**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC THĂNG LONG**

**---o0o---**

**A logo with a red and blue design

Description automatically generatedA**

**Giới Thiệu Công Nghệ WebSocket**

**Hướng dẫn tạo 1 ứng dụng   
Java Spring boot kết hợp Socket.io Client và Chatbot**

**Giáo viên hướng dẫn: Thầy Nguyễn Hùng Cường**

**Sinh viên thực hiện: A42582 Phù Minh Hương**

**A42815 Nguyễn Xuân Phong**

**HÀ NỘI**

**MỤC LỤC**

[CHƯƠNG 1. LỜI MỞ ĐẦU 1](#_Toc177761269)

[CHƯƠNG 2. GIỚI THIỆU TỔNG QUAN 2](#_Toc177761270)

[2.1. Giới thiệu về Java 2](#_Toc177761271)

[2.2. WebSocket là gì ? 4](#_Toc177761272)

[2.3. Giới thiệu về dependency spring-boot-starter-websocket 5](#_Toc177761273)

[CHƯƠNG 3. Thực hiện Cài đặt và Cấu hình cho dự Án 7](#_Toc177761274)

[3.1. Khởi tạo dự án - Starting with Spring Initializr 7](#_Toc177761275)

[3.2. Các dependency cần thiết cho dự án (pom.xml) 7](#_Toc177761276)

[3.3. Backend - ChatMessage 9](#_Toc177761277)

[3.3.1. Configuring WebSocket 9](#_Toc177761278)

[3.3.2. Account Information 11](#_Toc177761279)

[3.3.3. ChatMessage 11](#_Toc177761280)

[3.3.4. ChatRoom 12](#_Toc177761281)

[3.3.5. ChatNotification 12](#_Toc177761282)

[3.3.6. Status 13](#_Toc177761283)

[3.3.7. ChatImg 13](#_Toc177761284)

[3.3.8. ChatRoomService 16](#_Toc177761285)

[3.3.9. ChatMessageService 19](#_Toc177761286)

[3.3.10. UserAccountInfoController 22](#_Toc177761287)

[3.3.11. ChatSocketController 25](#_Toc177761288)

[3.4. Backend – ChatBot 29](#_Toc177761289)

[3.4.1. QuestionAnswer 29](#_Toc177761290)

[3.4.2. QuestionAnswerRepository 29](#_Toc177761291)

[3.4.3. Chatbotservice 31](#_Toc177761292)

[3.4.4. ChatRestController 36](#_Toc177761293)

[3.5. Frontend (JSX – Đầy đủ) 40](#_Toc177761294)

[CHƯƠNG 4. Triển khai và kiểm thử giao diện 65](#_Toc177761295)

[4.1. Chat Bot 65](#_Toc177761296)

[4.2. Chat Message 66](#_Toc177761297)

[CHƯƠNG 5. Tài liệu Tham khảo 69](#_Toc177761298)

# LỜI MỞ ĐẦU

Trong kỷ nguyên số, công nghệ thông tin không ngừng phát triển và thay đổi mạnh mẽ, kéo theo sự bùng nổ của các ứng dụng web hiện đại. Các ứng dụng ngày nay không chỉ đòi hỏi tính năng phong phú mà còn cần đáp ứng yêu cầu về tốc độ và khả năng tương tác thời gian thực. Người dùng mong muốn trải nghiệm các ứng dụng web mượt mà, nhanh chóng, không bị gián đoạn trong quá trình sử dụng, đặc biệt đối với những hệ thống yêu cầu giao tiếp liên tục như trò chuyện trực tuyến, dịch vụ chăm sóc khách hàng tự động, hoặc các ứng dụng làm việc nhóm.

Để đáp ứng những nhu cầu này, các công nghệ hỗ trợ giao tiếp hai chiều giữa máy khách và máy chủ đã ra đời, giúp nâng cao hiệu quả truyền tải dữ liệu và giảm độ trễ trong phản hồi. Trong đó, Java Spring Boot và Socket.io là hai nền tảng mạnh mẽ và phổ biến, cung cấp các giải pháp hiệu quả cho việc xây dựng ứng dụng web có khả năng tương tác thời gian thực. Java Spring Boot là một framework linh hoạt, giúp phát triển các ứng dụng Java dễ dàng và nhanh chóng, trong khi Socket.io là một thư viện mạnh mẽ hỗ trợ kết nối WebSocket giữa máy khách và máy chủ, đảm bảo dữ liệu được truyền đi nhanh chóng và chính xác.

Báo cáo này sẽ hướng dẫn chi tiết cách tạo ra một ứng dụng web sử dụng Java Spring Boot và Socket.io trên phía máy khách, đồng thời tích hợp thêm tính năng chatbot thông minh để nâng cao trải nghiệm người dùng. Trong quá trình này, chúng ta sẽ tìm hiểu cách thức các công nghệ tiên tiến như WebSocket được áp dụng để giải quyết các vấn đề liên quan đến tương tác thời gian thực, đảm bảo hiệu suất và tính ổn định của ứng dụng web.

Để hoàn thành báo cáo này, em xin gửi lời cảm ơn chân thành đến thầy **Nguyễn Hùng Cường**, người đã tận tình hỗ trợ và hướng dẫn chúng em trong suốt quá trình nghiên cứu và thực hiện đề tài. Sự hướng dẫn của thầy không chỉ giúp em hiểu sâu hơn về các khía cạnh kỹ thuật mà còn tạo điều kiện cho em phát triển các kỹ năng cần thiết để giải quyết các vấn đề thực tế trong công nghệ phần mềm. Những kiến thức quý báu mà thầy truyền đạt đã giúp em có được cái nhìn tổng quan và sâu sắc hơn về lĩnh vực phát triển ứng dụng web hiện đại.

# GIỚI THIỆU TỔNG QUAN

## Giới thiệu về Java



Java là một ngôn ngữ lập trình hướng đối tượng và một nền tảng phần mềm được phát triển bởi Sun Microsystems, sau này được Oracle Corporation mua lại. Java ra mắt lần đầu vào năm 1995 và nhanh chóng trở thành một trong những ngôn ngữ lập trình phổ biến nhất trên thế giới.

**Đặc điểm nổi bật của Java:**

1. **Ngôn ngữ hướng đối tượng (OOP)**: Java là một ngôn ngữ lập trình hoàn toàn hướng đối tượng, nghĩa là nó tập trung vào việc sử dụng các đối tượng và lớp để tổ chức và quản lý mã nguồn. Các khái niệm cơ bản trong OOP như kế thừa, đóng gói, đa hình và trừu tượng đều được hỗ trợ mạnh mẽ trong Java.
2. **Độc lập nền tảng (Platform Independence)**: Một trong những khẩu hiệu nổi tiếng của Java là "Write Once, Run Anywhere" (WORA). Mã Java được biên dịch thành mã bytecode, có thể chạy trên bất kỳ nền tảng nào có Java Virtual Machine (JVM) mà không cần phải biên dịch lại mã nguồn.
3. **Quản lý bộ nhớ tự động (Automatic Memory Management)**: Java sử dụng cơ chế thu gom rác (Garbage Collection) để tự động quản lý bộ nhớ, giảm thiểu các lỗi liên quan đến quản lý bộ nhớ thủ công như tràn bộ nhớ hay lỗi tham chiếu null.
4. **Thư viện phong phú (Rich Standard Library)**: Java đi kèm với một bộ thư viện phong phú, cung cấp các công cụ và tiện ích cho các tác vụ phổ biến như thao tác chuỗi, xử lý dữ liệu, kết nối mạng, giao diện đồ họa, và nhiều thứ khác.
5. **Bảo mật (Security)**: Java được thiết kế với nhiều tính năng bảo mật, chẳng hạn như sandboxing (chạy mã trong môi trường an toàn), kiểm tra bytecode trước khi thực thi, và các cơ chế bảo mật tích hợp khác.
6. **Hỗ trợ đa luồng (Multithreading)**: Java hỗ trợ lập trình đa luồng, cho phép các chương trình có thể thực hiện nhiều tác vụ đồng thời, cải thiện hiệu suất và khả năng phản hồi.
7. **Lịch sử và sự phát triển của Java:**

* **1991**: Java được phát triển bởi James Gosling và nhóm các kỹ sư tại Sun Microsystems với tên gọi ban đầu là "Oak". Mục tiêu ban đầu là tạo ra một ngôn ngữ lập trình cho các thiết bị điện tử gia dụng.
* **1995**: Ngôn ngữ này được đổi tên thành Java và chính thức ra mắt. Từ đó, Java nhanh chóng trở thành ngôn ngữ lập trình phổ biến trong phát triển ứng dụng web và doanh nghiệp.
* **2006**: Sun Microsystems công bố mã nguồn của Java dưới giấy phép GPL, biến Java trở thành một dự án mã nguồn mở.
* **2010**: Oracle mua lại Sun Microsystems, và từ đó, Java trở thành sản phẩm của Oracle.

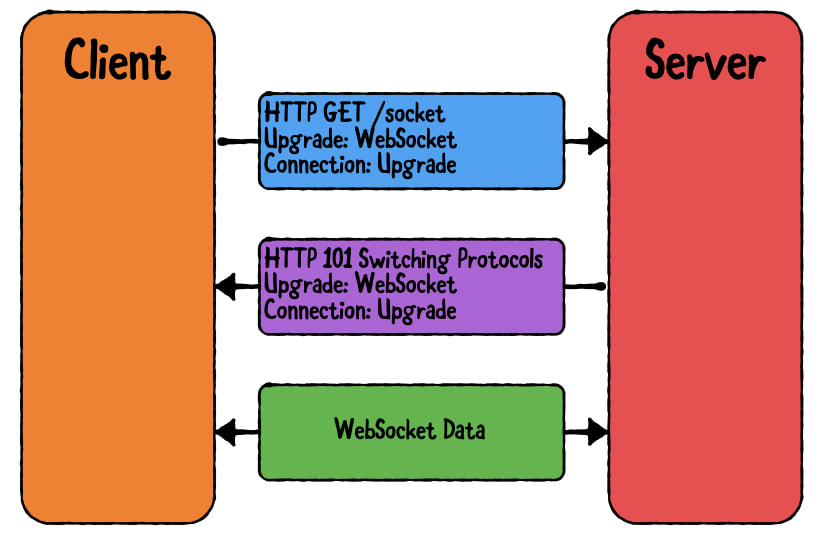
1. **Ứng dụng của Java:**

**Ứng dụng doanh nghiệp**: Java là nền tảng chính trong việc xây dựng các ứng dụng doanh nghiệp nhờ vào khả năng mở rộng, bảo mật, và tính ổn định cao.

