HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

CƠ SỞ TẠI THÀNH PHỐ HỒ CHÍ MINH

BÁO CÁO ĐỒ ÁN

PHÂN TÍCH THIẾT KẾ VÀ ĐẢM BẢO CHẤT LƯỢNG PHẦN MỀM

Đề tài: "XÂY DỰNG ỨNG DỤNG BLOG ĐÁP ỨNG THỜI GIAN THỰC"

Giảng viên hướng dẫn : Nguyễn Anh Hào

Tên	MSSV
Trần Đăng Khoa	N18DCCN101
Nguyễn Phan Nhựt Trường	N20DCCN082
Lê Trọng Chiến	N20DCCN090

Tp. Hồ Chí Minh - 10/2024

Mục lục

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI 4
1.1. Mục đích nghiên cứu đề tài 4
1.2. Mục tiêu nghiên cứu đề tài 4
1.3. Phương pháp tiến hành 5
CHƯƠNG 2: CƠ SỞ KHOA HỌC 6
2.1 Quy trình nghiệp vụ 6
2.1.1 Quy trình đăng ký và đăng nhập: 6
2.1.2 Quy trình quản lý bài đăng: 6
2.1.3. Quy trình quản lý bạn bè và theo dõi: 6
2.1.4. Quy trình nhắn tin thời gian thực: 7
2.1.5. Quy trình quản lý người dùng (dành cho admin): 7
2.1.6. Quy trình quản lý bài đăng và bình luận: 7
2.1.7. Quy trình tương tác với bài đăng: 8
2.2 Công nghệ sử dụng 8
2.2.1 Spring Boot 8
2.2.2 Spring Security và JWT 8
2.2.3 Spring Data JPA (Hibernate) 9
2.2.4 MySQL 9
2.2.5 ReactJS 9
2.2.6 Socket.IO 9
2.2.6 RabbitMQ 12
CHƯƠNG 3: PHÂN TÍCH HỆ THỐNG 15
3.1 Bối cảnh của hệ thống 15
3.1.1 Định nghĩa vấn đề 15
3.1.2 Hiện trạng trước khi có phần mềm 16
3.1.3 Giải pháp của đề tài 16
3.2 Các tương tác trên PM 16
3.2.1 Từ điển dữ liệu 16
3.2.2 Use case đăng bài 17
3.2.3 Use case bình luận bài viết 18
3.2.3.1 Sơ đồ use case 18
3.2.3.2 Sơ đồ tuần tự 19
3.2.3.3 Sơ đồ lớp 20
3.2.4 Use case kết nối người dùng 21
3.3 Yêu cầu và ràng buộc đối với phần mềm 23
3.3.1 Yêu cầu từ môi trường nghiệp vụ 23
3.3.2 Yêu cầu từ môi trường vận hành 24
3.4 Yêu cầu từ môi trường phát triển 25
CHƯƠNG 4: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG 26
4.1 Lược đồ use case cho giải pháp 26
4.1.1 Use case đăng bài 26
4.1.2 Use case bình luận bài viết 26

4.1.3 Use case kết nối người dùng 27
4.2 Thiết kế phần mềm 27
4.2.1 Form 27
4.2.1.1 Form đăng bài (ID: F01) 27
4.2.1.2 Form bình luận bài viết (ID: F02) 29
4.2.1.3 Form báo cáo bình luận (ID: F03) 29
4.2.1.4 Trang cá nhân người dùng khác (ID: F04) 30
4.2.1.5 Form nhắn tin (ID: F05) 31
4.2.2 API 31
4.2.2.1 API kiểm duyệt bài viết tự động 31
4.2.2.2 API Lưu bài viết vào danh sách chờ duyệt 32
4.2.2.3 API Lưu bài viết vào danh sách bài đăng đã được kiểm duyệt 32
4.2.2.5 API kiểm duyệt từ khóa 33
4.2.2.6 API lưu bình luận 34
4.2.2.7 API Tạo report 34
4.2.2.8 API lấy thông tin cơ bản của một user 35
4.2.2.9 API kiểm tra xem có đang follow user đó không 35
4.2.2.10 API Kiểm tra xem user đó đã kết bạn chưa 36
4.2.2.11 API Theo dõi(follow) một user 37
4.2.2.12 API gửi yêu cầu kết bạn 37
4.2.2.13 API hủy theo dõi user cụ thể 38
4.2.2.14 API hủy kết bạn user cụ thể 38
4.2.2.15 API lấy lịch sử tin nhắn với một user cụ thể 39
4.2.2.16 API gửi một tin nhắn 40
4.2.3 Socket 41
4.2.3.1 Socket gửi thông báo report 41
4.2.3.2 Socket gửi thông báo tin nhắn mới 41
4.3 Thiết kế cơ sở dữ liệu 43
4.3.1 Xác định thực thể 43
4.3.2 Mô hình ERD 43
4.3.3 Mô hình cơ sở dữ liệu quan hệ 43
4.3.4 Mô hình dữ liệu 47
4.3.5 Phân tích và hoàn thiện mô hình quan hệ 49
TÀI LIỆU THAM KHẢO 50

CHƯƠNG 1: GIỚI THIỆU ĐỀ TÀI

1.1. Mục đích nghiên cứu đề tài

Mục đích của đề tài "Xây dựng ứng dụng blog đáp ứng thời gian thực" là nghiên cứu và triển khai một ứng dụng web hiện đại, cho phép người dùng tương tác với hệ thống một cách mượt mà và nhanh chóng, đặc biệt trong các tình huống yêu cầu phản hồi ngay lập tức như nhắn tin và cập nhật bài viết. Trong bối cảnh công nghệ phát triển nhanh chóng, người dùng ngày càng kỳ vọng vào những trải nghiệm tức thời và liền mạch trên các nền tảng trực tuyến. Vì vậy, đề tài này nhằm khám phá và tích hợp các công nghệ tiên tiến như Spring Boot, Spring Security, JWT, Spring Data JPA (Hibernate), MySQL, ReactJS, Socket.IO, RabbitMQ cùng nhiều công nghệ khác để xây dựng một hệ thống blog không chỉ đảm bảo hiệu suất cao mà còn đáp ứng được nhu cầu về tính năng thời gian thực.

Ứng dụng blog sẽ phục vụ như một nền tảng nơi người dùng có thể tạo, chỉnh sửa, xóa và quản lý các bài viết của mình, đồng thời giao tiếp với các người dùng khác thông qua tính năng nhắn tin thời gian thực. Việc tích hợp các công nghệ như Socket.IO sẽ giúp các thông điệp được truyền tải ngay lập tức mà không cần tải lại trang, trong khi RabbitMQ sẽ đảm bảo rằng dữ liệu được xử lý một cách chính xác và hiệu quả ngay cả khi có nhiều người dùng tương tác cùng lúc trên cùng một đối tượng dữ liệu. Ngoài ra, việc sử dụng Spring Security kết hợp với JWT sẽ đảm bảo tính bảo mật cho hệ thống, chỉ cho phép những người dùng có quyền truy cập vào các tài nguyên nhạy cảm.

1.2. Mục tiêu nghiên cứu đề tài

Mục tiêu chính của đề tài bao gồm:

- Phát triển một nền tảng blog hoàn chỉnh: Ứng dụng blog sẽ được phát triển với đầy đủ các chức năng như đăng bài viết, chỉnh sửa, xóa, quản lý danh mục, và phân quyền người dùng. Hệ thống sẽ hỗ trợ nhiều loại nội dung, bao gồm văn bản, hình ảnh và video, tạo điều kiện cho người dùng tạo ra các bài viết phong phú và đa dạng.

- Tích hợp tính năng thời gian thực: Một trong những điểm nhấn của ứng dụng là tính năng nhắn tin thời gian thực, cho phép người dùng tương tác với nhau mà không cần tải lại trang. Socket.IO sẽ đóng vai trò là công cụ chính trong việc truyền tải dữ liệu thời gian thực giữa các người dùng, giúp cải thiện trải nghiệm người dùng và nâng cao hiệu quả giao tiếp.

- Bảo mật hệ thống: Đảm bảo rằng hệ thống có thể bảo vệ dữ liệu cá nhân của người dùng khỏi các nguy cơ bảo mật thông qua việc sử dụng Spring Security và JWT. Hệ thống sẽ yêu cầu người dùng đăng nhập và xác thực thông qua các token bảo mật, chỉ cho phép truy cập vào các tài nguyên nhất định khi người dùng đã được xác thực.

- Quản lý dữ liệu phức tạp: RabbitMQ sẽ được tích hợp để xử lý các tác vụ liên quan đến quản lý dữ liệu phức tạp, chẳng hạn như việc cập nhật đồng thời trên cùng một dữ liệu bởi nhiều người dùng. Điều này sẽ đảm bảo rằng dữ liệu trong cơ sở dữ liệu MySQL luôn được cập nhật chính xác và nhất quán.

1.3. Phương pháp tiến hành

Để đạt được các mục tiêu đề ra, đề tài sẽ sử dụng một loạt các phương pháp tiến hành như sau:

- Phân tích yêu cầu: Tiến hành khảo sát và thu thập yêu cầu từ người dùng cuối, sau đó phân tích và xác định các yêu cầu chức năng và phi chức năng của hệ thống. Điều này sẽ giúp đảm bảo rằng hệ thống đáp ứng đúng và đủ các nhu cầu thực tế của người dùng.

- Thiết kế kiến trúc hệ thống: Xây dựng mô hình kiến trúc tổng thể cho hệ thống, xác định các thành phần chính như backend, frontend, cơ sở dữ liệu và các dịch vụ hỗ trợ. Mỗi thành phần sẽ được thiết kế chi tiết với các công nghệ phù hợp, đảm bảo tính linh hoạt và khả năng mở rộng trong tương lai.

- Phát triển phần mềm: Sử dụng phương pháp phát triển phần mềm Agile để phát triển hệ thống theo từng giai đoạn nhỏ, mỗi giai đoạn sẽ bao gồm lập trình, kiểm thử và điều chỉnh dựa trên phản hồi của người dùng. Việc này giúp tối ưu hóa quy trình phát triển, đồng thời giảm thiểu rủi ro và chi phí phát sinh.

- Kiểm thử và triển khai: Sau khi hoàn thành việc phát triển, hệ thống sẽ được kiểm thử toàn diện bao gồm các bài kiểm tra chức năng, phi chức năng, kiểm tra hiệu suất và bảo mật. Sau khi vượt qua các bài kiểm tra, hệ thống sẽ được triển khai trên môi trường thực tế.

- Bảo trì và cập nhật: Sau khi triển khai, hệ thống sẽ được theo dõi và bảo trì định kỳ để đảm bảo hoạt động ổn định. Ngoài ra, các bản cập nhật và nâng cấp sẽ được thực hiện khi cần thiết để cải thiện tính năng và đáp ứng các yêu cầu mới từ người dùng.

Bằng cách áp dụng những phương pháp này, đề tài "Xây dựng ứng dụng blog đáp ứng thời gian thực" không chỉ đạt được các mục tiêu kỹ thuật mà còn góp phần vào việc nâng cao kỹ năng phát triển phần mềm của người thực hiện, đặc biệt là trong việc áp dụng các công nghệ hiện đại để giải quyết các bài toán phức tạp trong thực tế.

CHƯƠNG 2: CƠ SỞ KHOA HỌC

2.1 Quy trình nghiệp vụ

Về quy trình nghiệp vụ trong ứng dụng blog, có thể liệt kê như sau:

2.1.1 Quy trình đăng ký và đăng nhập:

- **Đăng ký**:
  - Người dùng truy cập trang đăng ký.
  - Điền các thông tin cơ bản như tên, email, mật khẩu.
  - Hệ thống xác thực thông tin và tạo tài khoản mới.
- **Đăng nhập**:
  - Người dùng nhập email và mật khẩu để đăng nhập.
  - Hệ thống xác thực thông tin đăng nhập và cấp quyền truy cập dựa trên vai trò (user hoặc admin).

2.1.2 Quy trình quản lý bài đăng:

Tạo bài đăng:

- Người dùng hoặc admin truy cập trang tạo bài đăng mới.
- Nhập tiêu đề, nội dung và chọn các tag liên quan.
- Hệ thống lưu trữ bài đăng và hiển thị trên giao diện người dùng.

Chỉnh sửa bài đăng:

- Người dùng hoặc admin chỉnh sửa nội dung bài đăng của mình thông qua giao diện chỉnh sửa.
- Hệ thống cập nhật bài đăng và lưu các thay đổi.

Xóa bài đăng:

- Người dùng hoặc admin có thể xóa bài đăng của mình.
- Bài đăng được xóa khỏi hệ thống và không hiển thị nữa.

2.1.3. Quy trình quản lý bạn bè và theo dõi:

- **Gửi yêu cầu kết bạn**:
    - Người dùng gửi yêu cầu kết bạn tới một người dùng khác.
    - Người nhận có thể chấp nhận hoặc từ chối yêu cầu.
- **Theo dõi một người dùng**:
    - Người dùng có thể theo dõi các bài đăng của người khác.
    - Hệ thống sẽ hiển thị bài viết từ các tài khoản đang theo dõi trên trang cá nhân của người dùng.

2.1.4. Quy trình nhắn tin thời gian thực:

- **Gửi tin nhắn**:
    - Người dùng chọn một người để gửi tin nhắn trực tiếp.
    - Tin nhắn được truyền tải ngay lập tức thông qua hệ thống thời gian thực.
- **Nhận tin nhắn**:
    - Người nhận nhận được tin nhắn ngay lập tức mà không cần tải lại trang.

2.1.5. Quy trình quản lý người dùng (dành cho admin):

- **Quản lý danh sách người dùng**:
    - Admin có thể xem toàn bộ danh sách người dùng, phân quyền và điều chỉnh các thông tin tài khoản nếu cần.
- **Cấp quyền admin**:
    - Admin có thể cấp quyền quản trị cho người dùng khác hoặc hạ quyền khi cần thiết.

2.1.6. Quy trình quản lý bài đăng và bình luận:

- **Quản lý bình luận**:
    - Người dùng có thể thêm bình luận hoặc trả lời bình luận trên các bài đăng.
    - Admin có thể xóa bình luận nếu cần thiết.
- **Quản lý tags**:

- Admin có thể tạo mới, chỉnh sửa hoặc xóa các tag không còn sử dụng.

2.1.7. Quy trình tương tác với bài đăng:

- **Like/Dislike**:
  - Người dùng có thể thích (like) hoặc không thích (dislike) bài đăng của người khác.
- **Xem bài đăng theo tag hoặc người theo dõi**:
  - Người dùng có thể lọc bài viết theo tag hoặc xem bài đăng từ những người họ theo dõi.

