản an toàn của HTTP, mã hóa dữ liệu truyền đi để đảm bảo tính bảo mật và toàn vẹn, đặc biệt quan trọng khi gửi các thông tin nhạy cảm như API key.



Hình 26: REST API

6.2.3 Định dạng JSON

★JSON (JavaScript Object Notation) là một định dạng văn bản gọn nhẹ, dễ đọc, được sử dụng rộng rãi để trao đổi dữ liệu giữa máy chủ và ứng dụng.

★Khi một API được gọi, máy chủ phản hồi bằng một văn bản định dạng JSON đại diện cho dữ liệu được yêu cầu. Điều này có thể là bất kỳ thứ gì từ thông tin người dùng đến danh sách sản phẩm. Định dạng JSON đảm bảo rằng dữ liệu này có thể dễ dàng được phân tích bởi ứng dụng khách và được sử dụng theo nhu cầu.

- Khi gửi tin nhắn tới Gemini thì chuỗi JSON có định dạng như sau:

Listing 1: Request Body

```
{
  "contents": [
    {
      "parts": [
        {
          "text": "User Request is here !"
        }
      ]
    },
    "generationConfig": {
      "temperature": 0.7
    }
  }
}
```

- Khi Gemini xử lý xong yêu cầu, nó sẽ trả về một chuỗi JSON có dạng:

Listing 2: Response Body

```
{
  "candidates": [
    {
      "content": {
        "parts": [
          {
            "text": "AI Respond is here !"
          }
        ],
        "role": "model"
      },
      "finishReason": "STOP",
      ".....",
      "...."
    }
  ]
}
```

6.3 WebView

Các mô hình AI tạo sinh hiện đại như Gemini thường không trả về văn bản thuần túy (plain text) hoàn toàn. Để làm cho câu trả lời có cấu trúc và dễ đọc hơn (ví dụ: có tiêu đề, danh sách, khái mã lệnh), chúng sử dụng một ngôn ngữ đánh dấu đơn giản đó chính là **Markdown**.

Tuy nhiên do chúng em muốn định dạng hiển thị đẹp hơn, có màu sắc đẹp mắt
=> Markdown không thể đáp ứng được, do đó **WebView** đã được sử dụng.

WebView của JavaFX về cơ bản là một trình duyệt web thu nhỏ. Ngôn ngữ mà nó có thể "đọc" và hiển thị là **HTML**. Do đó nếu đưa trực tiếp chuỗi Markdown (ví dụ: **Xin chào**) trả về của Gemini vào **WebView**, nó sẽ chỉ hiển thị đúng chuỗi ký tự đó, chứ không hiển thị chữ "Xin chào" được in đậm.

⇒ Do đó cần chuyển Markdown về HTML.

6.3.1 HTML

HTML (viết tắt của **HyperText Markup Language** - Ngôn ngữ Đánh dấu Siêu văn bản) là ngôn ngữ tiêu chuẩn được sử dụng để tạo và cấu trúc nội dung trên các trang web.

Một tài liệu HTML được xây dựng từ các **phản tử (elements)**. Một phản tử thường bao gồm:

- **Thẻ mở (Opening tag)**: Ví dụ: <p> để bắt đầu một đoạn văn.
- **Nội dung (Content)**: Văn bản hoặc các phản tử khác nằm bên trong.
- **Thẻ đóng (Closing tag)**: Ví dụ: </p> để kết thúc đoạn văn.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Tiêu đề trang</title>
</head>
<body>
    <h1>Đây là tiêu đề chính</h1>
    <p>Đây là một đoạn văn. Bạn có thể <strong>in đậm</strong> chữ.</p>
    <ul>
        <li>Mục danh sách 1</li>
        <li>Mục danh sách 2</li>
    </ul>
</body>
</html>
```

Hình 27: HTML Syntax

6.3.2 Chuyển đổi Markdown to HTML

Quá trình chuyển đổi không diễn ra đồng thời mà được thực hiện theo một pipeline xử lý tuần tự (sequential processing). Chuỗi Markdown đầu vào sẽ được truyền qua nhiều giai đoạn, mỗi giai đoạn chịu trách nhiệm xử lý một loại cú pháp cụ thể. Thứ tự xử lý được thiết kế cẩn thận như sau:

Tên hàm xử lý	Cú pháp Markdown	HTML tương ứng
processHeaders	# Tiêu đề cấp 1	<h1>Tiêu đề cấp 1</h1>
processCodeBlocks	''' hehe '''	<pre><code>hehe;</code></pre>
processInlineCode	'let a = 1;'	<code>let a = 1;</code>
processBold	**Văn bản in đậm**	Văn bản in đậm
processItalic	*Văn bản in nghiêng*	Văn bản in nghiêng
processBulletLists	* Mục danh sách	Mục danh sách
processNumberedLists	1. Mục danh sách	Mục danh sách
processLineBreaks	Dòng 1 Dòng 2 Dòng 3	<p>Dòng 1 Dòng 2</p><p>Dòng 3</p>

6.4 Socket TCP & Đa luồng

6.4.1 Socket & Giao tiếp mạng

Socket là một khái niệm trong lập trình mạng, cho phép hai thiết bị (client và server) giao tiếp với nhau qua một kết nối mạng. Trong Project này, Socket được sử dụng kết hợp với giao thức **TCP (Transmission Control Protocol)** để tạo ra **kết nối ổn định, lâu dài** giữa ứng dụng JavaFX (phía client) và máy chủ (server).

TCP đảm bảo dữ liệu được truyền đầy đủ, đúng thứ tự, không bị trùng lặp hay mất mát – điều này rất quan trọng trong ứng dụng chat thời gian thực, nơi mọi tin nhắn cần được gửi đi chính xác như người dùng nhập vào.

6.4.2 Giao tiếp Client – Server qua Socket

Hệ thống sử dụng mô hình **Client - Server**, trong đó:

- **Server** sử dụng **ServerSocket** để chờ kết nối đến từ các client.

- Khi một client kết nối, server tạo một kết nối socket riêng và xử lý trong một luồng (thread) độc lập, giúp đảm bảo có thể phục vụ nhiều người dùng cùng lúc mà không bị nghẽn.

Ở phía client, chương trình sử dụng một socket để kết nối với server, sau đó có thể gửi và nhận dữ liệu hai chiều. Nhờ sử dụng TCP, client có thể duy trì kết nối ổn định, không cần thiết lập lại mỗi lần gửi tin nhắn.

★**ServerSocket**: dùng cho phía server, để chờ và chấp nhận kết nối.

Ví dụ:

```
ServerSocket server = new ServerSocket(1234);  
Socket clientSocket = server.accept(); // chờ client kết nối
```

Hình 28: ServerSocket

★**Socket**: dùng cho phía client (hoặc từng client phía server)

```
Socket socket = new Socket("localhost", 1234); // client kết nối
```

Hình 29: Socket

6.4.3 Đa luồng và xử lý đồng thời

Đa luồng là kỹ thuật cho phép chạy **nhiều luồng (thread) song song**, giúp chương trình **xử lý đồng thời nhiều tác vụ**. Để hỗ trợ **nhiều người dùng đồng thời**, mỗi client sau khi kết nối sẽ được xử lý bởi **một luồng riêng biệt** trên server, thông qua một đối tượng xử lý (thường gọi là ClientHandler).

Điều này giúp:

- Không bị treo khi một client gửi/nhận dữ liệu lâu.
- Tăng khả năng mở rộng hệ thống.
- Đảm bảo mọi người dùng đều nhận được phản hồi kịp thời.

Ví dụ:

```

public class ClientHandler extends Thread {
    private Socket clientSocket;
    public void run() {
        // Giao tiếp với client ở đây
    }
}

```

Hình 30: Đa luồng

7 Tổng kết

7.1 Kết quả đã đạt được

7.1.1 Ưu điểm

- Đã xây dựng được ứng dụng hoàn chỉnh có tích hợp công nghệ Socket và REST API.
- Sử dụng đa dạng kiến thức từ học phần IT3103 được thầy Trần Thé Hùng giảng dạy hết sức tâm huyết.
- Báo cáo chính chu , cẩn thận .

7.1.2 Nhược điểm

- Chưa Deploy 1 Server riêng mà hiện tại chỉ chạy trên localhost.
- Chưa có tính năng chat 1-1 mà thay vào đó là 1 room tổng.