1. **Ứng dụng di động**: Java là ngôn ngữ chính được sử dụng để phát triển ứng dụng Android.
2. **Ứng dụng web**: Java hỗ trợ mạnh mẽ việc phát triển các ứng dụng web thông qua các công nghệ như Servlets, JSP, và các framework như Spring, Hibernate.
3. **Ứng dụng nhúng**: Java cũng được sử dụng trong các hệ thống nhúng, chẳng hạn như các thiết bị IoT và các hệ thống điều khiển công nghiệp.
4. **Ưu điểm của Java:**

* **Cộng đồng lớn**: Java có một cộng đồng nhà phát triển đông đảo, với vô số tài liệu, thư viện mã nguồn mở, và framework hỗ trợ.
* **Tính phổ biến**: Java là ngôn ngữ phổ biến trong các công ty công nghệ lớn, đặc biệt là trong các lĩnh vực tài chính, ngân hàng, và bảo hiểm.
* **Khả năng mở rộng**: Java được thiết kế để xử lý các ứng dụng quy mô lớn với hiệu suất cao.

## WebSocket là gì ?



WebSockets là một công nghệ mạnh mẽ cho phép giao tiếp hai chiều, theo thời gian thực giữa máy khách và máy chủ. Chúng được sử dụng rộng rãi trong các ứng dụng như hệ thống trò chuyện, cập nhật trực tiếp và chơi game trực tuyến. Khác với HTTP, nơi máy khách phải gửi yêu cầu và máy chủ phản hồi, WebSocket cho phép máy chủ có thể gửi dữ liệu ngược trở lại máy khách mà không cần yêu cầu từ trước. Điều này làm cho WebSocket rất phù hợp với các ứng dụng thời gian thực như chat, cập nhật trạng thái trực tuyến, game trực tuyến, và nhiều ứng dụng khác yêu cầu sự tương tác nhanh chóng giữa máy khách và máy chủ.. Spring Boot giúp dễ dàng triển khai giao tiếp WebSocket bằng cách cung cấp hỗ trợ tích hợp cho API WebSocket. Trong bài viết này, chúng ta sẽ khám phá cách thiết lập giao tiếp WebSocket trong ứng dụng Spring Boot và tạo một ứng dụng trò chuyện đơn giản làm ví dụ.

**Đặc điểm chính của WebSocket:**

1. **Kết nối hai chiều**: Sau khi kết nối được thiết lập, cả máy khách và máy chủ đều có thể gửi và nhận dữ liệu bất cứ lúc nào.
2. **Hiệu quả**: WebSocket sử dụng một kết nối duy nhất và giữ nó mở, giảm thiểu độ trễ và chi phí tài nguyên so với việc liên tục mở và đóng kết nối như trong HTTP.
3. **Dữ liệu dưới dạng khung (frame)**: Dữ liệu được truyền tải qua các khung nhỏ gọn, giảm thiểu overhead so với HTTP.
4. **Ứng dụng thời gian thực**: WebSocket thường được sử dụng trong các ứng dụng yêu cầu dữ liệu phải được cập nhật liên tục, như ứng dụng chat, hệ thống thông báo, hay dashboard.
5. **Quá trình hoạt động của WebSocket:**

* **Handshake**: Quá trình kết nối bắt đầu với một HTTP handshake thông thường, trong đó máy khách gửi một yêu cầu kết nối đến máy chủ. Nếu máy chủ chấp nhận, nó sẽ nâng cấp kết nối từ HTTP lên WebSocket.
* **Truyền tải dữ liệu**: Sau khi kết nối WebSocket được thiết lập, máy khách và máy chủ có thể trao đổi dữ liệu dưới dạng các khung (frame) cho đến khi một trong hai bên đóng kết nối.

1. **Ưu điểm của WebSocket:**

* **Giảm độ trễ**: Không cần thiết phải thiết lập lại kết nối cho mỗi yêu cầu.
* **Tiết kiệm tài nguyên**: Ít yêu cầu và phản hồi hơn so với HTTP.
* **Dễ dàng triển khai**: Được hỗ trợ bởi nhiều trình duyệt và thư viện, dễ dàng tích hợp vào các ứng dụng web.

1. **Ứng dụng của WebSocket:**

* **Chat ứng dụng**: Người dùng có thể gửi và nhận tin nhắn trong thời gian thực.
* **Thông báo trực tiếp**: Nhận thông báo tức thì từ server mà không cần phải liên tục kiểm tra.
* **Cập nhật dữ liệu trực tiếp**: Ví dụ như giá chứng khoán, số liệu thể thao, hoặc dữ liệu cảm biến.

WebSocket hiện đại là một phần quan trọng trong việc xây dựng các ứng dụng web có yêu cầu về hiệu suất cao và tương tác thời gian thực.

­­

## Giới thiệu về dependency spring-boot-starter-websocket

Ngoài các dependency phổ biến trong web của spring thì tôi sẽ giới thiệu thêm…..spring-boot-starter-websocket là một dependency trong hệ sinh thái Spring Boot, được sử dụng để tạo các ứng dụng hỗ trợ giao tiếp thời gian thực giữa máy khách và máy chủ thông qua giao thức WebSocket. WebSocket cung cấp một kênh truyền thông hai chiều trên một kết nối duy nhất, giúp giảm độ trễ và tăng hiệu suất cho các ứng dụng yêu cầu tương tác liên tục như chat, thông báo trực tiếp, hoặc cập nhật dữ liệu theo thời gian thực.

**Các tính năng chính của spring-boot-starter-websocket:**

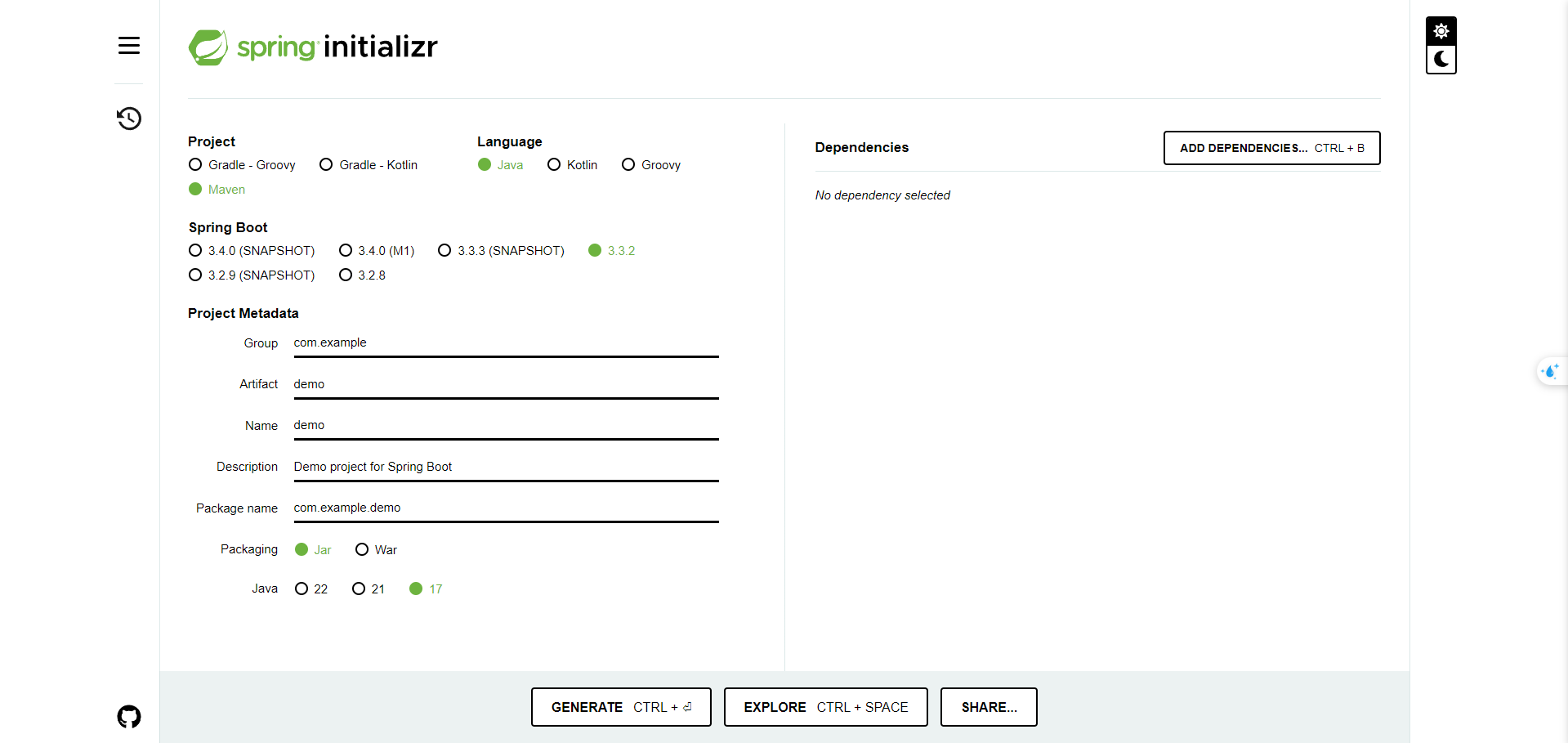
1. **Dễ dàng tích hợp**: spring-boot-starter-websocket giúp tích hợp WebSocket vào ứng dụng Spring Boot một cách dễ dàng bằng cách cung cấp cấu hình mặc định và các công cụ cần thiết để bắt đầu phát triển.
2. **Hỗ trợ STOMP**: Hỗ trợ cho Simple (or Streaming) Text Oriented Messaging Protocol (STOMP), một giao thức nhắn tin đơn giản, được xây dựng trên nền tảng WebSocket. Điều này cho phép xây dựng các ứng dụng nhắn tin phức tạp hơn, với tính năng như định tuyến thông điệp, phân phối thông điệp đến các thuê bao, và nhiều hơn nữa.
3. **Cấu hình tự động**: Spring Boot cung cấp cấu hình tự động cho các thành phần WebSocket, giúp giảm thiểu cấu hình thủ công, giúp bạn tập trung vào việc xây dựng logic kinh doanh.
4. **Tích hợp bảo mật**: Tích hợp dễ dàng với Spring Security để bảo vệ các kết nối WebSocket và đảm bảo rằng chỉ những người dùng được ủy quyền mới có thể truy cập vào các endpoint WebSocket.
5. **Hỗ trợ các API cao cấp**: Spring cung cấp các API tiện lợi để quản lý kết nối WebSocket, xử lý tin nhắn, và tương tác với máy khách thông qua các giao diện dễ sử dụng như @MessageMapping, @SendTo, và SimpMessagingTemplate.
6. **Khả năng mở rộng**: spring-boot-starter-websocket cho phép mở rộng và tùy chỉnh hành vi của WebSocket bằng cách cung cấp các hook và callback, giúp bạn có thể tùy biến cách mà ứng dụng xử lý kết nối và tin nhắn WebSocket.