2.2 Công nghệ sử dụng

2.2.1 Spring Boot

Spring Boot là một framework mã nguồn mở dựa trên Spring Framework, được thiết kế để đơn giản hóa việc tạo và triển khai các ứng dụng Java. Với Spring Boot, nhà phát triển có thể tập trung vào viết mã thay vì phải quản lý cấu hình phức tạp. Spring Boot cung cấp cơ chế tự động cấu hình, giúp thiết lập các thành phần ứng dụng một cách nhanh chóng và dễ dàng. Nó cũng hỗ trợ tích hợp với các công nghệ khác như Spring Security, Spring Data và Spring Cloud, tạo điều kiện thuận lợi cho việc xây dựng các ứng dụng microservices. Bằng cách sử dụng Spring Boot, bạn có thể phát triển và triển khai ứng dụng một cách nhanh chóng, giảm thiểu thời gian phát triển và tối ưu hóa hiệu suất.[1]

2.2.2 Spring Security và JWT

Spring Security là một framework mạnh mẽ dùng để bảo mật các ứng dụng web Java, cung cấp các tính năng như xác thực, phân quyền người dùng và bảo vệ chống lại các cuộc tấn công phổ biến như CSRF và XSS. JWT (JSON Web Token) là một tiêu chuẩn mở để truyền tải thông tin giữa các bên một cách an toàn dưới dạng token. Trong ứng dụng blog này, Spring Security được sử dụng để bảo vệ các API và JWT được sử dụng để quản lý phiên đăng nhập người dùng. Khi người dùng đăng nhập thành công, hệ thống sẽ tạo ra một JWT chứa thông tin xác thực của người dùng, giúp giảm tải cho máy chủ bằng cách loại bỏ nhu cầu kiểm tra trạng thái phiên liên tục.[2]

2.2.3 Spring Data JPA (Hibernate)

Spring Data JPA là một phần của Spring Data, cung cấp cách tiếp cận dễ dàng và mạnh mẽ để làm việc với các cơ sở dữ liệu quan hệ. Nó tích hợp chặt chẽ với Hibernate, một framework ORM (Object-Relational Mapping) phổ biến, giúp chuyển đổi giữa các đối tượng Java và các bảng cơ sở dữ liệu một cách tự động. Spring Data JPA giảm thiểu lượng mã nguồn phải viết để thao tác với cơ sở dữ liệu bằng cách cung cấp các repository có sẵn cho các thao tác CRUD (Create, Read, Update, Delete). Điều này giúp nhà phát triển tập trung vào logic nghiệp vụ thay vì phải viết các câu lệnh SQL phức tạp.[3]

2.2.4 MySQL

MySQL là một hệ quản trị cơ sở dữ liệu quan hệ mạnh mẽ và phổ biến, được sử dụng rộng rãi trong các ứng dụng web. Với MySQL, bạn có thể quản lý và truy vấn dữ liệu một cách hiệu quả, hỗ trợ đa người dùng và đa nhiệm. MySQL tích hợp tốt với nhiều ngôn ngữ lập trình và framework, bao gồm cả Spring Boot, giúp bạn dễ dàng quản lý cơ sở dữ liệu của ứng dụng blog. MySQL cung cấp hiệu suất cao, đáng tin cậy và khả năng mở rộng, là lựa chọn lý tưởng cho các ứng dụng yêu cầu cơ sở dữ liệu quan hệ lớn.

2.2.5 ReactJS

ReactJS là một thư viện JavaScript mã nguồn mở được phát triển bởi Facebook, dùng để xây dựng giao diện người dùng (UI). ReactJS sử dụng kiến trúc component-based, cho phép xây dựng các thành phần UI độc lập, tái sử dụng. Một trong những tính năng nổi bật của ReactJS là Virtual DOM, giúp cải thiện hiệu suất ứng dụng bằng cách cập nhật DOM một cách hiệu quả. JSX, một cú pháp mở rộng của JavaScript, giúp viết mã React dễ hiểu và dễ duy trì. Với ReactJS, bạn có thể tạo ra các ứng dụng web phức tạp với hiệu suất cao và trải nghiệm người dùng mượt mà.

2.2.6 Socket.IO

Socket.IO là một thư viện JavaScript phổ biến được sử dụng để xây dựng các ứng dụng thời gian thực. Trong ngữ cảnh của tính năng nhắn tin thời gian thực trong ứng dụng blog, Socket.IO đảm bảo rằng các tin nhắn được gửi và nhận ngay lập tức giữa các người dùng, tạo ra trải nghiệm tương tác liền mạch. Điểm mạnh của Socket.IO là khả năng thiết lập kết nối hai chiều (bi-directional) giữa máy khách và máy chủ, cho phép dữ liệu được truyền tải một cách nhanh chóng và hiệu quả. Socket.IO hoạt động trên nền tảng WebSocket

nhưng bổ sung thêm nhiều tính năng quan trọng như fallback (dự phòng) qua các giao thức khác (như HTTP long polling) khi WebSocket không khả dụng, đảm bảo rằng kết nối luôn được duy trì một cách ổn định bất kể môi trường mạng.

Trong ứng dụng blog, khi một người dùng gửi tin nhắn, Socket.IO đảm bảo rằng tin nhắn này được truyền tải ngay lập tức đến tất cả người nhận. Điều này giúp cải thiện đáng kể trải nghiệm người dùng, đặc biệt trong các trường hợp cần tính tức thì như nhắn tin trực tiếp giữa các người dùng. Thêm vào đó, Socket.IO cung cấp các cơ chế quản lý phiên, giúp xử lý các kết nối đồng thời và đảm bảo rằng thông tin liên lạc giữa các người dùng được duy trì ổn định.[4]

Chức năng chính

- **Thiết lập server WebSocket**:

```
    const io = new Server(server, {
  cors: {
    origin: "*",
    methods: ["GET", "POST"]
  }
});
```

  - `origin: "*"`: Cho phép tất cả các nguồn gốc (domain) kết nối đến server. Trong môi trường thực tế, bạn nên giới hạn `origin` để tăng cường bảo mật.
  - `methods: ["GET", "POST"]`: Chỉ định các phương thức HTTP được phép.

- **Lắng nghe kết nối từ client**:

```
    io.on('connection', (socket) => {
  console.log('a user connected: ' + socket.id);
});
```

  - Mỗi khi một client kết nối, server sẽ gán cho client một ID duy nhất (`socket.id`). Điều này hữu ích để theo dõi từng client riêng lẻ trong hệ thống.

- **Xử lý sự kiện từ client**:

```
    socket.on("messageClient", (data) => {
  console.log('Received data from client:', data);
  io.emit("messageServer", { message: 'Hello from server!' });
});
```

- Khi client gửi sự kiện `messageClient`, server nhận dữ liệu (`data`), thực hiện xử lý, và phát lại tin nhắn `messageServer` đến tất cả các client.

- **Ngắt kết nối**:

```
  socket.on('disconnect', () => {
  console.log('user disconnected: ' + socket.id);
});
```

- Khi client ngắt kết nối, server ghi nhận điều này để quản lý trạng thái.

Sử dụng Socket.IO trong trường hợp thông báo tin nhắn real-time:

Tích hợp Socket.IO:

- Kết nối với server:

```
  socketRef.current = io("http://localhost:9092", {
  transports: ["websocket", "polling"]
});
```

  - Kết nối đến server Socket.IO tại địa chỉ http://localhost:9092.
- Sự kiện socket:
  - `connect`, `disconnect`, `connect_error`: Theo dõi trạng thái kết nối.
  - `messageServer`: Nhận dữ liệu real-time từ server khi có tin nhắn mới.

Gọi API REST:

- `fetchMessages`:
  - Gửi yêu cầu GET đến API để lấy lịch sử tin nhắn, hỗ trợ phân trang.
  - Tùy chọn `append` để thêm tin nhắn mới vào cuối hoặc làm mới toàn bộ danh sách.
- `sendMessage`:
  - Gửi tin nhắn đến server thông qua API POST.
  - Sau khi gửi, phát sự kiện `messageClient` qua Socket.IO để thông báo cho server rằng có tin nhắn mới.

Quản lý giao diện người dùng:

- Cuộn để tải thêm tin nhắn:
  - Xử lý cuộn trong `chatHistoryRef` để phát hiện khi người dùng cuộn đến cuối và tải thêm tin nhắn.
- Hiển thị danh sách tin nhắn:
  - Mỗi tin nhắn được hiển thị với định dạng `sent` hoặc `received` dựa trên người gửi.

Tích hợp với server Socket.IO

Khi ghép nối với đoạn code Socket.IO server đã cung cấp, các sự kiện tương ứng sẽ hoạt động như sau:

Tin nhắn mới:

Khi người dùng gửi tin nhắn, sự kiện `messageClient` sẽ được phát lên server.

Server nhận sự kiện này, xử lý và phát lại tin nhắn cho tất cả người dùng hoặc người nhận cụ thể thông qua sự kiện `messageServer`.

Tải lại tin nhắn:

Mỗi khi nhận được sự kiện `messageServer`, client sẽ gọi lại API `fetchMessages` để làm mới danh sách tin nhắn.

Cách phối hợp giữa server và client

Kết nối:

Server Socket.IO sẽ quản lý tất cả các kết nối từ client.

Client `MessagePage` sẽ duy trì kết nối thông qua `socketRef`.

Gửi và nhận dữ liệu:

-Client phát sự kiện:

Khi gửi tin nhắn, client phát sự kiện `messageClient` với thông tin cần thiết.

-Server phát sự kiện:

Server nhận sự kiện `messageClient`, xử lý, và phát lại sự kiện `messageServer` để cập nhật tất cả client liên quan.

Cập nhật giao diện:

Client nhận sự kiện `messageServer` và tự động gọi lại API để làm mới giao diện.

2.2.6 RabbitMQ

RabbitMQ là một hệ thống hàng đợi tin nhắn (message broker) mã nguồn mở, được sử dụng để quản lý và điều phối luồng dữ liệu giữa các dịch vụ trong một ứng dụng phân tán. Trong bối cảnh ứng dụng blog có nhiều người dùng cùng tương tác và thay đổi các thuộc tính trong cơ sở dữ liệu đồng thời, RabbitMQ đóng vai trò quan trọng trong việc tránh xung đột dữ liệu và đảm bảo tính nhất quán.

RabbitMQ hoạt động bằng cách nhận các thông điệp từ các producer (nhà sản xuất), sau đó phân phối chúng đến các consumer (người tiêu thụ) thông qua các hàng đợi tin nhắn. Khi nhiều người dùng cùng thực hiện các thay đổi lên cùng một thuộc tính trong bảng cơ sở dữ liệu, RabbitMQ có thể được sử dụng để điều phối các yêu cầu này, xếp chúng vào hàng đợi và xử lý từng yêu cầu một cách tuần tự. Điều này giúp ngăn chặn các tình huống xung đột dữ liệu, nơi mà một thuộc tính có thể bị ghi đè bởi các giá trị khác nhau từ nhiều người dùng.

Ngoài ra, RabbitMQ còn cung cấp các cơ chế để xác nhận thông điệp đã được xử lý thành công, cũng như khả năng xử lý các thông điệp bị lỗi hoặc không thể xử lý, giúp đảm bảo tính tin cậy và chính xác của dữ liệu trong hệ thống. Điều này đặc biệt quan trọng trong các ứng dụng yêu cầu độ chính xác cao.[5]

Các khái niệm chính trong RabbitMQ

1. **Queue (Hàng đợi)**: Lưu trữ thông điệp chờ được xử lý.
2. **Exchange**: Điều phối thông điệp đến các hàng đợi dựa trên routing key.
3. **Binding**: Liên kết giữa Exchange và Queue.
4. **Message Converter**: Chuyển đổi định dạng thông điệp (JSON, XML, text).

Cách cấu hình trong code

- **Định nghĩa hàng đợi**:

```
    @Bean
public Queue postViewQueue() {
    return new Queue(AppConstants.POST_VIEW_QUEUE, true);
}
```

  - Hàng đợi `postViewQueue` được khai báo là bền vững (`durable`), nghĩa là nó sẽ không bị mất khi RabbitMQ khởi động lại.
  - Tương tự, các hàng đợi khác như `postLikeQueue`, `commentOnPostQueue`... được khai báo để xử lý các sự kiện tương ứng trong hệ thống.

- **Message Converter**:

```
    @Bean
public Jackson2JsonMessageConverter
producerJackson2MessageConverter() {
    return new Jackson2JsonMessageConverter();
}
```

  - `Jackson2JsonMessageConverter` đảm bảo các thông điệp được gửi đi hoặc nhận về được chuyển đổi sang định dạng JSON.

- **RabbitTemplate**:

```
    @Bean
public RabbitTemplate rabbitTemplate(ConnectionFactory
connectionFactory) {
    RabbitTemplate template = new
RabbitTemplate(connectionFactory);

template.setMessageConverter(producerJackson2MessageConverter());
```

```
    return template;
}
```

- `RabbitTemplate` được sử dụng để gửi thông điệp đến các hàng đợi trong RabbitMQ. Nó tích hợp sẵn `MessageConverter` để làm việc với định dạng JSON.

Phối hợp giữa Socket.IO và RabbitMQ

Luồng xử lý

1. **Socket.IO nhận dữ liệu từ client**:
   a. Client gửi một sự kiện (`messageClient`) đến server qua Socket.IO.
   b. Dữ liệu có thể bao gồm các hành động như "like bài viết", "bình luận", hoặc "xem bài viết".
2. **Đưa dữ liệu vào hàng đợi RabbitMQ**:
   a. Sau khi nhận dữ liệu, server có thể đưa sự kiện này vào hàng đợi RabbitMQ tương ứng:

```
    const message = { action: 'likePost', postId: 123, userId: 456
};
rabbitTemplate.convertAndSend(AppConstants.POST_LIKE_QUEUE,
message);
```

      i. Ví dụ: Khi người dùng "like" bài viết, sự kiện được đưa vào hàng đợi `POST_LIKE_QUEUE`.

3. **Xử lý thông điệp trong hàng đợi**:
   a. Một dịch vụ khác (worker) sẽ lắng nghe các hàng đợi để xử lý thông điệp:

```
    @RabbitListener(queues = AppConstants.POST_LIKE_QUEUE)
public void handlePostLike(String message) {
    System.out.println("Processing like for post: " + message);
    // Thực hiện logic như cập nhật cơ sở dữ liệu
}
```

4. **Phản hồi lại client (nếu cần)**:
   a. Sau khi xử lý xong, dịch vụ có thể phát tín hiệu lại cho tất cả các client qua Socket.IO:

```
io.emit("updatePostLikes", { postId: 123, likes: 10 });
```

Lợi ích của việc kết hợp

- **Socket.IO**:
    - Đáp ứng tức thì (real-time) với các sự kiện từ client.
    - Thích hợp cho giao tiếp hai chiều, ví dụ: thông báo, trò chuyện.
- **RabbitMQ**:
    - Xử lý thông điệp bất đồng bộ, giúp giảm tải cho server thời gian thực.
    - Tăng khả năng mở rộng khi tích hợp nhiều dịch vụ xử lý sự kiện.