* **Ứng dụng của spring-boot-starter-websocket:**
* **Ứng dụng chat thời gian thực**: Xây dựng các ứng dụng chat, nơi người dùng có thể gửi và nhận tin nhắn ngay lập tức.
* **Thông báo trực tiếp**: Gửi các thông báo theo thời gian thực đến người dùng khi có sự kiện mới.
* **Cập nhật dữ liệu trực tiếp**: Sử dụng trong các hệ thống cần cập nhật dữ liệu liên tục như bảng giá chứng khoán, số liệu thể thao, hoặc giám sát hệ thống.
* **Game trực tuyến**: Xây dựng các trò chơi nhiều người chơi, nơi dữ liệu cần được cập nhật liên tục giữa các người chơi.

# Thực hiện Cài đặt và Cấu hình cho dự Án

## Khởi tạo dự án - Starting with Spring Initializr

Trước hết, bạn cần tạo một ứng dụng **Spring Boot**



## Các dependency cần thiết cho dự án (pom.xml)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

## Backend - ChatMessage

### Configuring WebSocket

A screenshot of a computer program

Description automatically generated

Sử dụng để cấu hình WebSocket trong Spring Boot, cho phép ứng dụng xử lý giao tiếp thời gian thực giữa client và server thông qua giao thức WebSocket và STOMP (Simple Text Oriented Messaging Protocol).   
***Cấu hình này có ba phương thức chính:***

#### configureMessageBroker(MessageBrokerRegistry registry)

Phương thức này cấu hình cho hệ thống Message Broker (hệ thống quản lý các thông điệp):

* registry.enableSimpleBroker("/user", "/queue"): Kích hoạt một Message Broker nội bộ, có thể hiểu như một nơi trung gian lưu trữ và phân phối các thông điệp cho các client. /user và /queue là các điểm mà client có thể đăng ký để nhận tin nhắn. Những đường dẫn này được dùng để gửi tin nhắn từ server tới client.
* registry.setApplicationDestinationPrefixes("/app"): Định nghĩa một tiền tố để phân biệt các đường dẫn mà client gửi tin nhắn lên server. Bất kỳ tin nhắn nào được gửi từ client có tiền tố /app sẽ được định tuyến tới các method xử lý ở phía server.
* registry.setUserDestinationPrefix("/user"): Xác định tiền tố cho các tin nhắn được gửi đến các người dùng cụ thể. Với tiền tố này, bạn có thể gửi tin nhắn trực tiếp đến một người dùng cụ thể thông qua SimpMessagingTemplate với tiền tố /user/.

#### registerStompEndpoints(StompEndpointRegistry registry)

Phương thức này đăng ký các điểm cuối (endpoint) WebSocket để client kết nối:

* registry.addEndpoint("/ws"): Định nghĩa một endpoint WebSocket tại đường dẫn /ws. Client sẽ kết nối tới server thông qua đường dẫn này.
* .setAllowedOriginPatterns("\*"): Cho phép mọi nguồn (origin) kết nối đến endpoint WebSocket này, không giới hạn nguồn của yêu cầu (cross-origin).
* .withSockJS(): Kích hoạt SockJS fallback, một giải pháp thay thế cho WebSocket khi client hoặc server không hỗ trợ WebSocket. Nếu WebSocket không thể sử dụng, SockJS sẽ dùng các cơ chế thay thế như long polling để đảm bảo kết nối vẫn hoạt động.

#### configureMessageConverters(Liist<MessageConverter> messageConverters)

Phương thức này cấu hình các bộ chuyển đổi thông điệp (message converters) để chuyển đổi giữa đối tượng Java và JSON trong giao tiếp WebSocket:

* DefaultContentTypeResolver resolver = new DefaultContentTypeResolver(); resolver.setDefaultMimeType(MimeTypeUtils.APPLICATION\_JSON);: Thiết lập bộ phân giải kiểu nội dung mặc định để tất cả các tin nhắn sẽ được xử lý dưới dạng JSON (MIME type application/json).
* MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();:Bộ chuyển đổi MappingJackson2MessageConverter sẽ chuyển đổi các thông điệp JSON thành các đối tượng Java và ngược lại, sử dụng thư viện Jackson để xử lý JSON.
* messageConverters.add(converter);: Thêm bộ chuyển đổi JSON vào danh sách các bộ chuyển đổi tin nhắn.
* return false;: Trả về false để giữ lại các bộ chuyển đổi mặc định của Spring, cùng với bộ chuyển đổi tùy chỉnh vừa được thêm vào.

### Account Information

A screen shot of a computer program

Description automatically generated

### ChatMessage

A screen shot of a computer program

Description automatically generated

### ChatRoom

A screen shot of a computer program

Description automatically generated

### ChatNotification

A screen shot of a computer program

Description automatically generated

### Status

A screenshot of a computer

Description automatically generated

### ChatImg

A screen shot of a computer program

Description automatically generated

Các Dtos ở trên là những đối tượng được lưu trong chatmessage đồng thời là thứ quan trọng nhất để giúp ta quản lý các đối tượng chatmessage và chatbot (ở dưới)



**Function saveUser(UserAccountInfo user) Method**

* **Purpose**: This method saves a new user or updates the status and role of an existing user.
* **Steps**:
  1. It first checks if a user with the same name already exists in the database using the findUserTestByName method.
  2. If a user with the same name exists (check != null):
     + The method updates the existing user's status to ONLINE and their role to the one provided in the user object.
     + The existing user's image URL remains unchanged, as setImg\_url is not called again in this branch.
     + The updated user information is saved back to the database.
  3. If no user with the same name exists:
     + The new user's status is set to ONLINE.
     + The method then saves the new user to the database.

A screen shot of a computer code

Description automatically generated

**Function disconnect(UserAccountInfo user) Method**

* **Purpose**: Phương pháp này có nhiệm vụ đánh dấu người dùng là OFFLINE.
* **Steps**:
  1. Tìm kiếm người dùng theo tên trong cơ sở dữ liệu.
  2. Nếu tìm thấy người dùng (storedUser != null):
     + Trạng thái của người dùng được đặt thành OFFLINE.
     + Thông tin người dùng đã cập nhật được lưu lại vào cơ sở dữ liệu.

A computer screen shot of a program code

Description automatically generated

* **findConnectedUsers() Method**
* **Purpose**: Phương thức này lấy tất cả người dùng từ cơ sở dữ liệu, phân loại họ theo vai trò (QUẢN TRỊ VIÊN hoặc NGƯỜI DÙNG) và trạng thái (TRỰC TUYẾN hoặc NGOẠI TUYẾN).
* **Steps**:
  1. Phương thức lấy danh sách người dùng theo vai trò và trạng thái::
     + Người dùng ADMIN đang TRỰC TUYẾN.
     + Người dùng USER đang TRỰC TUYẾN.
     + Người dùng ADMIN đang NGOẠI TUYẾN.
     + Người dùng USER đang NGOẠI TUYẾN.
  2. Sau đó, nó kết hợp các danh sách này thành một danh sách duy nhất (lưu) và trả về

**Tóm tắt**

* **saveUser**: Đảm bảo người dùng được lưu dưới dạng mới hoặc được cập nhật nếu họ đã tồn tại, tập trung vào việc thiết lập trạng thái và vai trò của họ.
* **disconnect**: Cập nhật trạng thái của người dùng thành OFFLINE.
* **findConnectedUsers**: Truy xuất và phân loại người dùng dựa trên vai trò và trạng thái của họ, trả về danh sách người dùng toàn diện.

Được thiết kế để quản lý các phiên người dùng và vai trò của họ trong một ứng dụng, duy trì trạng thái của người dùng theo thời gian thực.

### ChatRoomService



Lớp ChatRoomService cung cấp các dịch vụ liên quan đến việc quản lý phòng chat trong một ứng dụng chat. Hãy phân tích từng chức năng:

**Function getChatRoomId(String senderId, String recipientId, boolean createNewRoomIfNotExists)**

Phương thức này lấy ID của phòng chat cho một cuộc trò chuyện giữa hai người dùng, được xác định bởi senderId và recipientId. Nếu phòng chat không tồn tại và createNewRoomIfNotExists là true, nó sẽ tạo một ID phòng chat mới.

* **Tham số:**
  + senderId: ID của người gửi.
  + recipientId: ID của người nhận.
  + createNewRoomIfNotExists: Một cờ boolean chỉ định có tạo một phòng chat mới nếu phòng chat chưa tồn tại hay không.
* **Logic:**
  + Phương thức này đầu tiên cố gắng tìm phòng chat bằng cách gọi chatRoomRepository.findBySenderIdAndRecipientId(senderId, recipientId).
  + Nếu tìm thấy phòng chat, nó sẽ ánh xạ đến chatId bằng cách sử dụng .map(ChatRoom::getChatId).
  + Nếu phòng chat không tồn tại (Optional.empty() được trả về), phương thức kiểm tra cờ createNewRoomIfNotExists.
  + Nếu createNewRoomIfNotExists là true, nó tạo một ID phòng chat mới bằng cách sử dụng phương thức createChatId và trả về nó dưới dạng Optional.
  + Nếu createNewRoomIfNotExists là false, nó chỉ trả về Optional.empty().
* **Trả về:**
  + Một Optional<String> chứa chatId nếu tìm thấy hoặc được tạo, hoặc Optional.empty() nếu không có phòng chat nào tồn tại và không tạo phòng mới.

A computer screen shot of a program

Description automatically generated

**Function createChatId(String senderId, String recipientId)**

Phương thức này tạo một ID phòng chat mới và lưu các thực thể ChatRoom tương ứng vào cơ sở dữ liệu cho cả hai chiều của cuộc trò chuyện (người gửi - người nhận và người nhận - người gửi).

* **Tham số:**
  + senderId: ID của người gửi.
  + recipientId: ID của người nhận.
* **Logic:**
  + Phương thức này tạo một chatId bằng cách định dạng senderId và recipientId thành một chuỗi duy nhất, được phân tách bằng dấu gạch dưới (ví dụ: senderId\_recipientId).
  + Sau đó, nó tạo hai thực thể ChatRoom:
    1. senderRecipient: Đại diện cho phòng chat từ góc nhìn của người gửi.
    2. recipientSender: Đại diện cho phòng chat từ góc nhìn của người nhận.
  + Cả hai thực thể này đều được lưu vào repository (chatRoomRepository.save()), đảm bảo rằng cuộc trò chuyện có thể được truy cập từ góc nhìn của bất kỳ người dùng nào.
* **Trả về:**
  + Phương thức này trả về chuỗi chatId đã được tạo và lưu.
* **Điểm chính:**
* **Xử lý tính dư thừa**: Bằng cách lưu trữ cả hai chiều của cuộc trò chuyện (senderRecipient và recipientSender), phương thức đảm bảo rằng bất kỳ người dùng nào cũng có thể tìm thấy phòng chat một cách dễ dàng.
* **Tính linh hoạt**: Cờ createNewRoomIfNotExists trong phương thức getChatRoomId mang lại tính linh hoạt trong việc xử lý các trường hợp mà phòng chat có thể không tồn tại, cho phép ứng dụng quyết định có tạo mới một phòng hay không.
* **Sử dụng Optional**: Việc sử dụng Optional giúp tránh các giá trị null và làm cho mã nguồn trở nên rõ ràng hơn về khả năng không có giá trị.

Thiết kế này rất hiệu quả cho các ứng dụng chat, nơi mà việc tra cứu nhanh chóng và tạo phòng chat là cần thiết, đồng thời tránh được các vấn đề về việc thiếu hoặc dư thừa các mục phòng chat.

### ChatMessageService

A screen shot of a computer program

Description automatically generated

Phân tích lớp ChatMessageService từng phương thức:

**Function save(ChatMessage chatMessage)**

Phương thức này lưu một tin nhắn chat mới vào cơ sở dữ liệu.

* **Logic:**
  + Lấy chatId cho tin nhắn bằng cách gọi chatRoomService.getChatRoomId(). Nếu không tìm thấy phòng chat, nó sẽ tạo mới (do cờ true).
  + Thiết lập chatId cho tin nhắn và đảm bảo rằng hình ảnh của người gửi (senderImage) được đặt đúng cách.
  + Lưu tin nhắn vào cơ sở dữ liệu thông qua repository.
* **Trả về:**
  + Trả về đối tượng ChatMessage sau khi đã được lưu.
* **Chú ý:**
  + Nếu không tìm thấy phòng chat, phương thức sẽ ném ra một ngoại lệ (orElseThrow), có thể thay thế bằng một ngoại lệ tuỳ chỉnh.

A screen shot of a computer code

Description automatically generated

**Function update(Integer id, String reContent)**

Phương thức này cập nhật nội dung của một tin nhắn đã tồn tại.

* **Logic:**
  + Tìm tin nhắn theo ID.
  + Nếu tin nhắn tồn tại, nó sẽ cập nhật nội dung (reContent) và thiết lập hành động là "UPDATE".
  + Lưu tin nhắn đã cập nhật vào cơ sở dữ liệu.
* **Trả về:**
  + Trả về đối tượng ChatMessage đã được cập nhật, hoặc null nếu không tìm thấy tin nhắn.

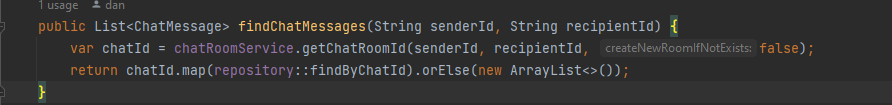
A computer screen with text

Description automatically generated

**Function delete(Integer id)**

Phương thức này xử lý việc "thu hồi" hoặc xoá một tin nhắn chat.

* **Logic:**
  + Tìm tin nhắn bằng ID.
  + Nếu tin nhắn tồn tại và nội dung không phải là thông báo đã thu hồi, đồng thời người gửi không phải là Chat Bot, nó sẽ cập nhật nội dung tin nhắn thành thông báo người gửi đã thu hồi tin nhắn và thiết lập hành động là "DELETE".
  + Nếu tin nhắn đã bị thu hồi trước đó hoặc người gửi là Chat Bot, tin nhắn sẽ bị xoá hoàn toàn khỏi cơ sở dữ liệu.
* **Trả về:**
  + Trả về đối tượng ChatMessage đã cập nhật hoặc sau khi tin nhắn bị xoá.
* **Chú ý:**
  + Hành động "thu hồi" không thực sự xoá tin nhắn, mà chỉ cập nhật nội dung tin nhắn thành thông báo.



**Function findChatMessages(String senderId, String recipientId)**

Phương thức này tìm tất cả các tin nhắn giữa một người gửi và một người nhận.

* **Logic:**
  + Lấy chatId từ chatRoomService. Nếu không tìm thấy phòng chat, trả về một danh sách trống.
  + Nếu tìm thấy, sử dụng chatId để tìm các tin nhắn tương ứng từ repository.
* **Trả về:**
  + Trả về danh sách các tin nhắn giữa senderId và recipientId. Nếu không có phòng chat, trả về một danh sách trống.

**Tổng quan:**

* **Tính nhất quán**: Mỗi phương thức đều thực hiện các thao tác dựa trên ChatMessage và ChatRoom.
* **Xử lý hợp lý**: Các thao tác thu hồi và cập nhật tin nhắn đều được thực hiện thông qua các cập nhật trạng thái (không xoá hoàn toàn trừ khi cần thiết).
* **Lưu ý bảo trì**: Ngoại lệ cần được tuỳ chỉnh để rõ ràng hơn, tránh ném ngoại lệ chung chung.

### UserAccountInfoController

A screen shot of a computer program

Description automatically generated

**Function addUser(@Payload UserAccountInfo userTest)**

Phương thức này xử lý việc thêm một người dùng mới khi nhận được thông điệp qua WebSocket.

* **Annotations:**
  + @MessageMapping("/addUser"): Gắn phương thức với một điểm cuối WebSocket tại /addUser. Khi một thông điệp được gửi đến điểm này, phương thức sẽ được kích hoạt.
  + @SendTo("/user/queue/public"): Sau khi xử lý, kết quả sẽ được gửi đến địa chỉ /user/queue/public, nơi mà các người dùng khác có thể nhận được thông tin về người dùng mới.
* **Logic:**
  + Gọi phương thức saveUser từ userService để lưu thông tin người dùng.
  + Trả về đối tượng UserAccountInfo đã được lưu.
* **Xử lý ngoại lệ:**
  + Nếu có lỗi trong quá trình lưu, phương thức có thể ném ra một IllegalArgumentException.

A screen shot of a computer code

Description automatically generated

**Function disconnectUser(@Payload UserAccountInfo userTest)**

Phương thức này xử lý việc ngắt kết nối một người dùng khi nhận được thông điệp qua WebSocket.

* **Annotations:**
  + @MessageMapping("/disconnectUser"): Gắn phương thức với một điểm cuối WebSocket tại /disconnectUser.
  + @SendTo("/user/queue/public"): Sau khi xử lý, kết quả sẽ được gửi đến địa chỉ /user/queue/public.
* **Logic:**
  + Gọi phương thức disconnect từ userService để ngắt kết nối người dùng.
  + Trả về đối tượng UserAccountInfo của người dùng đã ngắt kết nối.
* **Xử lý ngoại lệ:**
  + Nếu có lỗi trong quá trình ngắt kết nối, phương thức có thể ném ra một IllegalArgumentException.

A screen shot of a computer

Description automatically generated

**Function findConnectedUsers()**

Phương thức này xử lý yêu cầu HTTP GET để lấy danh sách tất cả người dùng đang kết nối.

* **Annotations:**
  + @GetMapping("/users"): Gắn phương thức với một điểm cuối HTTP GET tại /users.
* **Logic:**
  + Gọi phương thức findConnectedUsers từ userService để lấy danh sách người dùng đang kết nối.
  + Trả về một ResponseEntity chứa danh sách người dùng với trạng thái HTTP 200 (OK).
* **Trả về:**
  + Một ResponseEntity<List<UserAccountInfo>> chứa danh sách người dùng đang kết nối.

**Tổng quan:**

* **Xử lý WebSocket:** Các phương thức addUser và disconnectUser xử lý việc thêm và ngắt kết nối người dùng thông qua WebSocket, sau đó gửi thông báo đến các người dùng khác về trạng thái của người dùng đó.
* **Xử lý HTTP:** Phương thức findConnectedUsers cung cấp một điểm cuối HTTP để lấy danh sách người dùng đang kết nối, giúp hiển thị danh sách này trên giao diện người dùng.
* **Phân tách chức năng rõ ràng:** Mỗi phương thức có một nhiệm vụ cụ thể, giúp mã dễ hiểu và bảo trì.