Ví dụ thực tế

1. Người dùng nhấn "like" một bài viết:
    a. **Socket.IO** nhận sự kiện từ client và gửi đến RabbitMQ.
    b. **RabbitMQ** đưa sự kiện vào hàng đợi `POST_LIKE_QUEUE`.
    c. Worker xử lý sự kiện, cập nhật dữ liệu trong cơ sở dữ liệu.
    d. Worker phát tín hiệu qua Socket.IO để cập nhật số lượt "like" trên giao diện.
2. Người dùng nhận thông báo mới:
    a. **RabbitMQ** phát thông báo đến hàng đợi `NOTIFICATION_QUEUE`.
    b. Một worker gửi thông báo qua Socket.IO đến các client liên quan.

CHƯƠNG 3: PHÂN TÍCH HỆ THỐNG

3.1 Bối cảnh của hệ thống

3.1.1 Định nghĩa vấn đề

Trong bối cảnh xã hội hiện đại, các nền tảng chia sẻ thông tin không chỉ cần tính năng đăng tải mà còn cần tính tương tác thời gian thực. Các blog truyền thống chỉ cung cấp chức năng đăng bài và nhận bình luận một cách chậm trễ, thiếu đi tính năng kết nối ngay tức thì giữa người dùng, dẫn đến một số hạn chế:

- Tương tác chậm: Người dùng không thể phản hồi, kết nối hoặc nhận thông tin tức thời, gây khó khăn cho những cuộc trò chuyện qua bình luận.

- Thiếu kết nối: Các blog truyền thống không có tính năng kết bạn và nhắn tin tức thì, làm giảm trải nghiệm người dùng.

Mục tiêu của ứng dụng blog thời gian thực:

- Tạo một môi trường tương tác cho người dùng, cho phép đăng bài, bình luận, kết bạn và nhắn tin ngay lập tức.
- Nâng cao trải nghiệm người dùng thông qua việc cập nhật bài viết và tương tác một cách tức thời, giúp người dùng dễ dàng kết nối và giao tiếp.

3.1.2 Hiện trạng trước khi có phần mềm

Trước khi phần mềm blog thời gian thực được phát triển, các blog truyền thống gặp nhiều khó khăn trong việc đáp ứng nhu cầu tương tác nhanh và khả năng kết nối:

- Đăng bài và bình luận chậm: Bài viết và bình luận chỉ được tải lại khi người dùng làm mới trang, dẫn đến chậm trễ trong việc nhận phản hồi và mất tính tương tác.
- Thiếu tính năng nhắn tin trực tiếp và kết bạn: Người dùng không thể kết bạn hoặc nhắn tin trực tiếp cho nhau, dẫn đến hạn chế trong việc tạo dựng cộng đồng và duy trì các mối quan hệ.

3.1.3 Giải pháp của đề tài

Ứng dụng blog thời gian thực sẽ được thiết kế để khắc phục các nhược điểm trên, mang lại trải nghiệm tương tác liền mạch cho người dùng. Phần mềm sẽ đảm bảo các chức năng chính như sau:

- Đăng bài thời gian thực: Người dùng có thể đăng bài và thấy ngay lập tức nội dung trên trang của mình cũng như trong nguồn cấp của những người theo dõi.
- Bình luận tức thì: Khi có bình luận mới, các bình luận sẽ xuất hiện ngay lập tức mà không cần tải lại trang, đảm bảo tương tác trực tiếp giữa tác giả và người đọc.
- Kết bạn và nhắn tin trực tiếp: Người dùng có thể kết bạn và gửi tin nhắn trực tiếp cho nhau mà không cần dùng thêm các nền tảng khác. Các tin nhắn sẽ hiển thị tức thời, giúp người dùng giao tiếp thuận tiện và dễ dàng duy trì các mối quan hệ.

3.2 Các tương tác trên PM

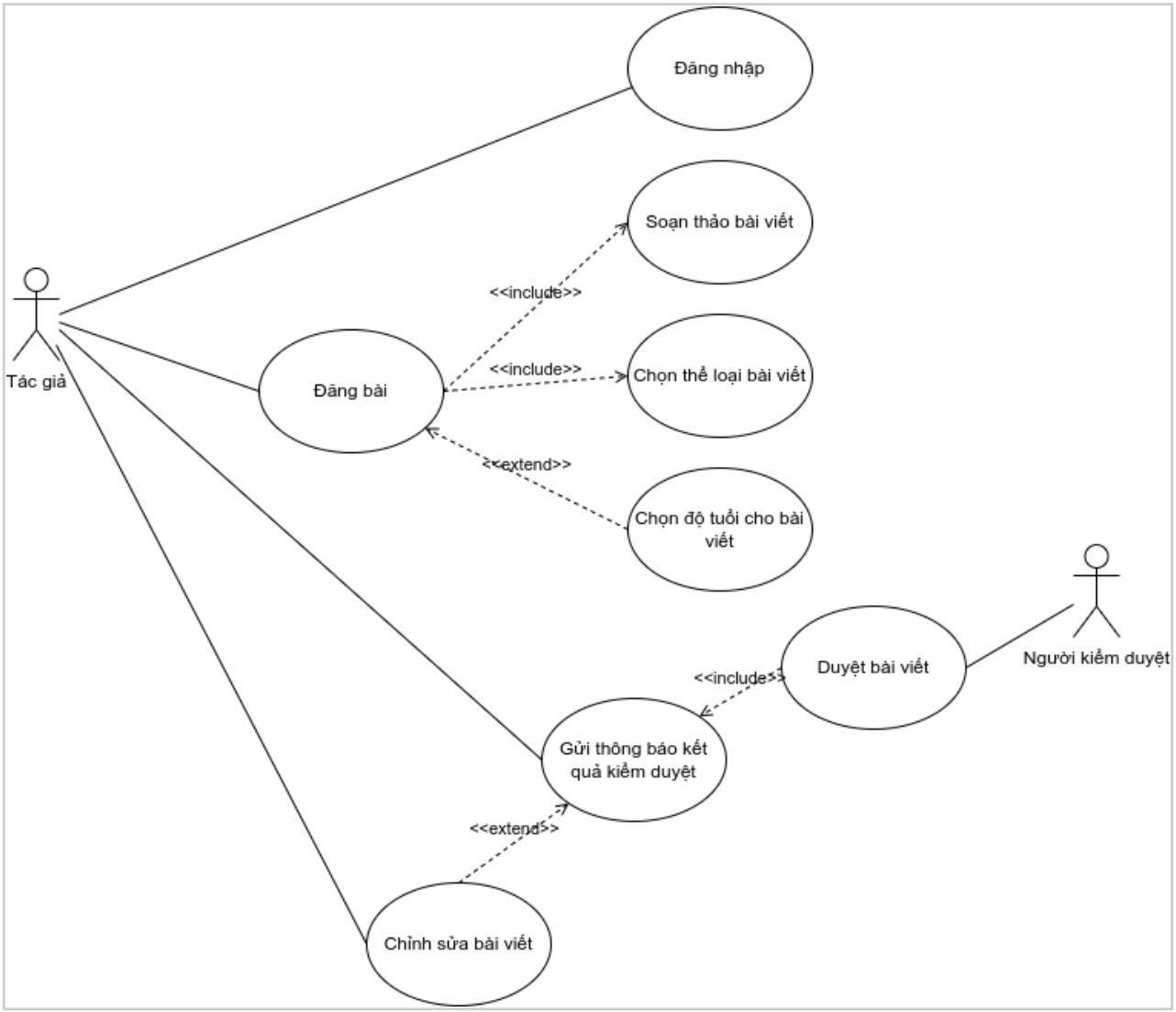
3.2.1 Từ điển dữ liệu

Tên	Chi tiết	Nguồn
Comment	Id, content, idAuthor, idPost idParent, created_at	User, Process
Vote	idAuthor, idPost	User, Process
Report	Id, idComment, idAuthor, reason	User, Process, Admin

DLCN	Name, username, email, joined, role	User, Process

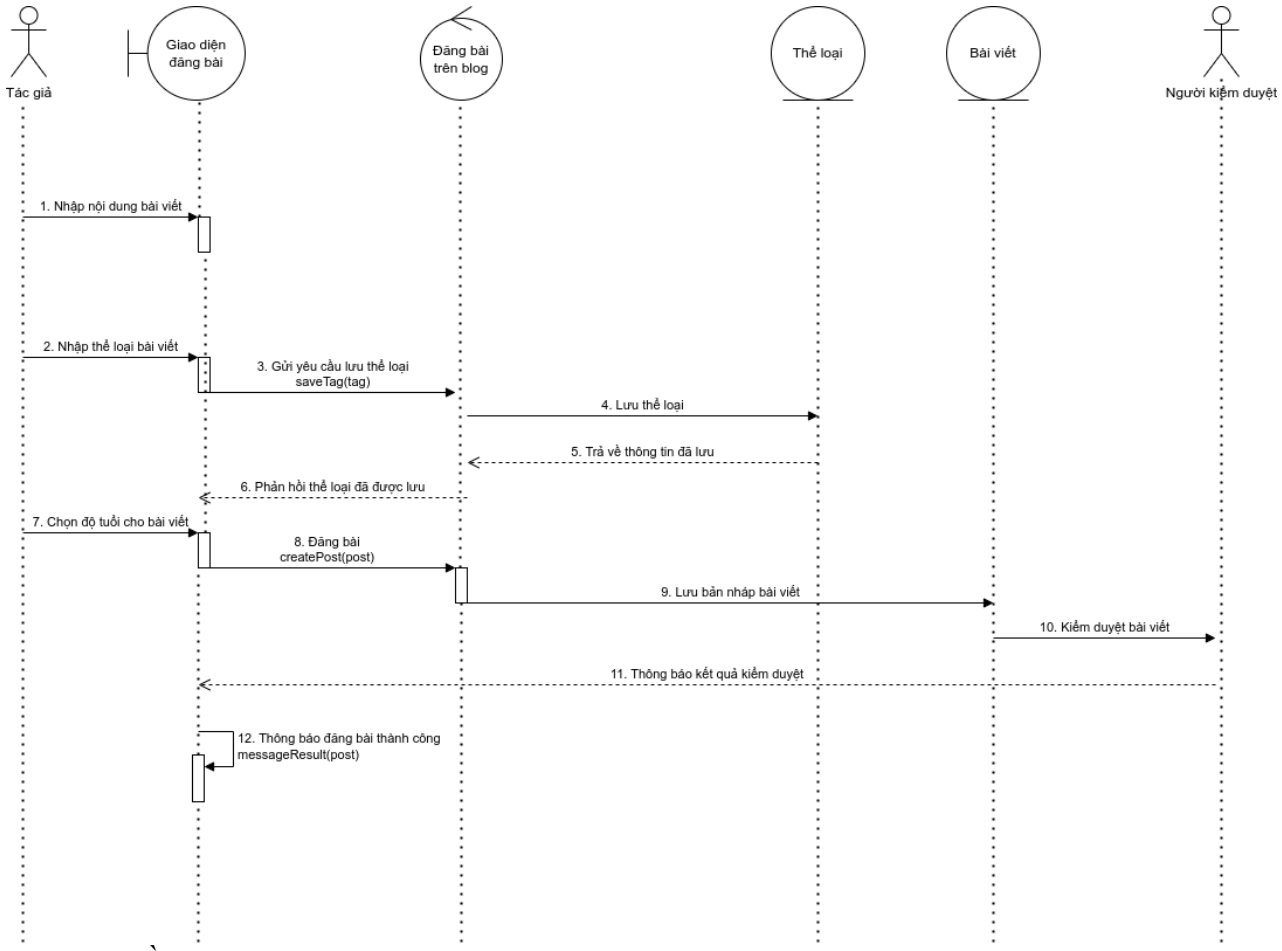
3.2.2 Use case đăng bài

3.2.2.1 Sơ đồ use case
Use case: Đăng bài
Id: U01

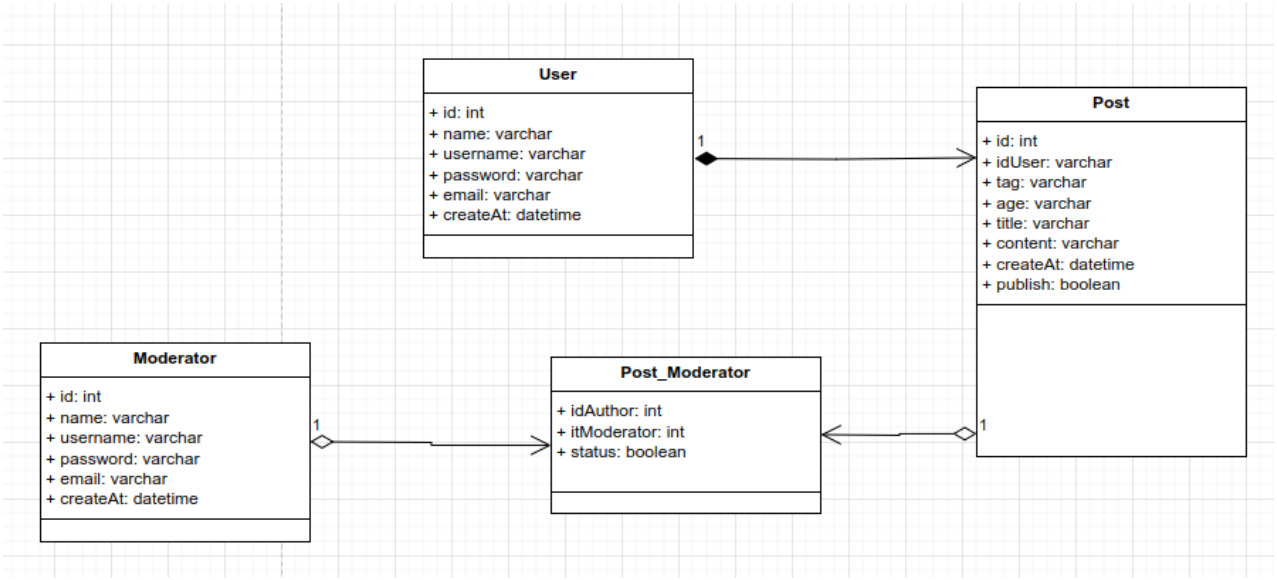


Ý nghĩa: tác giả có thể tạo bài đăng sau khi đăng nhập với nội dung bài đăng hợp lệ.