### ChatSocketController

A computer screen shot of a program

Description automatically generated

**Function processMessage(@Payload ChatMessage chatMessage)**

Phương thức này xử lý các thông điệp chat nhận được qua WebSocket và phân loại hành động để xử lý tương ứng.

* **Annotations:**
  + @MessageMapping("/chat"): Gắn phương thức với một điểm cuối WebSocket tại /chat.
* **Logic:**
  + Kiểm tra hành động (action) của tin nhắn chat và gọi phương thức xử lý tương ứng:
    - "SEND": Gọi handleSendMessage(chatMessage).
    - "UPDATE": Gọi handleUpdateMessage(chatMessage).
    - "DELETE": Gọi handleDeleteMessage(chatMessage).
  + Nếu hành động không xác định, ném ngoại lệ IllegalArgumentException.

A screen shot of a computer program

Description automatically generated

**Function handleSendMessage(ChatMessage chatMessage)**

Phương thức này xử lý việc gửi một tin nhắn mới.

* **Logic:**
  + Lưu tin nhắn vào cơ sở dữ liệu bằng chatMessageService.save(chatMessage).
  + Gửi thông báo đến người nhận thông qua messagingTemplate.convertAndSendToUser() với thông tin của tin nhắn vừa được lưu.
* **Chú ý:**
  + Phương thức này gửi thông báo đến /queue/messages của người nhận, chứa thông tin về tin nhắn đã gửi.

A screen shot of a computer program

Description automatically generated

**Function handleUpdateMessage(ChatMessage chatMessage)**

Phương thức này xử lý việc cập nhật nội dung của một tin nhắn.

* **Logic:**
  + Cập nhật tin nhắn trong cơ sở dữ liệu bằng chatMessageService.update(chatMessage.getId(), chatMessage.getContent()).
  + Gửi thông báo cập nhật đến người nhận với nội dung mới của tin nhắn.
* **Chú ý:**
  + Phương thức này cũng gửi thông báo đến /queue/messages của người nhận, nhưng với nội dung đã được cập nhật.

A screen shot of a computer program

Description automatically generated

**Function handleDeleteMessage(ChatMessage chatMessage)**

Phương thức này xử lý việc xoá (hoặc thu hồi) một tin nhắn.

* **Logic:**
  + Thu hồi hoặc xoá tin nhắn trong cơ sở dữ liệu bằng chatMessageService.delete(chatMessage.getId()).
  + Gửi thông báo đến người nhận rằng tin nhắn đã bị xoá, cùng với thông báo "This message was deleted".
* **Chú ý:**
  + Phương thức này gửi thông báo đến /queue/messages của người nhận, cho biết rằng tin nhắn đã bị xoá.

A screen shot of a computer program

Description automatically generated

**Function getResponsec(@PathVariable String senderId)**

Phương thức này xử lý yêu cầu từ người dùng với một senderId cụ thể để lấy phản hồi từ chatbot.

* **Annotations:**
  + @GetMapping("/chat/{senderId}"): Gắn phương thức với một điểm cuối HTTP GET tại /chat/{senderId}.
* **Logic:**
  + Gọi phương thức chatBotService.handleInput(senderId) và trả về phản hồi từ chatbot.
* **Trả về:**
  + Chuỗi phản hồi từ chatbot.

**Function findChatMessages(@PathVariable String senderId, @PathVariable String recipientId)**

Phương thức này xử lý yêu cầu HTTP GET để lấy tất cả các tin nhắn giữa hai người dùng cụ thể.

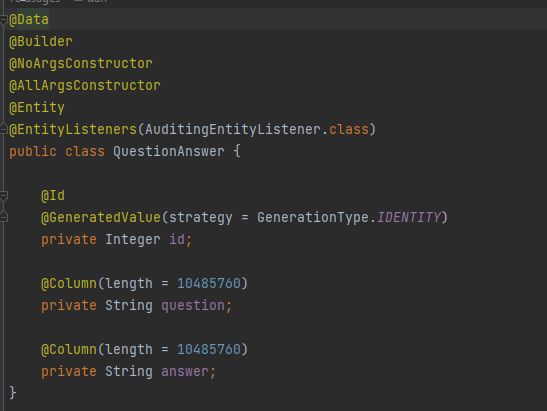
* **Annotations:**
  + @GetMapping("/messages/{senderId}/{recipientId}"): Gắn phương thức với một điểm cuối HTTP GET tại /messages/{senderId}/{recipientId}.
* **Logic:**
  + Gọi phương thức chatMessageService.findChatMessages(senderId, recipientId) để tìm các tin nhắn giữa hai người dùng.
  + Trả về danh sách các tin nhắn trong một ResponseEntity với trạng thái HTTP 200 (OK).
* **Trả về:**
  + ResponseEntity<List<ChatMessage>> chứa danh sách các tin nhắn giữa người gửi và người nhận.

**Tổng quan:**

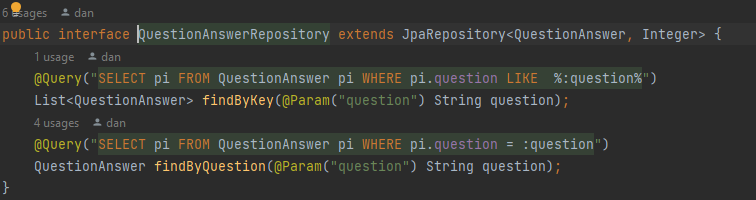
* **Xử lý đa dạng:** Lớp này cung cấp các chức năng xử lý cả giao tiếp WebSocket (gửi, cập nhật, xoá tin nhắn) và HTTP (lấy danh sách tin nhắn, giao tiếp với chatbot).
* **Tính mở rộng:** Các phương thức có thể dễ dàng mở rộng hoặc tùy chỉnh để phục vụ các mục đích khác, như thêm các hành động mới cho tin nhắn hoặc mở rộng tính năng chatbot.
* **Phân tách hợp lý:** Các phương thức xử lý từng loại hành động riêng biệt, giúp mã dễ hiểu và bảo trì.

## Backend – ChatBot

### QuestionAnswer



### QuestionAnswerRepository



**Function List<QuestionAnswer> findByKey(@Param("question") String question)**

Phương thức này tìm kiếm tất cả các bản ghi trong bảng QuestionAnswer mà trường question chứa chuỗi tìm kiếm được truyền vào.

* **@Query**: Định nghĩa truy vấn HQL (Hibernate Query Language) tùy chỉnh.
* Truy vấn này tìm kiếm tất cả các đối tượng QuestionAnswer có thuộc tính question chứa chuỗi con mà bạn truyền vào (sử dụng cú pháp %:question% trong câu lệnh LIKE để thực hiện tìm kiếm chuỗi con).
* **@Param("question")**: Tham số question được gắn vào truy vấn để xác định giá trị cho phép so sánh trong câu LIKE.
* **Kết quả**: Trả về danh sách các bản ghi (List<QuestionAnswer>) mà question chứa chuỗi tìm kiếm.

**Function QuestionAnswer findByQuestion(@Param("question") String question)**

* Phương thức này tìm kiếm một bản ghi trong bảng QuestionAnswer mà trường question khớp hoàn toàn với chuỗi tìm kiếm.
* **@Query**: Định nghĩa truy vấn HQL để tìm kiếm chính xác một bản ghi:
* Truy vấn này tìm kiếm đối tượng QuestionAnswer có thuộc tính question khớp chính xác với chuỗi bạn truyền vào (= được dùng cho so sánh chính xác).
* **@Param("question")**: Tham số question truyền vào truy vấn để so sánh trong câu WHERE.
* **Kết quả**: Trả về một đối tượng QuestionAnswer duy nhất nếu tìm thấy, hoặc null nếu không có kết quả nào.

### Chatbotservice

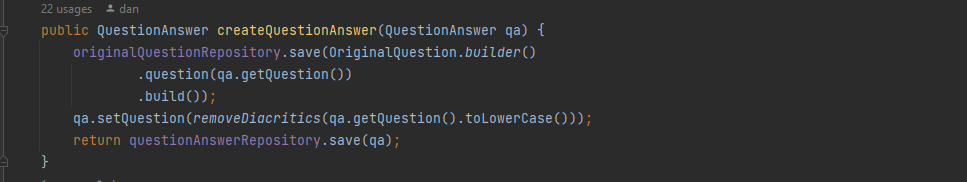


**Các repository:**

* OriginalQuestionRepository: Quản lý các câu hỏi gốc được lưu vào cơ sở dữ liệu.
* QuestionAnswerRepository: Quản lý việc lưu trữ và truy vấn các câu trả lời cho câu hỏi.
* QuestionOfGuestRepository: Quản lý các câu hỏi từ khách hàng (khách truy cập) và câu trả lời tương ứng.
* UserRepository: Quản lý thông tin người dùng trong hệ thống.

**Phương thức handleInput(String message)**

* Nhận vào một tin nhắn từ người dùng và tìm kiếm các câu trả lời có liên quan bằng cách sử dụng phương thức findByKey từ QuestionAnswerRepository. Chuỗi tìm kiếm được xử lý để loại bỏ dấu (sử dụng removeDiacritics) và chuyển thành chữ thường để tránh phân biệt chữ hoa/chữ thường.
* Nếu tìm thấy một câu trả lời, nó sẽ được trả về. Nếu có nhiều câu trả lời, sẽ trả về một câu trả lời kèm thông báo về số lượng kết quả tìm thấy.
* Nếu tin nhắn chứa sticker, trả về một thông báo đặc biệt.
* Nếu không có câu trả lời, trả về thông báo không có câu trả lời.



**Phương thức createQuestionAnswer(QuestionAnswer qa)**

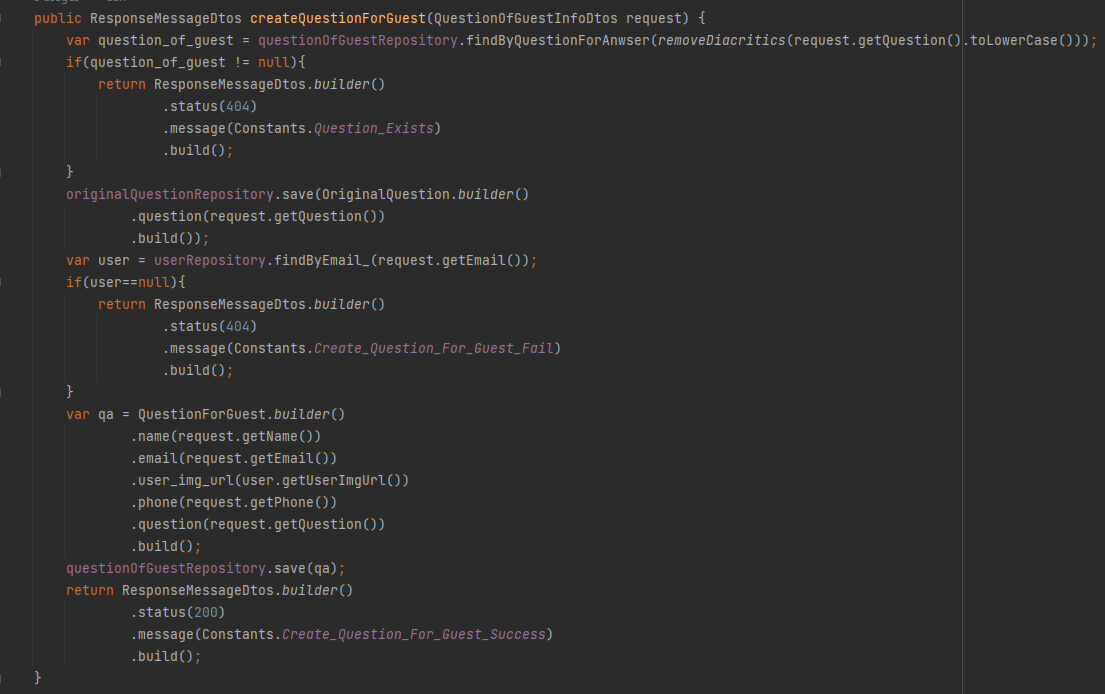
* Tạo mới một câu hỏi và câu trả lời.
* Câu hỏi gốc sẽ được lưu vào bảng OriginalQuestion.
* Sau đó, câu hỏi sẽ được xử lý để loại bỏ dấu và chuyển thành chữ thường, và câu trả lời sẽ được lưu vào QuestionAnswerRepository.



**Phương thức createQuestionAnswerForGuest(QuestionAnswer request)**

**(Nâng cao)**

* Tạo câu trả lời cho câu hỏi từ khách truy cập.
* Tìm kiếm câu hỏi của khách truy cập bằng phương thức findByQuestionForAnwser. Nếu câu hỏi không tồn tại hoặc đã có câu trả lời, trả về thông báo lỗi tương ứng.
* Tìm kiếm thông tin người dùng dựa trên email. Nếu không tìm thấy người dùng, trả về thông báo lỗi.
* Lưu câu trả lời vào bảng QuestionAnswer và cập nhật câu hỏi của khách truy cập với câu trả lời và hình ảnh người dùng.



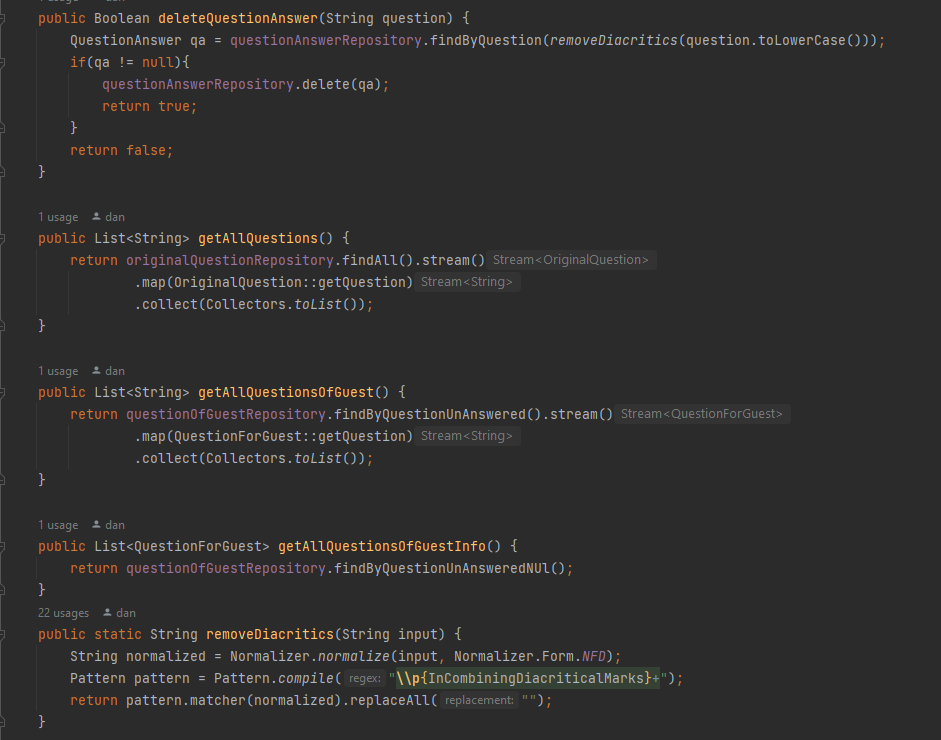
**Phương thức createQuestionForGuest(QuestionOfGuestInfoDtos request)   
(Nâng cao)**

* Tạo mới một câu hỏi từ khách truy cập.
* Kiểm tra xem câu hỏi đã tồn tại chưa, nếu có thì trả về thông báo lỗi.
* Lưu câu hỏi gốc vào OriginalQuestionRepository.
* Tìm kiếm thông tin người dùng theo email và tạo một câu hỏi từ khách với thông tin người dùng và lưu vào bảng QuestionForGuest.



**Phương thức createManyQuestionAnswer(ManyQuestionAnswerDtos qa)**

* Tạo nhiều câu hỏi và câu trả lời cùng một lúc.
* Duyệt qua danh sách các câu hỏi và câu trả lời, xử lý từng câu hỏi để loại bỏ dấu và lưu vào QuestionAnswerRepository.



**Phương thức updateQuestionAnswer(String question, String newAnswer)**

* Tìm kiếm câu hỏi trong cơ sở dữ liệu bằng phương thức findByQuestion.
* Nếu tìm thấy câu hỏi, cập nhật câu trả lời mới và lưu lại vào cơ sở dữ liệu.

**Phương thức deleteQuestionAnswer(String question)**

* Tìm kiếm câu hỏi trong cơ sở dữ liệu.
* Nếu tìm thấy, xóa câu hỏi khỏi cơ sở dữ liệu.

**Phương thức getAllQuestions()**

* Truy vấn tất cả các câu hỏi gốc từ bảng OriginalQuestion và trả về danh sách câu hỏi.

**Phương thức getAllQuestionsOfGuest()**

* Truy vấn tất cả các câu hỏi chưa có câu trả lời từ khách truy cập và trả về danh sách câu hỏi.

**Phương thức getAllQuestionsOfGuestInfo()**

* Trả về danh sách các câu hỏi từ khách chưa được trả lời kèm thông tin chi tiết.

**Phương thức removeDiacritics(String input)**

* Loại bỏ dấu (diacritics) từ chuỗi ký tự. Phương thức này dùng Normalizer để chuẩn hóa chuỗi và sau đó dùng Pattern để xóa các dấu ký tự.

### ChatRestController

A screen shot of a computer program

Description automatically generated

**Thuộc tính phụ thuộc:**

* EmailSenderService: Dịch vụ gửi email được sử dụng để gửi email thông báo trả lời cho các câu hỏi của khách truy cập.
* ChatbotService: Lớp dịch vụ chính chịu trách nhiệm xử lý các yêu cầu liên quan đến chatbot, bao gồm việc tạo, cập nhật, xóa và lấy thông tin câu hỏi/câu trả lời.
* QuestionOfGuestRepository: Repository để truy vấn các câu hỏi từ khách truy cập và xử lý các câu trả lời tương ứng.

**Các phương thức chính:**

**Phương thức questions()**

* **Phương thức:** GET /getAllQuestions
* **Chức năng:** Trả về danh sách tất cả các câu hỏi đã được lưu trữ trong cơ sở dữ liệu từ repository OriginalQuestion.
* **Trả về:** Danh sách các câu hỏi (kiểu List<String>).

**Phương thức chat(RequestMessageChatBotDtos message)**

* **Phương thức:** POST /chat
* **Chức năng:** Nhận tin nhắn từ người dùng, sau đó sử dụng ChatbotService để xử lý và trả về câu trả lời tương ứng từ hệ thống chatbot.
* **Trả về:** Câu trả lời của chatbot dưới dạng chuỗi.

**Phương thức createQuestionAnswer(QuestionAnswer qa)**

* **Phương thức:** POST /qa
* **Chức năng:** Tạo mới một câu hỏi và câu trả lời. Phương thức này sẽ lưu câu hỏi và câu trả lời vào cơ sở dữ liệu thông qua ChatbotService.
* **Trả về:** Đối tượng QuestionAnswer vừa được tạo.

**Phương thức createManyQuestionAnswer(ManyQuestionAnswerDtos qa)**

* **Phương thức:** POST /many-qa
* **Chức năng:** Tạo nhiều câu hỏi và câu trả lời cùng lúc. Nhận một đối tượng DTO chứa danh sách các câu hỏi và câu trả lời, sau đó lưu chúng vào cơ sở dữ liệu.
* **Trả về:** Danh sách các đối tượng QuestionAnswer đã được lưu.

A computer screen shot of a program

Description automatically generated

**Phương thức updateQuestionAnswer(QuestionAnswer request)**

* **Phương thức:** PUT /qa
* **Chức năng:** Cập nhật câu trả lời cho một câu hỏi đã tồn tại. Phương thức này nhận vào câu hỏi và câu trả lời mới, sau đó cập nhật trong cơ sở dữ liệu.
* **Trả về:** Đối tượng QuestionAnswer sau khi được cập nhật.