3.2.2.2 Sơ đồ tuần tự



Sơ đồ lớp:

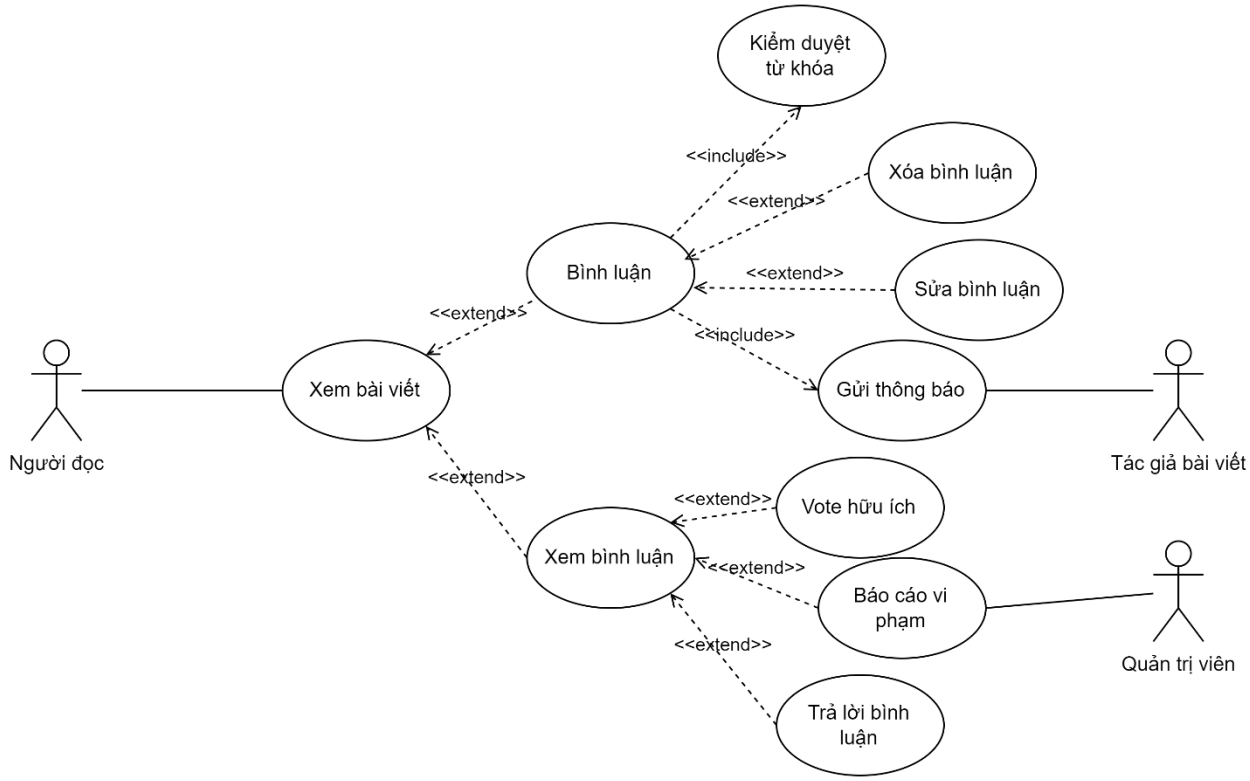


3.2.3 Use case bình luận bài viết

3.2.3.1 Sơ đồ use case

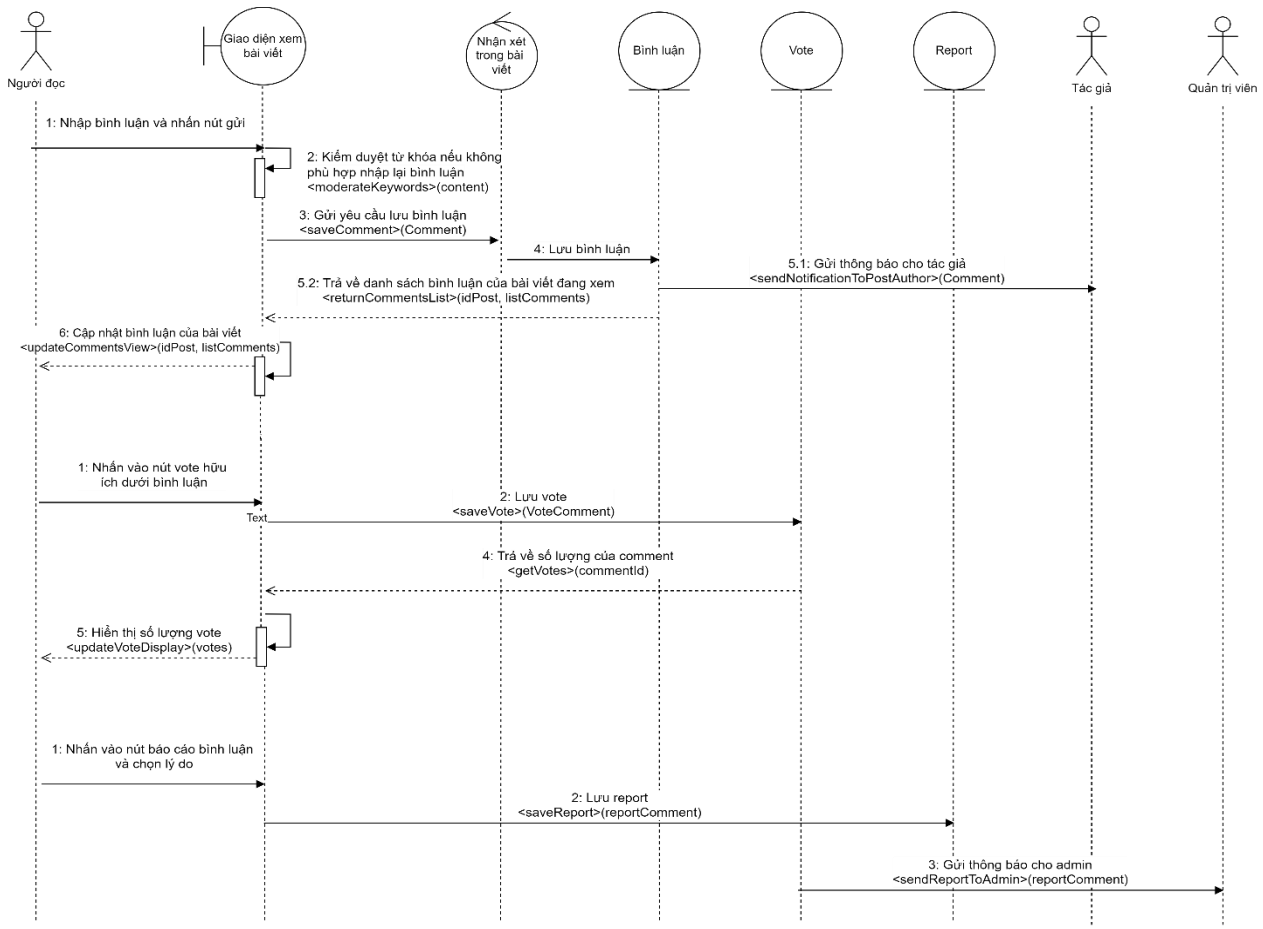
Use case: bình luận bài viết

Id: U02

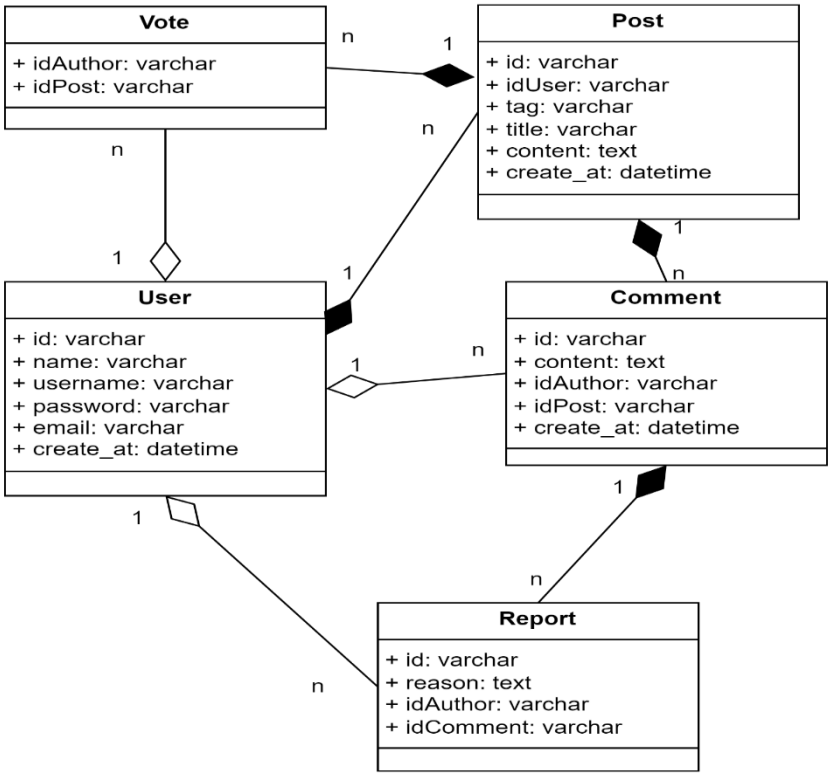


Ý nghĩa: Người dùng có thể tương tác với nội dung bài viết và bình luận theo nhiều cách khác nhau

3.2.3.2 Sơ đồ tuần tự



3.2.3.3 Sơ đồ lớp

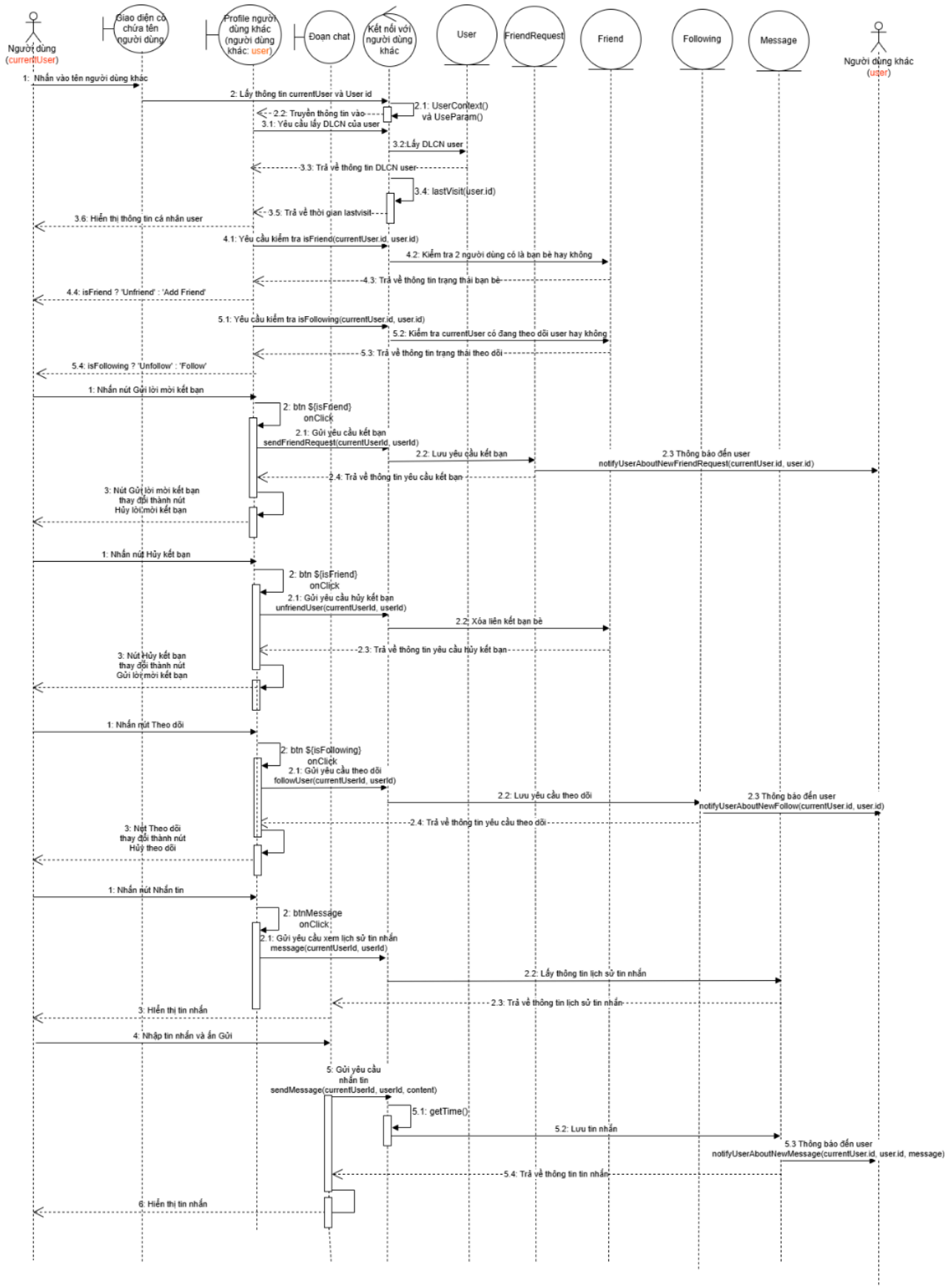


3.2.4 Use case kết nối người dùng

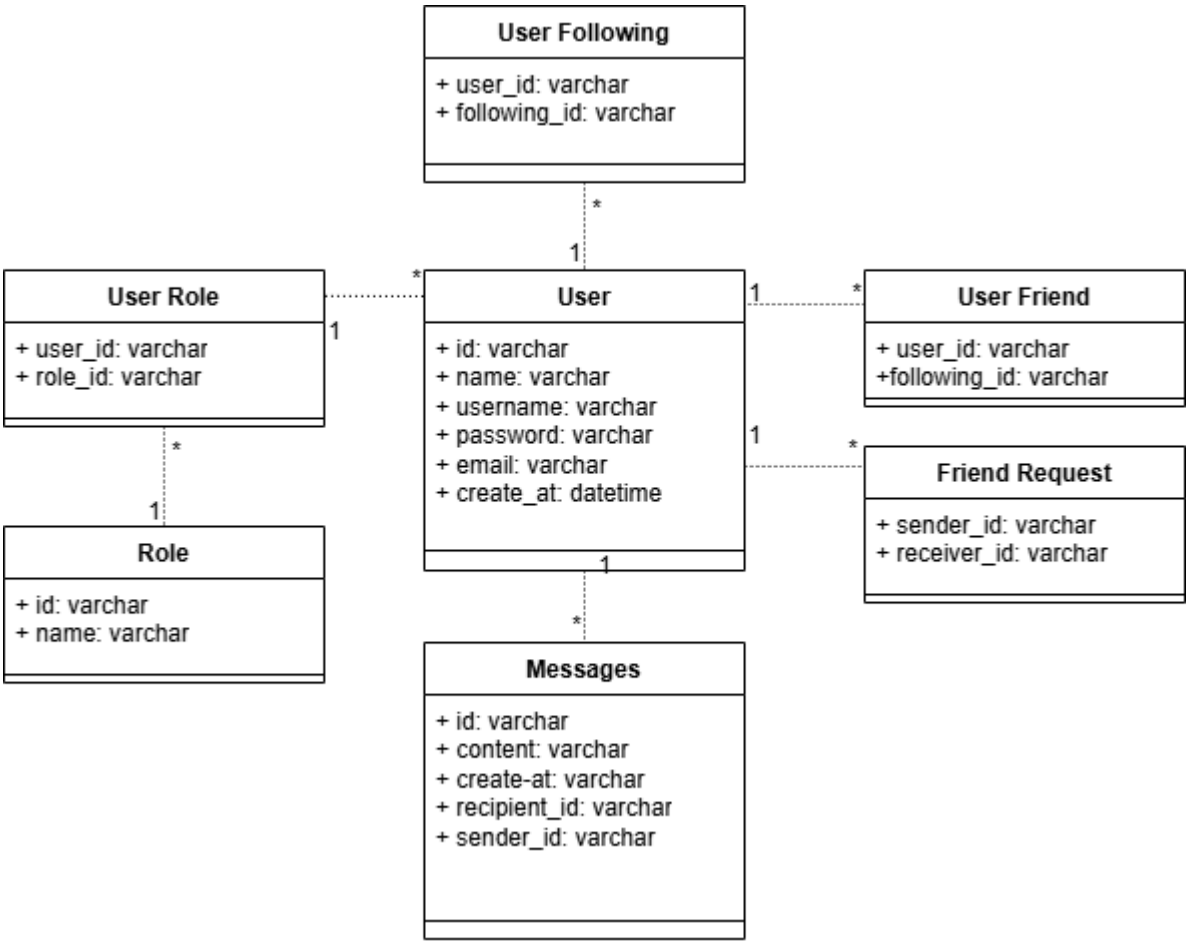
Use case: kết nối người dùng
Id: U03

Ý nghĩa: Người dùng có thể tương tác với người dùng khác qua nhiều cách khác nhau như kết bạn, theo dõi, nhắn tin và xem các bài đăng do người mình theo dõi đăng tải qua trang FollowingFeed

Sơ đồ tuần tự:


Người dùng (currentUser)
Giao diện có chứa tên người dùng
Profile người dùng khác (người dùng khác: user)
Đoạn chat
Kết nối với người dùng khác
User
FriendRequest
Friend
Following
Message
Người dùng khác (user)
1: Nhấn vào tên người dùng khác
2: Lấy thông tin currentUser và User id
2.1: UserContext() và UseParam()
2.2: Truyền thông tin vào
3.1: Yêu cầu lấy DLCN của user
3.2:Lấy DLCN user
3.3: Trả về thông tin DLCN user
3.4: lastVisit(user.id)
3.5: Trả về thời gian lastvisit
3.6: Hiển thị thông tin cá nhân user
4.1: Yêu cầu kiểm tra isFriend(currentUser.id, user.id)
4.2: Kiểm tra 2 người dùng có là bạn bè hay không
4.3: Trả về thông tin trạng thái bạn bè
4.4: isFriend ? 'Unfriend' : 'Add Friend'
5.1: Yêu cầu kiểm tra isFollowing(currentUser.id, user.id)
5.2: Kiểm tra currentUser có đang theo dõi user hay không
5.3: Trả về thông tin trạng thái theo dõi
5.4: isFollowing ? 'Unfollow' : 'Follow'
1: Nhấn nút Gửi lời mời kết bạn
2: btn ${isFriend} onClick
2.1: Gửi yêu cầu kết bạn sendFriendRequest(currentUserId, userId)
2.2: Lưu yêu cầu kết bạn
2.3 Thông báo đến user notifyUserAboutNewFriendRequest(currentUser.id, user.id)
2.4: Trả về thông tin yêu cầu kết bạn
3: Nút Gửi lời mời kết bạn thay đổi thành nút Hủy lời mời kết bạn
1: Nhấn nút Hủy kết bạn
2: btn ${isFriend} onClick
2.1: Gửi yêu cầu hủy kết bạn unfriendUser(currentUserId, userId)
2.2: Xóa liên kết bạn bè
2.3: Trả về thông tin yêu cầu hủy kết bạn
3: Nút Hủy kết bạn thay đổi thành nút Gửi lời mời kết bạn
1: Nhấn nút Theo dõi
2: btn ${isFollowing} onClick
2.1: Gửi yêu cầu theo dõi followUser(currentUserId, userId)
2.2: Lưu yêu cầu theo dõi
2.3 Thông báo đến user notifyUserAboutNewFollow(currentUser.id, user.id)
2.4: Trả về thông tin yêu cầu theo dõi
3: Nút Theo dõi thay đổi thành nút Hủy theo dõi
1: Nhấn nút Nhắn tin
2: btnMessage onClick
2.1: Gửi yêu cầu xem lịch sử tin nhắn message(currentUserId, userId)
2.2: Lấy thông tin lịch sử tin nhắn
2.3: Trả về thông tin lịch sử tin nhắn
3: Hiển thị tin nhắn
4: Nhập tin nhắn và ấn Gửi
5: Gửi yêu cầu nhắn tin sendMessage(currentUserId, userId, content)
5.1: getTime()
5.2: Lưu tin nhắn
5.3 Thông báo đến user notifyUserAboutNewMessage(currentUser.id, user.id, message)
5.4: Trả về thông tin tin nhắn
6: Hiển thị tin nhắn

Sơ đồ Lớp:



3.3 Yêu cầu và ràng buộc đối với phần mềm

3.3.1 Yêu cầu từ môi trường nghiệp vụ

Usecase	Id	Nội dung yêu cầu	Stack-Holder
UC01	B01.01	**Yêu cầu về hiển thị bài viết.** Bài viết cần hiển thị đầy đủ nội dung, tên tác giả, ngày đăng, tag.	Tác giả
	B01.02	**Yêu cầu về hiển thị tương tác bài viết.** Bài viết cần hiển thị đầy lượt like/disklike, bình luận.	Tác giả
UC02	B02.01	**Yêu cầu về hiển thị bình luận.** Bình luận cần hiển thị đầy đủ nội dung, tên người bình luận, thời gian bình luận	Người đọc
	B02.02	**Yêu cầu về hiển thị tương tác bình luận.** Bình luận cần hiển thị đầy lượt like/disklike, trả lời bình luận.	Người đọc
UC03	B03.01	**Yêu cầu về kết bè.**	Người dùng

		Lời mời kết bạn cần phải hiển thị thông tin người gửi, thời gian gửi	
	B03.02	**Yêu cầu nhắn tin.** Tin nhắn phải hiển thị đầy đủ nội dung, thời gian, người gửi	Người dùng

3.3.2 Yêu cầu từ môi trường vận hành

3.3.2.1 Yêu cầu chức năng

Usecase	Id	Nội dung yêu cầu	Stack-Holder
UC01	F01.01	**Yêu cầu về xem bài viết.** Ứng dụng phải hiển thị đầy đủ thông tin về tác giả và thời gian đăng tải của mỗi bài viết.	Tác giả
	F01.02	**Yêu cầu về tìm kiếm bài viết.** Người dùng có thể tìm kiếm bài viết theo từ khóa, tác giả hoặc thời gian đăng.	Tác giả
UC02	F02.01	**Yêu cầu về bình luận.** Người dùng có thể thêm, sửa, hoặc xóa bình luận của mình trên các bài viết.	Người đọc
	F02.02	**Yêu cầu về tương tác (like, dislike).** Người dùng có thể like hoặc dislike các bài viết và có khả năng bỏ like hoặc dislike đã chọn.	Người đọc
UC03	F03.01	**Yêu cầu về kết nối bạn bè.** Người dùng có thể gửi và nhận yêu cầu kết bạn từ các người dùng khác.	Người dùng
	F03.02	**Yêu cầu nhắn tin.** Người dùng có thể gửi tin nhắn cá nhân cho bạn bè của mình và nhận được phản hồi trong thời gian thực.	Người dùng

3.3.2 Yêu cầu chất lượng

Usecase	Id	Nội dung yêu cầu	Stack-Holder
UC01	F01.01	**Availability:** Ứng dụng phải đảm bảo có sẵn 24/7 để người dùng có thể xem bài viết bất kỳ lúc nào.	Tác giả

	F01.02	**Performance Efficiency:** Tìm kiếm bài viết phải trả kết quả trong vòng ≤ 1 giây, dù có lượng dữ liệu lớn.	Tác giả
UC02	F02.01	**Usability:** Giao diện bình luận phải dễ sử dụng, cho phép người dùng dễ dàng thêm, sửa hoặc xóa bình luận.	Người đọc
	F02.02	**Scalability:** Hệ thống cần có khả năng mở rộng để xử lý một lượng lớn bình luận và tương tác (like, dislike) mà không làm giảm hiệu suất.	Người đọc
UC03	F03.01	**Reliability:** Hệ thống phải đảm bảo rằng yêu cầu kết bạn luôn được gửi và nhận chính xác mà không bị gián đoạn.	Người dùng
	F03.02	**Maintainability:** Hệ thống cần dễ dàng mở rộng và bảo trì khi thêm tính năng nhắn tin hoặc thay đổi quy trình kết bạn.	Người dùng

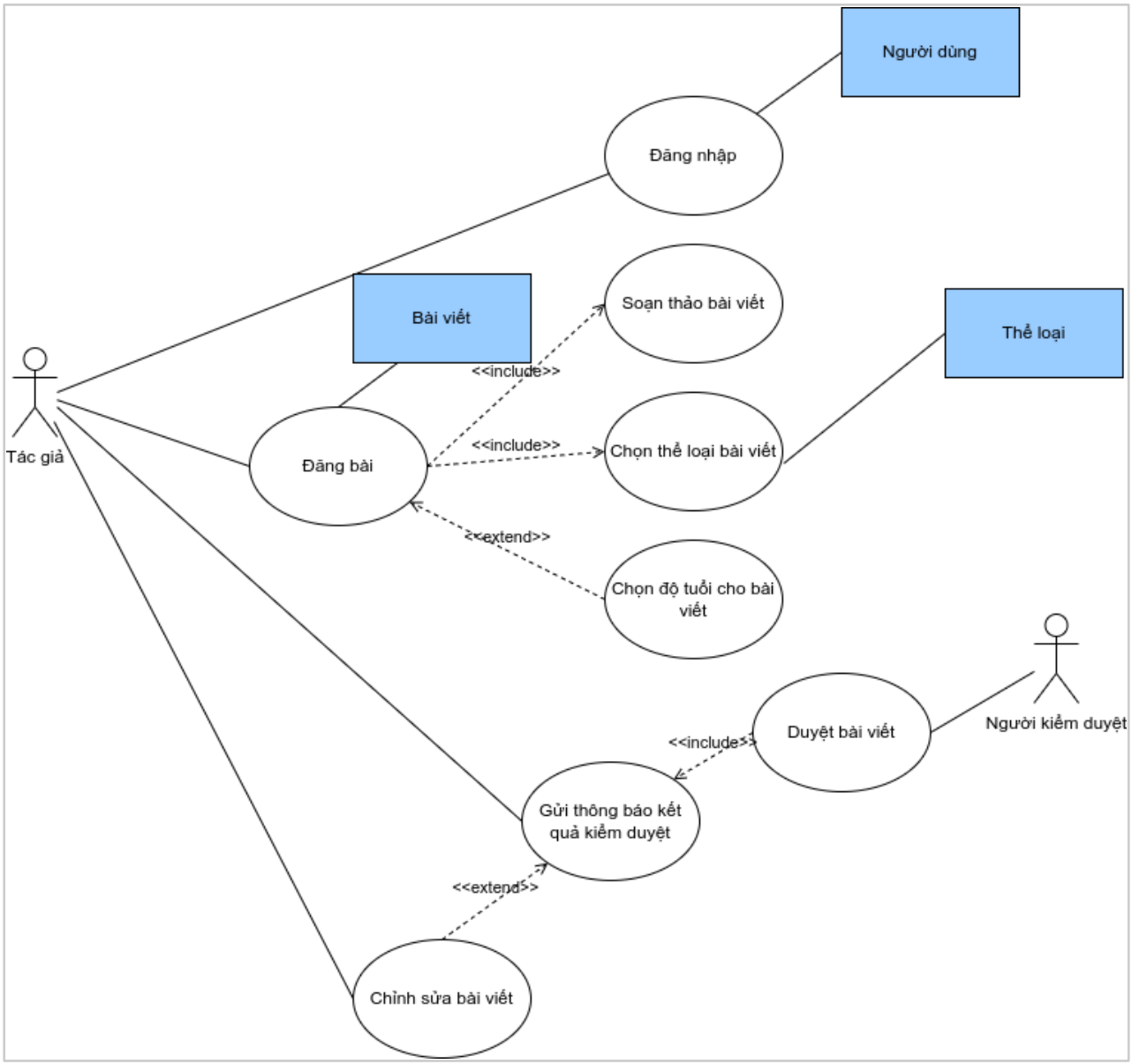
3.4 Yêu cầu từ môi trường phát triển

Id	Đối tượng	Nội dung yêu cầu	Stack-Holder
TA01	Hệ điều hành	Ubuntu 20.04 LTS	Nhân viên, quản lý
TD02	IDE	Intelij idea , VS code	Dev team
TD03	Server	ReactJS, NodeJS, Spring boot	Dev team