**Phương thức deleteQuestionAnswer(RequestMessageChatBotDtos question)**

* **Phương thức:** DELETE /qa
* **Chức năng:** Xóa một câu hỏi và câu trả lời tương ứng khỏi cơ sở dữ liệu dựa trên nội dung câu hỏi được gửi lên.
* **Trả về:** Đối tượng ResponseMessageDtos chứa thông tin về kết quả (thành công hoặc thất bại) của quá trình xóa.

A computer screen shot of a program code

Description automatically generated

**3.Phần dành cho khách truy cập (Guest):**

**Phương thức questionsOfGuest()**

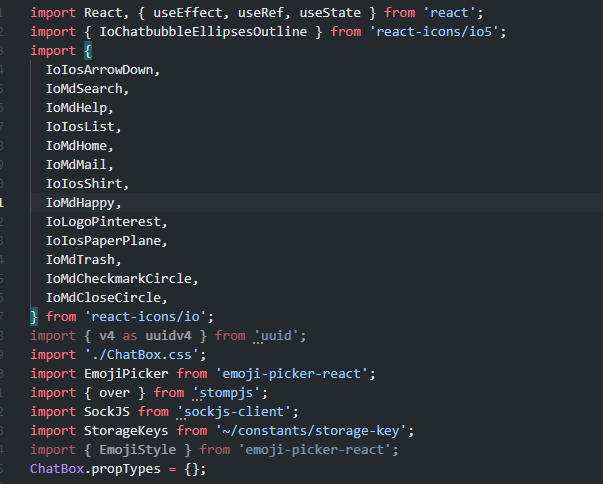
* **Phương thức:** GET /getAllQuestionsOfGuest
* **Chức năng:** Lấy danh sách tất cả các câu hỏi chưa được trả lời của khách truy cập.
* **Trả về:** Danh sách câu hỏi (kiểu List<String>).
* **3.2 Phương thức questionsOfGuestInfo()**
* **Phương thức:** GET /getAllQuestionsOfGuestInfor
* **Chức năng:** Lấy danh sách thông tin chi tiết về các câu hỏi chưa được trả lời của khách truy cập.
* **Trả về:** Danh sách các đối tượng QuestionForGuest.

**Phương thức createQuestionAnswerOfGuest(QuestionAnswer qa)**

* **Phương thức:** POST /qa-answer-guest
* **Chức năng:** Tạo câu trả lời cho câu hỏi của khách truy cập. Nếu câu trả lời được tạo thành công, một email sẽ được gửi đến khách truy cập thông qua dịch vụ EmailSenderService.
* **Trả về:** Đối tượng ResponseMessageDtos chứa thông tin về kết quả (thành công hoặc thất bại) của quá trình tạo câu trả lời.

------------------------------------------------

## Frontend (JSX – Đầy đủ)



 **React and Hooks:** useEffect, useRef và useState là các hook React để quản lý vòng đời thành phần, truy cập các phần tử DOM và quản lý trạng thái.

 **Icons:** Nhiều biểu tượng khác nhau từ react-icons/io được nhập cho các thành phần UI.

 **UUID:** uuidv4 được sử dụng để tạo ra các mã định danh duy nhất.

 **CSS:** ChatBox.css được nhập để tạo kiểu cho thành phần trò chuyện.

 **STOMP and SockJS:** Những thư viện này được sử dụng để liên lạc WebSocket giữa máy khách và máy chủ.

 **StorageKeys:** Một mô-đun cố định tùy chỉnh để truy cập các khóa lưu trữ (như TOKEN và USER).

A computer screen shot of a program code

Description automatically generated

let save\_ReiverID\_unreadMessages = '';

* **RECEIVER:** Xác định tên được sử dụng cho người nhận cụ thể như chatbot và quản trị viên.
* **WebSocket and State Variables:** Thiết lập cho URL WebSocket, đối tượng máy khách và nhiều biến trạng thái khác nhau được sử dụng trong ứng dụng trò chuyện.

A screen shot of a computer code

Description automatically generated

 **State Hooks:** Quản lý nhiều trạng thái khác nhau như người dùng có đang trò chuyện không, người nhận được chọn, danh sách người nhận, tin nhắn hiện tại và lịch sử tin nhắn.

 **useRef:** Tham chiếu đến cuối danh sách tin nhắn, được sử dụng để cuộn đến cuối cuộc trò chuyện khi có tin nhắn mới.

**Emoji Picker:** Trạng thái và trình xử lý để hiển thị/ẩn bộ chọn biểu tượng cảm xúc và thêm các biểu tượng cảm xúc đã chọn vào tin nhắn hiện tại.

A computer code with text

Description automatically generated with medium confidence

Chức năng isImageUrl trong mã để kiểm tra xem một URL có phải là liên kết đến hình ảnh hay không bằng cách so sánh phần mở rộng của URL với các định dạng hình ảnh phổ biến như .jpeg, .jpg, .gif, và .png.

A computer screen shot of a program code

Description automatically generated

 **Options Handling:** Trạng thái và chức năng để quản lý khả năng hiển thị và lọc các tùy chọn mà người dùng có thể chọn.

 **normalizeString:** Một hàm tiện ích để chuẩn hóa chuỗi bằng cách loại bỏ dấu phụ, giúp thực hiện việc so khớp không phân biệt chữ hoa chữ thường và không phân biệt dấu phụ dễ dàng hơn.

A screen shot of a computer program

Description automatically generated

**fetchOptions:** Lấy danh sách các tùy chọn (ví dụ: câu hỏi) từ API và lưu trữ chúng ở trạng thái để lọc và hiển thị.

A screen shot of a computer code

Description automatically generated

**useEffect:** Lấy các tùy chọn khi danh sách tùy chọn hiển thị và lọc các tùy chọn khi bộ lọc đầu vào thay đổi.A screenshot of a computer program

Description automatically generated

 **handleOptionClick:** Đặt tùy chọn đã chọn làm tin nhắn hiện tại.

 **handleInputChange:** Cập nhật đầu vào bộ lọc cho các tùy chọn.

-----------------------------------------------------------------------------------

A screenshot of a computer code

Description automatically generated

**Khai báo các state:**

* newOptions: Là một mảng chứa danh sách các sản phẩm được lấy từ API. Dùng useState để quản lý giá trị này.
* inputFilter2: Là một chuỗi chứa giá trị mà người dùng nhập vào ô tìm kiếm, dùng để lọc danh sách sản phẩm. Dùng useState để quản lý giá trị này.
* optionalListVisible2: Một biến boolean dùng để điều khiển việc hiển thị danh sách các tùy chọn (sản phẩm) khi tìm kiếm.
* filteredOptions2: Mảng chứa danh sách sản phẩm đã được lọc theo từ khóa nhập vào của người dùng.

A screen shot of a computer program

Description automatically generated

**Hàm handleInputChange2:**

* Hàm này được gọi khi người dùng nhập vào ô tìm kiếm. Nó cập nhật giá trị của inputFilter2 dựa trên giá trị nhập vào.

A screen shot of a computer program

Description automatically generated

**Hook useEffect:**

* Hook này theo dõi sự thay đổi của inputFilter2 và newOptions.
* Khi inputFilter2 hoặc newOptions thay đổi, hook này sẽ chạy lại và thực hiện lọc danh sách newOptions dựa trên giá trị của inputFilter2.
* Hàm normalizeString có thể được sử dụng để chuẩn hóa chuỗi (ví dụ, bỏ dấu hoặc viết thường) trước khi so sánh.
* Bộ lọc kiểm tra xem từ khóa nhập vào có xuất hiện trong tên sản phẩm (option.name), giá gốc (option.originalPrice) hoặc giá cuối cùng (option.finalPrice) hay không.

**Hàm handleOptionClick2:**

* Hàm này được gọi khi người dùng chọn một sản phẩm từ danh sách.
* Nó cập nhật giá trị tin nhắn với mã SKU của sản phẩm đã chọn và xóa bỏ giá trị trong ô tìm kiếm (inputFilter2).
* Ngoài ra, có thể thêm chức năng ẩn danh sách tùy chọn (được đề cập trong comment).

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Similar Setup:** Các danh sách này có chức năng tương tự như danh sách đầu tiên nhưng lấy dữ liệu từ các điểm cuối khác nhau và lọc các trường khác nhau.

A screen shot of a computer program

Description automatically generated

**Toggling Lists:** Các hàm xử lý việc chuyển đổi danh sách tùy chọn và trình chọn biểu tượng cảm xúc, đảm bảo chỉ có một biểu tượng hiển thị tại một thời điểm.

A computer screen shot of a computer code

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Hàm handleDeleteClick:**

* Hàm này được gọi khi người dùng nhấn vào nút xóa một tin nhắn.
* **Xác nhận xóa:** Sử dụng window.confirm để hiển thị thông báo hỏi người dùng xem họ có chắc chắn muốn xóa tin nhắn hay không.
* Nếu người dùng xác nhận:
  + Tạo một đối tượng chatMessage chứa hành động "DELETE", cùng với các thông tin như ID của tin nhắn (messageItem.id), người gửi (senderId), và người nhận (recipientId).
  + Gửi đối tượng này tới máy chủ thông qua stompClient.send, gửi tới endpoint /app/chat (WebSocket).
  + Hiển thị thông báo "Message deletion initiated." trong console.
* Nếu người dùng hủy bỏ việc xóa, sẽ hiển thị thông báo "Message deletion canceled." trong console.
* Sau khi xử lý, setEditingMessageId(null) sẽ đặt ID của tin nhắn đang được chỉnh sửa về null (nghĩa là không còn tin nhắn nào đang chỉnh sửa).
* Cuối cùng, gọi fetchAndDisplayUserChat() để làm mới và hiển thị lại danh sách tin nhắn sau khi xóa.

A computer screen shot of a program code

Description automatically generated

**Hàm handleConfirmClick:**

* Hàm này được gọi khi người dùng xác nhận cập nhật tin nhắn đã chỉnh sửa.
* Tạo một đối tượng chatMessage chứa hành động "UPDATE", cùng với ID tin nhắn, thông tin người gửi, người nhận và nội dung đã chỉnh sửa (editContent.trim()).
* Nội dung tin nhắn được cắt bỏ khoảng trắng thừa bằng hàm trim().
* Gửi đối tượng này tới máy chủ thông qua stompClient.send đến endpoint /app/chat.
* Hiển thị thông báo "Message update initiated." trong console.
* Đặt ID của tin nhắn đang chỉnh sửa về null và gọi fetchAndDisplayUserChat() để làm mới danh sách tin nhắn.