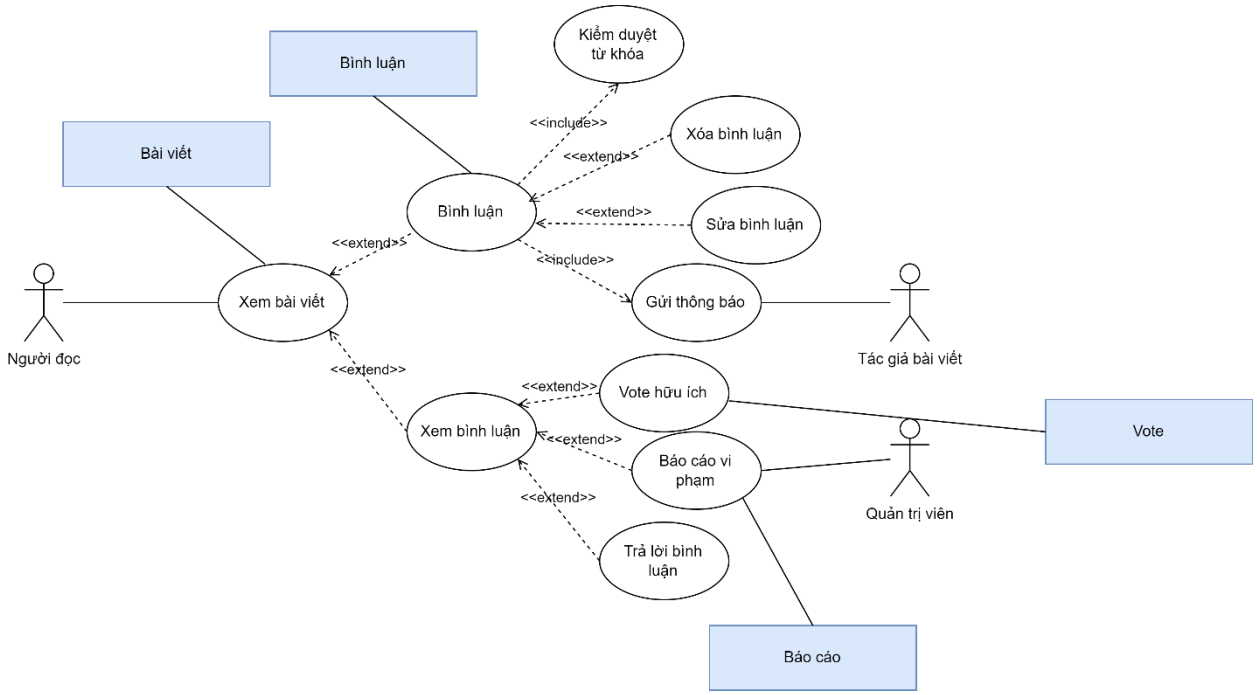
CHƯƠNG 4: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

4.1 Lược đồ use case cho giải pháp

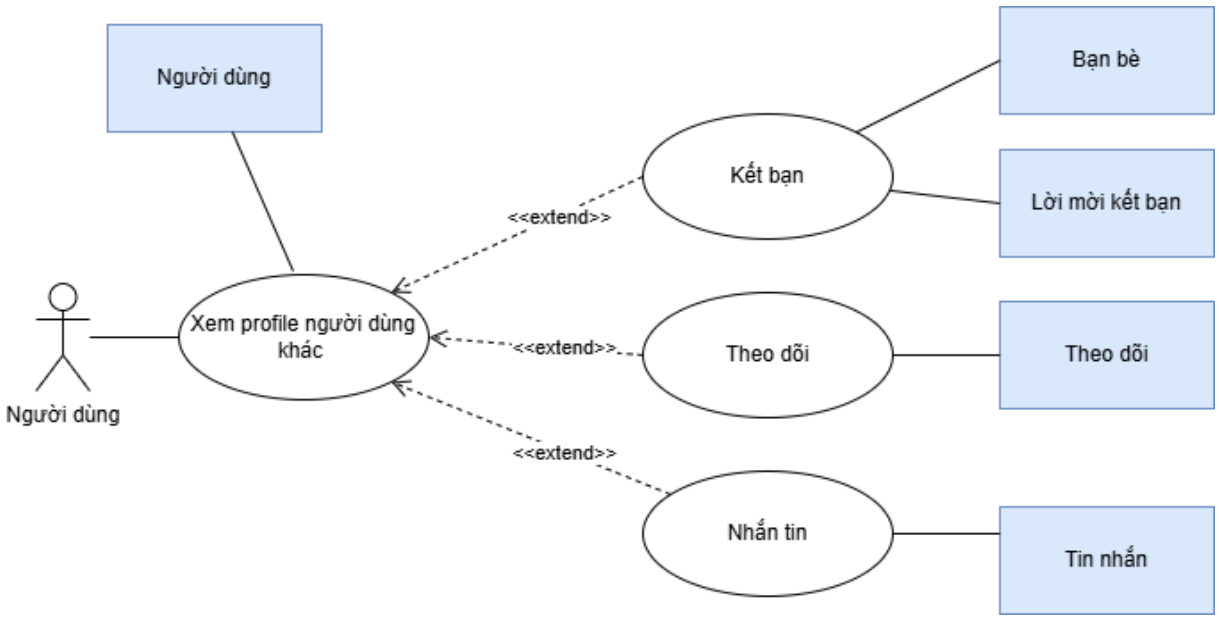
4.1.1 Use case đăng bài



4.1.2 Use case bình luận bài viết



4.1.3 Use case kết nối người dùng

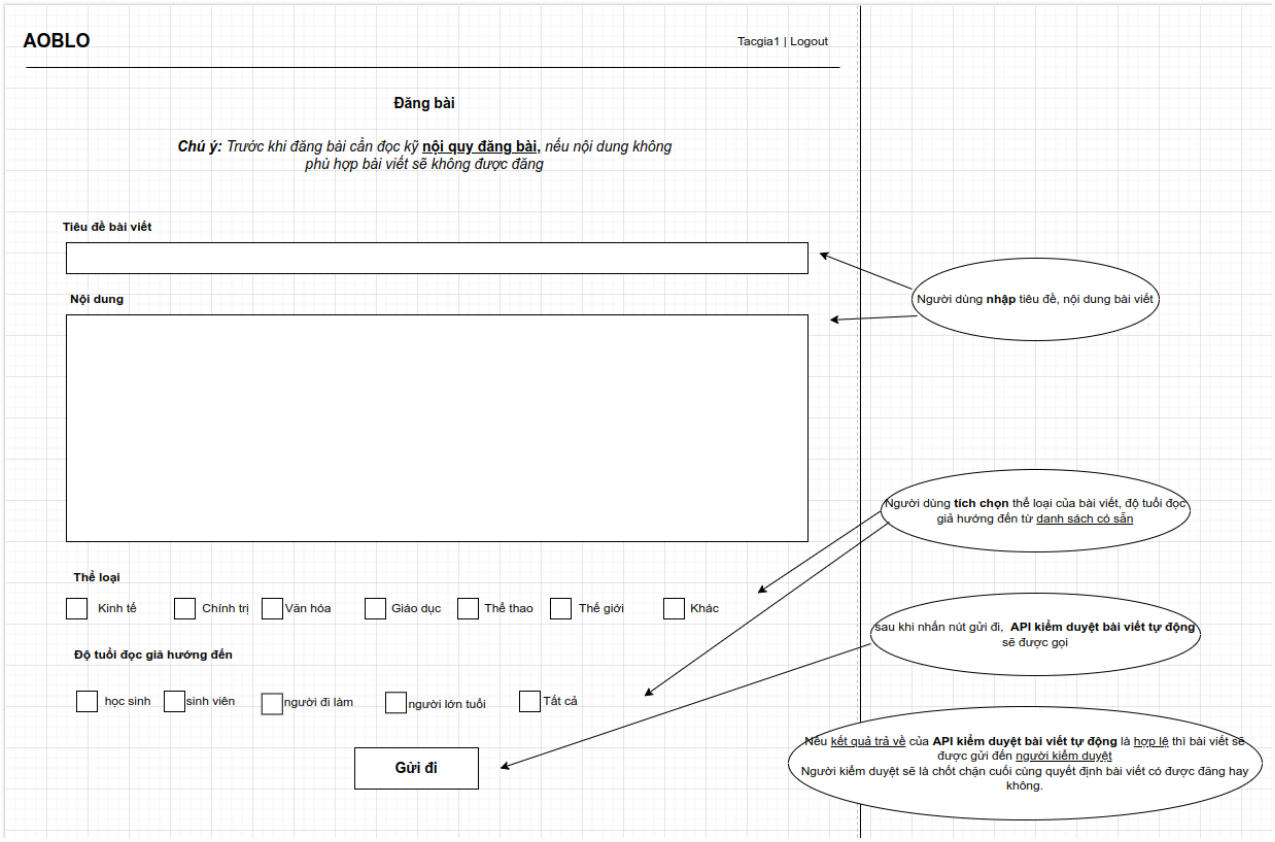


4.2 Thiết kế phần mềm

4.2.1 Form

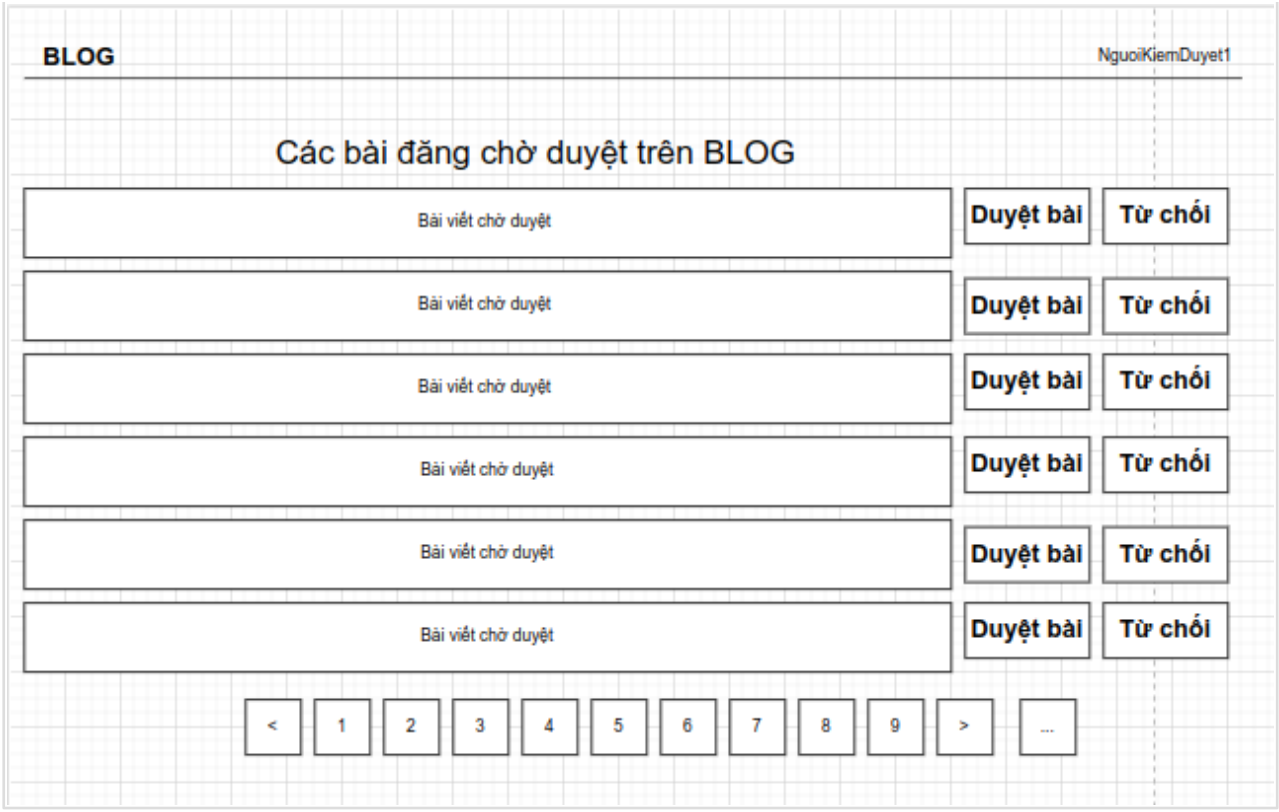
4.2.1.1 Form đăng bài (ID: F01)

- Form tác giả:



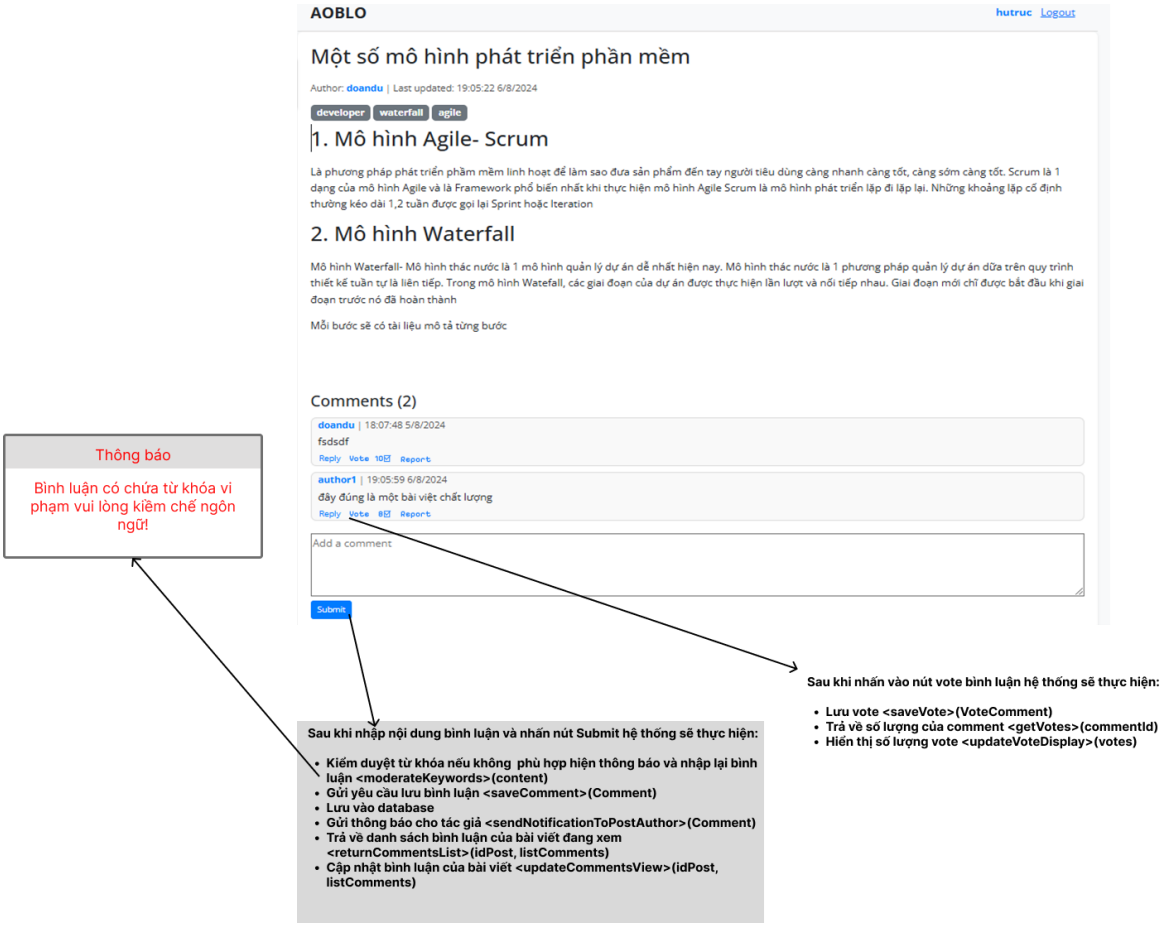
Sau khi nhập đầy đủ thông tin tiêu đề, nội dung, chọn thể loại, chọn độ tuổi, tác giả gửi bài viết đi, nếu kết quả kiểm duyệt tự động trả về hợp lệ thì bài viết sẽ được thêm vào danh sách bài viết chờ duyệt, và bài viết sẽ chờ người kiểm duyệt phê duyệt trước khi đăng lên BLOG.

- Form người kiểm duyệt

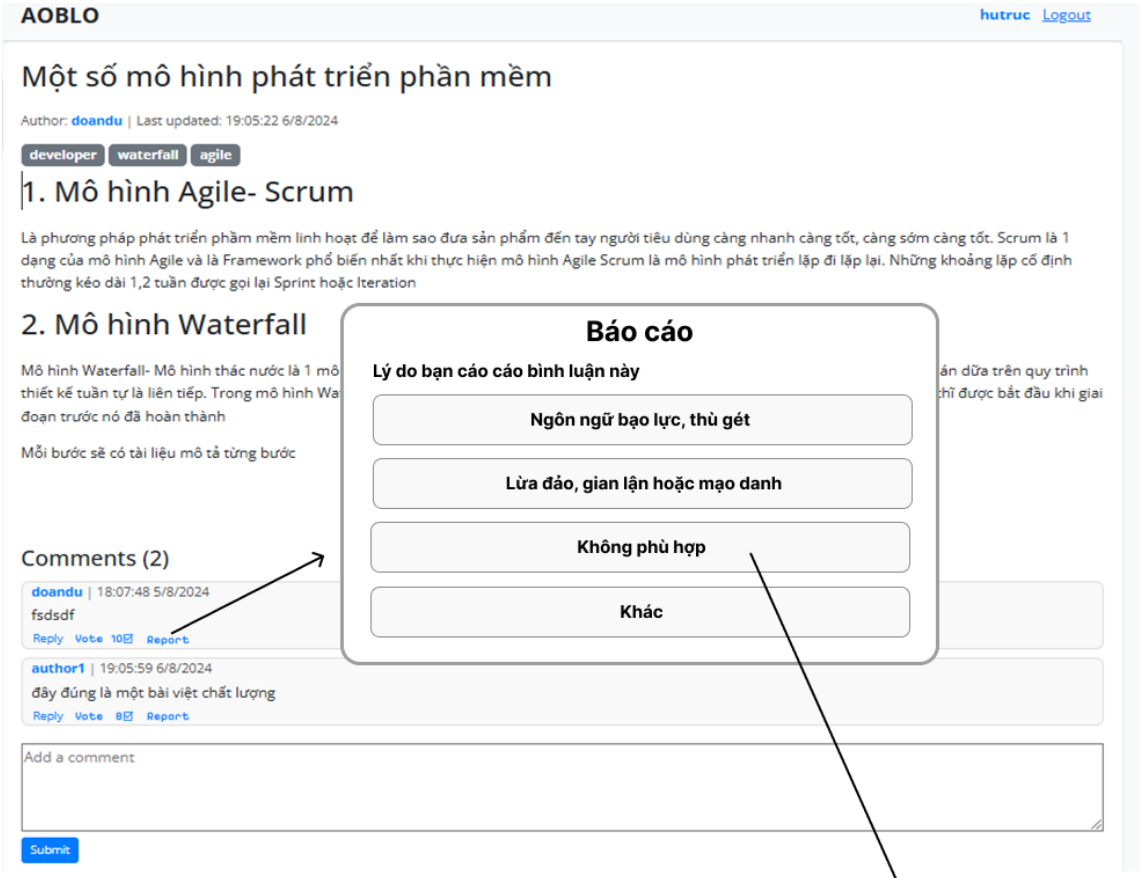


Sau khi người kiểm duyệt duyệt bài, thông báo sẽ được gửi về tác giả, nếu bài viết hợp lệ sẽ được thông báo đăng thành công, ngược lại yêu cầu tác giả chỉnh sửa bài viết.

4.2.1.2 Form bình luận bài viết (ID: F02)



4.2.1.3 Form báo cáo bình luận (ID: F03)

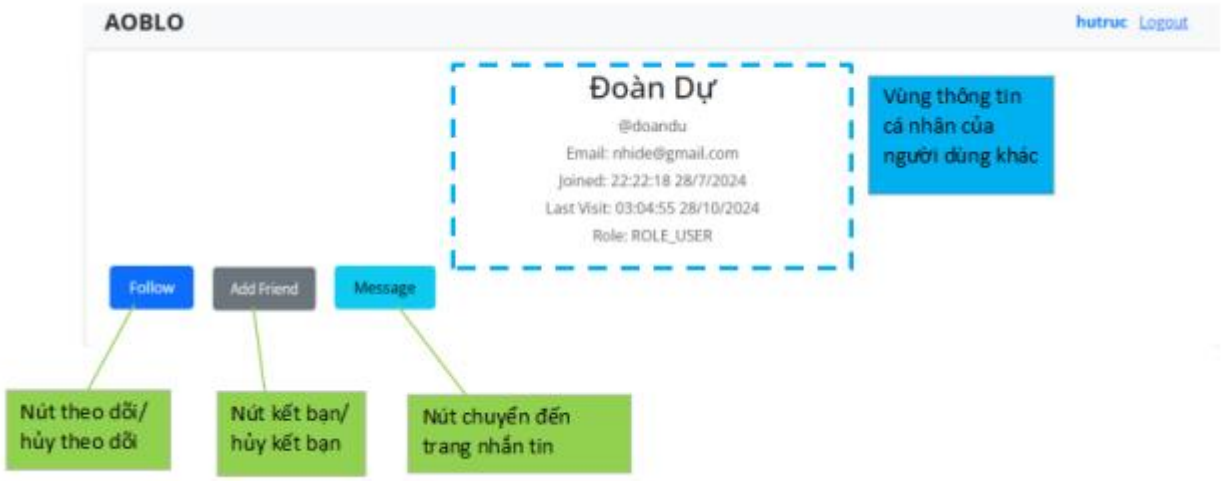


Sau khi nhấn vào nút report và chọn lý do hệ thống sẽ thực hiện:

- **Lưu report <saveReport>(reportComment)**
- **Gửi thông báo cho admin <sendReportToAdmin>(reportComment)**

4.2.1.4 Trang cá nhân người dùng khác (ID: F04)

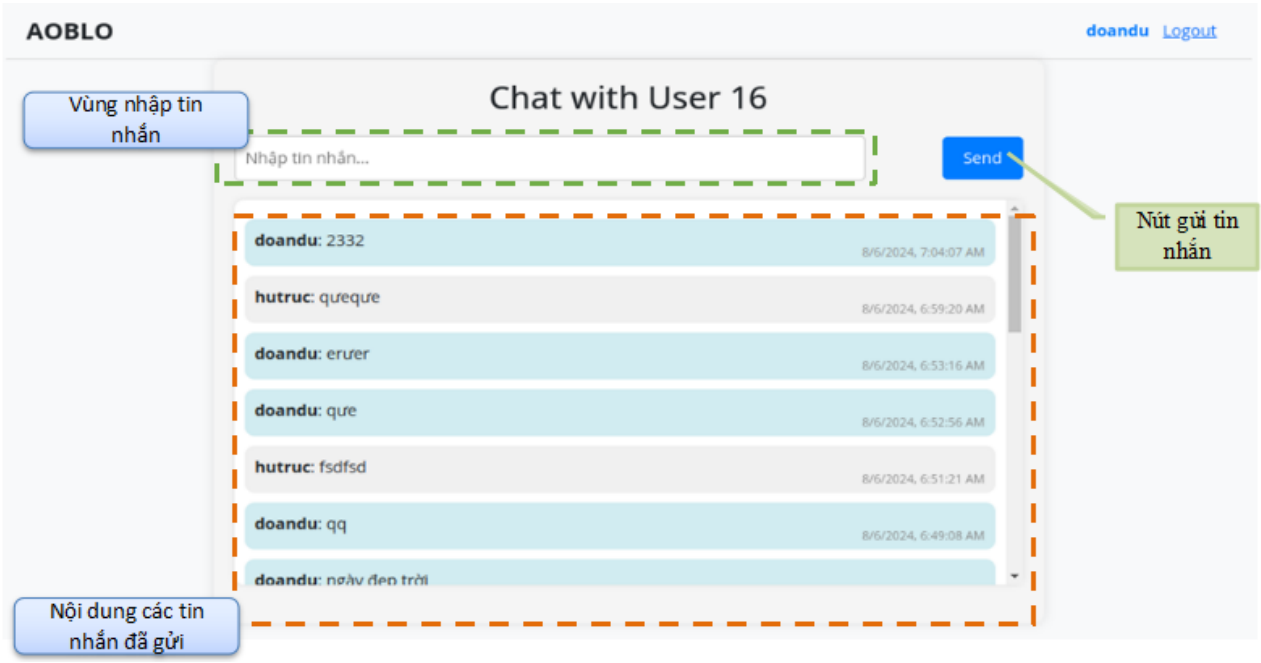
Form sử dụng cho Usecase: U03



Button ID 1: btn-isFollowing
Button ID 2: btn-isFriend
Button ID 3: btn-message

4.2.1.5 Form nhắn tin (ID: F05)

Form sử dụng cho Usecase: U03



TextBox ID: tbx-messageContent
Button ID: btn-sendMessage

4.2.2 API

4.2.2.1 API kiểm duyệt bài viết tự động

Tên	automaticallyModeratePost
Method	POST
Input	title: tiêu đề bài viết cần kiểm duyệt (dạng text) content: nội dung bài viết cần kiểm duyệt (dạng text)
Output	Trả về kết quả kiểm duyệt (hợp lệ hay không hợp lệ). **Violation**: **Boolean** (**true** hoặc **false**), trong đó: - **true**: Nếu nội dung chứa từ khóa cấm. - **false**: Nếu nội dung không chứa từ khóa cấm.
Mô tả	1. **Lấy danh sách từ khóa cấm:** API sẽ truy vấn từ cơ sở dữ liệu để lấy danh sách các từ khóa cấm từ bảng **BlockedKeyword**. Các từ khóa này được lưu trữ dưới dạng ***varchar*** trong cơ sở dữ liệu.

	2. **Kiểm duyệt nội dung:** API nhận đầu vào là **title**, **content** (tiêu đề, nội dung) dưới dạng ***text***. Duyệt qua từng từ khóa trong danh sách từ khóa cấm và kiểm tra xem có chứa từ khóa nào không. Nếu tìm thấy từ khóa cấm trong nội dung, API sẽ đánh dấu là vi phạm. 3. **Kết quả trả về:** Nếu nội dung chứa bất kỳ từ khóa cấm nào, API trả về kết quả là violation: **true**. Nếu không có từ khóa cấm nào trong nội dung, API trả về kết quả là violation: **false**.

4.2.2.2 API Lưu bài viết vào danh sách chờ duyệt.

Tên	addPostToWaitingList
Method	POST
Input	post: Đối tượng bài viết chứa các trường cần thiết để lưu vào cơ sở dữ liệu. - **title**: Tiêu đề bài viết (dạng **text**). - **content**: Nội dung bài viết (dạng **text**). - **idAuthor**: ID của người viết bài (dạng **varchar**) - **idPost**: ID bài viết (dạng **varchar**). - **createdAt**: Thời gian tạo bài viết (dạng **Date**). - **publish**: Trạng thái bài viết(dạng **Boolean**).
Output	Trả về kết quả lưu thành công hoặc thất bại. **Violation**: **Boolean** (**true** hoặc **false**), trong đó: - **true**: Lưu thành công. - **false**: Lưu thất bại.
Mô tả	1. Lưu bài viết vào cơ sở dữ liệu: API sẽ lưu bài viết vào cơ sở dữ liệu bằng cách tạo một đối tượng mới và lưu vào bảng Posts. 2. Trả về kết quả: API sẽ trả về một thông báo thông báo kết quả thành công hoặc thất bại, và dữ liệu của bài viết đã được lưu (hoặc thông tin lỗi).

4.2.2.3 API Lưu bài viết vào danh sách bài đăng đã được kiểm duyệt

Tên	postModerated
Method	PUT
Input	post: Đối tượng bài viết chứa các trường cần thiết để lưu vào cơ sở dữ liệu. - **title**: Tiêu đề bài viết (dạng **text**). - **content**: Nội dung bài viết (dạng **text**). - **idAuthor**: ID của người viết bài (dạng **varchar**)

	- **idPost**: ID bài viết (dạng **varchar**). - **createdAt**: Thời gian tạo bài viết (dạng **Date**). - **publish**: Trạng thái bài viết(dạng **Boolean**).
Output	Trả về kết quả lưu thành công hoặc thất bại. **Violation**: **Boolean** (**true** hoặc **false**), trong đó: - **true**: Lưu thành công. - **false**: Lưu thất bại.
Mô tả	1. Lưu bài viết vào cơ sở dữ liệu: API sẽ lưu bài viết vào cơ sở dữ liệu bằng cách thay đổi Trạng thái bài viết (Publish) từ false thành true (true là bài viết đã được kiểm duyệt). 2. Trả về kết quả: API sẽ trả về một thông báo thông báo kết quả thành công hoặc thất bại, và dữ liệu của bài viết đã được lưu (hoặc thông tin lỗi).

4.2.2.4 API người kiểm duyệt bài viết

Tên	postModerator
Method	POST
Input	post: Đối tượng post_moderator chứa các trường cần thiết để lưu vào cơ sở dữ liệu. - **id**Moderator: ID của người viết bài (dạng **varchar**) - **idPost**: ID bài viết (dạng **varchar**). - status: kết quả kiểm duyệt (dạng **boolean**)
Output	Trả về kết quả lưu thành công hoặc thất bại. **Violation**: **Boolean** (**true** hoặc **false**), trong đó: - **true**: Lưu thành công. - **false**: Lưu thất bại.
Mô tả	1. Lưu vào cơ sở dữ liệu: API sẽ lưu vào cơ sở dữ liệu 2. Trả về kết quả: API sẽ trả về một thông báo thông báo kết quả thành công hoặc thất bại, và dữ liệu đã được lưu (hoặc thông tin lỗi).

4.2.2.5 API kiểm duyệt từ khóa

Tên	moderateKeywords
Method	POST
Input	**content**: Nội dung bình luận cần kiểm duyệt (dạng **text**).
Output	Trả về kết quả kiểm duyệt (hợp lệ hay không hợp lệ).