**Hàm handleCancelEdit:**

* Hàm này được gọi khi người dùng hủy bỏ việc chỉnh sửa một tin nhắn.
* Đặt ID của tin nhắn đang chỉnh sửa về null, nghĩa là giao diện sẽ không còn ở trạng thái chỉnh sửa nữa.

A computer screen shot of text

Description automatically generated

**Hàm scrollToBottom:**

* Hàm này dùng để tự động cuộn xuống cuối danh sách tin nhắn.
* messagesEndRef.current?.scrollIntoView({}): Sử dụng ref messagesEndRef để cuộn đến vị trí cuối cùng của danh sách tin nhắn, đảm bảo rằng người dùng luôn thấy được tin nhắn mới nhất.

**Hook** useEffect**:**

* useEffect(scrollToBottom, [messages]): Mỗi khi messages thay đổi (có tin nhắn mới), scrollToBottom sẽ được gọi để cuộn xuống cuối danh sách.

**Hàm connect:**

* Kết nối tới WebSocket bằng cách sử dụng SockJS và STOMP.
* Tạo đối tượng socket từ URL của WebSocket (SOCKET\_URL), sau đó tạo stompClient bằng hàm over() của STOMP.
* Gọi stompClient.connect để kết nối với WebSocket, truyền vào token xác thực qua header và gọi hàm onConnected khi kết nối thành công, hoặc onError nếu có lỗi xảy ra.

A screen shot of a computer program

Description automatically generated

**Hàm onConnected:**

* Khi kết nối thành công, hàm này sẽ đăng ký (subscribe) vào hai kênh tin nhắn:
  + /user/${name}/queue/messages: Kênh riêng dành cho tin nhắn của người dùng hiện tại.
  + /user/queue/public: Kênh dành cho các tin nhắn công khai.
* Sau đó, gửi thông tin người dùng tới máy chủ qua endpoint /app/addUser, bao gồm tên, URL ảnh đại diện, trạng thái (ONLINE), và vai trò của người dùng.
* Nếu các biến như optionalListVisible, optionalListVisible2, optionalListVisible3, và showEmojiPicker đều false, nó sẽ gọi fetchOptions2() để lấy danh sách sản phẩm, đồng thời hiển thị danh sách này bằng cách thay đổi trạng thái setOptionalListVisible2.
* Cuối cùng, gọi hàm findAndDisplayConnectedUsers() để hiển thị danh sách người dùng đã kết nối.

**Hàm logout:**

* Được gọi khi người dùng đăng xuất.
* Kiểm tra nếu tên người dùng không phải là RECEIVER.chatbot, thì sẽ gửi một tin nhắn tới server qua endpoint /app/disconnectUser, thông báo rằng người dùng đã chuyển trạng thái sang OFFLINE.
* Xóa sạch thông tin của người nhận tin nhắn (save\_ReiverID = null) và danh sách tin nhắn (setMessages([])).
* Cuối cùng, ngắt kết nối với WebSocket thông qua stompClient.disconnect().

A screen shot of a computer code

Description automatically generated

**Fetch danh sách người dùng:**

* const connectedUsersResponse = await fetch('http://localhost:8080/users', {...}):
  + Gửi yêu cầu GET đến API http://localhost:8080/users để lấy danh sách người dùng đang kết nối.
  + Yêu cầu này bao gồm phần Authorization với Bearer token để xác thực người dùng hiện tại.
  + Sau khi nhận được phản hồi từ server, phương thức then() được dùng để chuyển đổi phản hồi thành JSON (là danh sách người dùng).

**Lọc người dùng hiện tại:**

* connectedUsersResponse.filter((user) => user.name !== name):
  + Lọc bỏ người dùng có tên trùng với người dùng hiện tại (user.name !== name) để không hiển thị chính mình trong danh sách những người nhận tin nhắn.
  + Kết quả là mảng connectedUsers, chỉ bao gồm những người dùng khác đang kết nối.

**Cập nhật danh sách người nhận:**

* setReceiversList(connectedUsers):
  + Cập nhật danh sách những người nhận tin nhắn bằng cách sử dụng setReceiversList, gán giá trị mới là danh sách những người dùng đã lọc (connectedUsers).

A screen shot of a computer program

Description automatically generated**Fetch tin nhắn giữa người dùng và người nhận:**

* const userChatResponse = await fetch(...): Hàm này gửi một yêu cầu GET tới API http://localhost:8080/api/v1/chat-box/messages/${name}/${receiverId} để lấy danh sách tin nhắn giữa người dùng hiện tại (name) và người nhận (receiverId).
  + Phần Authorization sử dụng Bearer token để xác thực người dùng hiện tại.
  + Kết quả của yêu cầu này, sau khi chuyển đổi từ JSON, được lưu trong biến userChatResponse.

**Cập nhật danh sách tin nhắn:**

* setMessages(userChatResponse): Hàm này cập nhật danh sách tin nhắn bằng cách gọi setMessages với dữ liệu từ userChatResponse, tức là danh sách các tin nhắn giữa người dùng và người nhận.

**Kiểm tra người nhận là chatbot:**

* if (receiverId === RECEIVER.chatbot): Kiểm tra xem người nhận có phải là chatbot hay không. Nếu đúng:
  + Gửi yêu cầu thứ hai tới API để lấy các tin nhắn giữa chatbot và người dùng, với URL dạng http://localhost:8080/api/v1/chat-box/messages/${receiverId}/${name} (đảo ngược thứ tự receiverId và name so với yêu cầu đầu tiên).
  + Sau khi nhận phản hồi, cũng cập nhật danh sách tin nhắn với dữ liệu mới từ chatbot bằng cách gọi lại setMessages(userChatResponse).

A screenshot of a computer program

Description automatically generatedA computer screen shot of text

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

Dfasdfasdf

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A computer screen shot of a program code

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

# TRIỂN KHAI VÀ KIỂM THỬ GIAO DIỆN

## Chat Bot

Giao diện

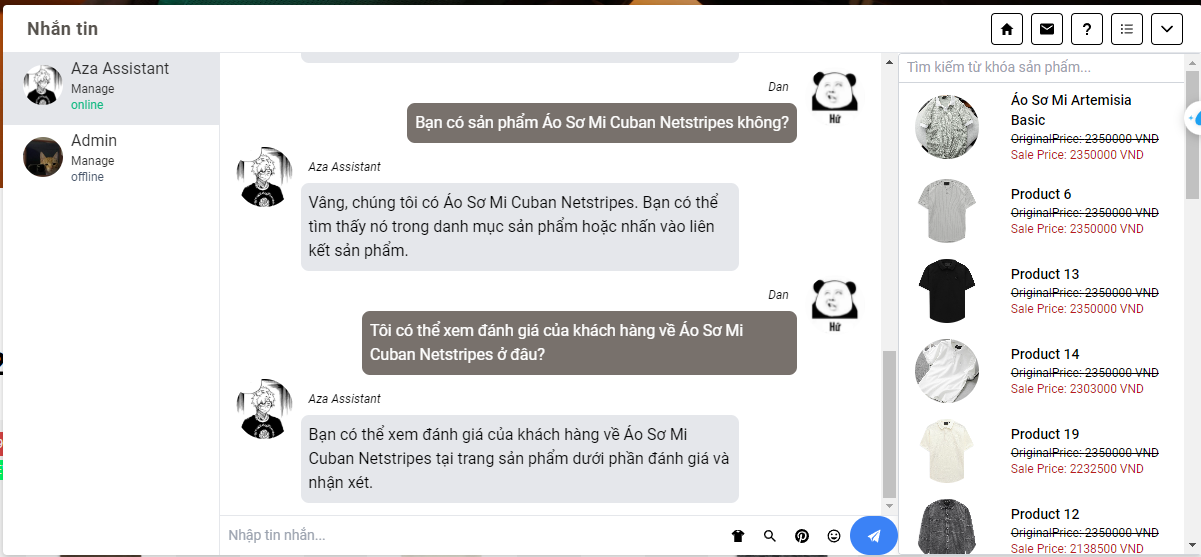
A screenshot of a computer

Description automatically generated

A screenshot of a video chat

Description automatically generated

Đặt câu hỏi thắc mắc



Kiểm tra có khả năng đọc emoji, nhãn dán không ?

A screenshot of a chat

Description automatically generated

Kiểm tra khả năng phản ứng của chatbot

A screenshot of a chat

Description automatically generated

## Chat Message

Giao diện chat User

A screenshot of a computer

Description automatically generated

Hiện thị trạng thái Online hoặc Offline của user

A screenshot of a phone

Description automatically generated



Kiểm tra nhận tín nhắn và thông báo

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Nâng cao

Bao gồm Nhãn dán, Emoji, Sản phẩm, câu hỏi gợi ý cho chatbot, đều được gói gọn ở phần front

A screenshot of a cell phone

Description automatically generatedA screenshot of a product

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

# TÀI LIỆU THAM KHẢO

Tài liệu đọc thêm và Tham khảo

1. [React Document](https://legacy.reactjs.org/docs/introducing-jsx.html) – Giới thiệu JSX
2. [Java Spring boot guides](https://spring.io/guides) – Hướng dẫn CRUD cơ bản (Rest, Restfull….)
3. [Chat Bot Documenter](https://documenter.getpostman.com/view/25622053/2s93sZ5tap) – Chatbot guilde test PostMan
4. [Chat Bot Git Document](https://github.com/chitrakarayat/chatbot-springboot?tab=readme-ov-file) – Link git Tham Khảo
5. [Socket Document Guide](https://socket.io/get-started/chat) – Document Guilde

Link Code Tham khảo

6.0 [E-commerce-Website-Backend](https://github.com/Dan-blcw/spring-boot-E-commerce-Website)

7.0 [E-commerce-Website-Frontend](https://github.com/Dan-blcw/E-commerce-Website)