	Violation: **Boolean** (**true** hoặc **false**), trong đó: - **true**: Nếu nội dung chứa từ khóa cấm. - **false**: Nếu nội dung không chứa từ khóa cấm.
Mô tả	1. **Lấy danh sách từ khóa cấm:** API sẽ truy vấn từ cơ sở dữ liệu để lấy danh sách các từ khóa cấm từ bảng **BlockedKeyword**. Các từ khóa này được lưu trữ dưới dạng ***varchar*** trong cơ sở dữ liệu. 2. **Kiểm duyệt nội dung:** API nhận đầu vào là **content** (nội dung bình luận) dưới dạng ***text***. Duyệt qua từng từ khóa trong danh sách từ khóa cấm và kiểm tra xem nội dung có chứa từ khóa nào không. Nếu tìm thấy từ khóa cấm trong nội dung, API sẽ đánh dấu là vi phạm. 3. **Kết quả trả về:** Nếu nội dung chứa bất kỳ từ khóa cấm nào, API trả về kết quả là violation: **true**. Nếu không có từ khóa cấm nào trong nội dung, API trả về kết quả là violation: **false**.

4.2.2.6 API lưu bình luận

Tên	saveComment
Method	POST
Input	**comment**: Đối tượng bình luận chứa các trường cần thiết để lưu vào cơ sở dữ liệu. - **content**: Nội dung bình luận (dạng **text**). - **idAuthor**: ID của người viết bình luận (dạng **varchar**) - **idPost**: ID bài viết mà bình luận liên quan (dạng **varchar**). - **createdAt**: Thời gian tạo bình luận (dạng **Date**).
Output	**status**: Trạng thái thực hiện API ("success" hoặc "fail"). **message**: Thông báo kết quả thực hiện API (ví dụ: "Bình luận được lưu thành công" hoặc "Không thể lưu bình luận").
Mô tả	1. **Lưu bình luận vào cơ sở dữ liệu:** API sẽ lưu bình luận vào cơ sở dữ liệu bằng cách tạo một đối tượng mới và lưu vào bảng Comments. 2. **Trả về kết quả:** API sẽ trả về một thông báo thông báo kết quả thành công hoặc thất bại, và dữ liệu của bình luận đã được lưu (hoặc thông tin lỗi).

4.2.2.7 API Tạo report

Tên	createReport
Method	POST
Input	**report**: Đối tượng chứa thông tin về báo cáo. - **idAuthor**: ID của người tạo báo cáo (**varchar**).

	- **idComment**: ID của bình luận bị báo cáo (**varchar**). - **idReason**: ID của lý do báo cáo (**varchar**). - **createdAt**: Thời gian tạo báo cáo (dạng **Date**).
Output	**status**: Trạng thái thực hiện API ("success" hoặc "fail"). **message**: Thông báo kết quả thực hiện API (ví dụ: "Báo cáo thành công" hoặc "Không thể tạo báo cáo").
Mô tả	1. **Lưu báo cáo vào cơ sở dữ liệu:** API tạo một đối tượng báo cáo mới trong bảng Reports trong cơ sở dữ liệu. 2. **Trả về kết quả:** API sẽ trả về một thông báo thông báo kết quả thành công hoặc thất bại, và dữ liệu của báo cáo đã được tạo (hoặc thông tin lỗi).

4.2.2.8 API lấy thông tin cơ bản của một user

API sử dụng cho việc lấy dữ liệu hiển thị ban đầu cho "**Vùng thông tin cá nhân của người dùng khác**" ở **Form F04** sau khi click vào tên người dùng

Tên	getUserInfo
Method	GET
Input	User ID: ID của người dùng cần lấy thông tin (dạng số nguyên, nằm trong đường dẫn URL).
Output	**JSON Object**: Thông tin người dùng bao gồm: `id` (Number): ID của người dùng. `name` (String): Tên người dùng. `username` (String): Tên đăng nhập. `email` (String): Địa chỉ email. `createAt` (String): Thời gian tạo tài khoản (định dạng ISO 8601). `lastVisit` (String): Lần truy cập gần nhất (định dạng ISO 8601). `roles` (Array of Objects): Các vai trò của người dùng. Mỗi vai trò bao gồm: `name` (String): Tên của vai trò.
Mô tả	API này lấy thông tin chi tiết của một người dùng dựa trên ID được cung cấp trong URL. Kết quả trả về bao gồm thông tin cơ bản như tên, tên đăng nhập, email, thời gian tạo tài khoản, lần truy cập gần nhất và danh sách các vai trò mà người dùng đảm nhiệm.

4.2.2.9 API kiểm tra xem có đang follow user đó không

API sử dụng cho việc xác định trạng thái của **btn-isFollowing** thuộc **Form F04**

Tên	isFollowing
Method	GET

Input	User ID (dạng số nguyên): ID của người dùng cần kiểm tra xem có theo dõi người dùng khác hay không. **Authorization**: Token JWT trong header `Authorization` (Bearer Token).
Output	**Boolean**: Trả về `true` nếu người dùng hiện tại đang theo dõi người dùng có ID là `userId`, ngược lại trả về `false`.
Mô tả	API này kiểm tra xem người dùng hiện tại có theo dõi một người dùng khác dựa trên `userId` được cung cấp trong URL. Nếu người dùng hiện tại đang theo dõi người dùng với ID `userId`, trả về `true`. Nếu không theo dõi, trả về `false`. **Header Authorization** **Authorization**: Bearer {JWT Token} Token này xác thực người dùng hiện tại và đảm bảo tính bảo mật khi kiểm tra theo dõi giữa các người dùng.

4.2.2.10 API Kiểm tra xem user đó đã kết bạn chưa

API sử dụng cho việc xác định trạng thái của **btn-isFriend** thuộc **Form F04**

Tên	isFriend
Method	GET
Input	User ID (dạng số nguyên): ID của người dùng cần kiểm tra xem có phải là bạn của người dùng hiện tại hay không. **Authorization**: Token JWT trong header `Authorization` (Bearer Token).
Output	**Boolean**: Trả về `true` nếu người dùng hiện tại là bạn của người dùng có ID là `userId`, ngược lại trả về `false`.
Mô tả	API này kiểm tra xem người dùng hiện tại có phải là bạn của một người dùng khác dựa trên `userId` được cung cấp trong URL. Nếu người dùng hiện tại và người dùng có ID `userId` là bạn bè, trả về `true`. Nếu không phải bạn bè, trả về `false`. **Header Authorization** **Authorization**: Bearer {JWT Token} Token này xác thực người dùng hiện tại và đảm bảo tính bảo mật khi kiểm tra mối quan hệ bạn bè giữa hai người dùng.

4.2.2.11 API Theo dõi(follow) một user

API sử dụng cho thao tác click vào **btn-isFollowing** thuộc **Form F04** khi trạng thái isFollowing = **false**

Tên	Follow
Method	POST
Input	User ID (dạng số nguyên): ID của người dùng cần kiểm tra xem có phải là bạn của người dùng hiện tại hay không. **Authorization**: Token JWT trong header `Authorization` (Bearer Token).
Output	Trả về một thông điệp xác nhận rằng người dùng hiện tại đã bắt đầu theo dõi người dùng với ID `userId`.
Mô tả	API này cho phép người dùng hiện tại bắt đầu theo dõi một người dùng khác (lưu vào cơ sở dữ liệu), thông qua việc gửi yêu cầu POST với ID của người dùng mà bạn muốn theo dõi trong URL. Sau khi API thực hiện thành công, nó sẽ trả về thông báo xác nhận rằng người dùng đã bắt đầu theo dõi người dùng có ID `userId`. **Header Authorization** **Authorization**: Bearer {JWT Token} Token này được dùng để xác thực người dùng thực hiện hành động theo dõi.

4.2.2.12 API gửi yêu cầu kết bạn

API sử dụng cho thao tác click vào **btn-isFriend** thuộc **Form F04** khi trạng thái isFriend = **false**

Tên	sendFriendRequest
Method	POST
Input	User ID (dạng số nguyên): ID của người dùng mà bạn muốn gửi yêu cầu kết bạn **Authorization**: Token JWT trong header `Authorization` (Bearer Token).
Output	Trả về thông điệp xác nhận rằng yêu cầu kết bạn đã được gửi đến người dùng có ID `userId`.
Mô tả	API này cho phép người dùng gửi yêu cầu kết bạn đến một người dùng khác (lưu vào cơ sở dữ liệu). Yêu cầu được thực hiện thông qua một

	POST request với `userId` của người dùng mà bạn muốn gửi yêu cầu kết bạn. Sau khi API thực hiện thành công, nó sẽ trả về thông báo xác nhận rằng yêu cầu kết bạn đã được gửi đến người dùng có ID `userId` **Header Authorization** **Authorization**: Bearer {JWT Token} Token này được dùng để xác thực người dùng thực hiện hành động gửi yêu cầu kết bạn.

4.2.2.13 API hủy theo dõi user cụ thể

API sử dụng cho thao tác click vào **btn-isFollowing** thuộc **Form F04** khi trạng thái isFollowing = **true**

Tên	unfollow
Method	DELETE
Input	User ID (dạng số nguyên): ID của người dùng mà bạn muốn hủy theo dõi. **Authorization**: Token JWT trong header `Authorization` (Bearer Token).
Output	Trả về thông điệp xác nhận rằng bạn đã hủy theo dõi người dùng có ID `userId`.
Mô tả	API này cho phép người dùng hủy theo dõi một người dùng khác (xóa khỏi cơ sở dữ liệu). Yêu cầu được thực hiện thông qua một DELETE request với `userId` của người dùng mà bạn muốn hủy theo dõi. Sau khi thực hiện thành công, API sẽ trả về thông báo xác nhận rằng bạn đã hủy theo dõi người dùng có ID `userId`. **Header Authorization** **Authorization**: Bearer {JWT Token} Token này được dùng để xác thực người dùng thực hiện hành động hủy theo dõi.

4.2.2.14 API hủy kết bạn user cụ thể

API sử dụng cho thao tác click vào **btn-isFriend** thuộc **Form F04** khi trạng thái isFriend = **true**

Tên	unfriend
Method	DELETE
Input	User ID (dạng số nguyên): ID của người dùng mà bạn muốn hủy kết bạn. **Authorization**: Token JWT trong header `Authorization` (Bearer Token).
Output	Trả về thông điệp xác nhận rằng bạn đã hủy kết bạn với người dùng có ID `userId`.
Mô tả	API này cho phép người dùng hủy kết bạn với một người dùng khác (xóa khỏi cơ sở dữ liệu). Yêu cầu được thực hiện thông qua một DELETE request với `userId` của người dùng mà bạn muốn hủy kết bạn. Sau khi thực hiện thành công, API sẽ trả về thông báo xác nhận rằng bạn đã hủy kết bạn với người dùng có ID `userId`. **Header Authorization** **Authorization**: Bearer {JWT Token} Token này được dùng để xác thực người dùng thực hiện hành động hủy kết bạn.

4.2.2.15 API lấy lịch sử tin nhắn với một user cụ thể

API sử dụng cho việc lấy dữ liệu ban đầu cho "**Vùng nội dung các tin nhắn đã gửi**" thuộc **Form F05** sau khi click vào **btn-message** ở Form F04

Tên	messageHistory
Method	GET
Input	User ID (dạng số nguyên): ID của người dùng mà bạn muốn lấy lịch sử tin nhắn. **Authorization**: Token JWT trong header `Authorization` (Bearer Token). **Query Parameters**: `pageNo` (dạng số nguyên): Số trang cần lấy (ví dụ: `pageNo=0`). `pageSize` (dạng số nguyên): Số lượng tin nhắn trên mỗi trang (ví dụ: `pageSize=10`).
Output	Trả về danh sách các tin nhắn giữa người dùng với `userId` cụ thể. Các tin nhắn được phân trang, với thông tin chi tiết như người gửi, người nhận, nội dung tin nhắn và thời gian gửi.

Mô tả	API này cho phép người dùng lấy lịch sử tin nhắn với người dùng có ID `userId`. Các tin nhắn sẽ được trả về dưới dạng phân trang với các tham số `pageNo` và `pageSize`. Các thông tin của mỗi tin nhắn bao gồm: `id`: ID của tin nhắn. `content`: Nội dung tin nhắn. `sender`: Thông tin người gửi (ID và username). `recipient`: Thông tin người nhận (ID và username). `createAt`: Thời gian tin nhắn được gửi. **Header Authorization** **Authorization**: Bearer {JWT Token} Token này được dùng để xác thực người dùng gửi yêu cầu lấy lịch sử tin nhắn.

4.2.2.16 API gửi một tin nhắn

API sử dụng cho thao tác nhập **content** (nội dung tin nhắn) vào **tbx-messageContent** và click **btn-sendMessage** ở **Form F05**

Tên	sendMessage
Method	POST
Input	User ID (dạng số nguyên): ID của người dùng mà bạn muốn gửi tin nhắn tới. **Authorization**: Token JWT trong header `Authorization` (Bearer Token). **Request Body**: `content` (dạng chuỗi): Nội dung của tin nhắn gửi đi được nhập vào tbx-mesageContent ở Form F05
Output	Trả về thông điệp xác nhận rằng tin nhắn đã được gửi đến người dùng có ID `userId`.
Mô tả	API này cho phép người dùng gửi tin nhắn đến người dùng có ID `userId` (lưu vào cơ sở dữ liệu) Tham số `content` trong body của yêu cầu chứa nội dung tin nhắn. Sau khi tin nhắn được gửi thành công, API sẽ trả về thông báo xác nhận rằng tin nhắn đã được gửi thành công.

	Header Authorization **Authorization**: Bearer {JWT Token} Token này sẽ được dùng để xác thực yêu cầu gửi tin nhắn từ người dùng.

4.2.3 Socket

4.2.3.1 Socket gửi thông báo report

Tên	sendReportNotificationToModerators
Input	**Room**: moderators((nơi tất cả kiểm duyệt viên được kết nối) **IdReport**: Id của report mới được tạo(varchar)
Output	Không trả về giá trị cho server. Thông báo sẽ được hiển thị trên giao diện của kiểm duyệt viên.
Mô tả	Socket event này được sử dụng để gửi thông báo report real-time đến tất cả kiểm duyệt viên (moderators). Sau khi một report được tạo, event này sẽ được phát đến room moderators qua Socket.IO, thông báo cho tất cả kiểm duyệt viên về report mới. **1. Khi tạo một report mới:** API tạo report gọi hàm gửi thông báo Socket.IO. Dữ liệu bao gồm idReport, thông điệp, và thời gian được gửi đến room moderators. **2. Socket.IO trên server phát sự kiện:** Sự kiện sendReportNotificationToModerators được gửi đến tất cả kiểm duyệt viên đã tham gia room moderators. **3. Client-side nhận sự kiện:** Các kiểm duyệt viên đang kết nối sẽ nhận thông báo và hiển thị trên giao diện.

4.2.3.2 Socket gửi thông báo tin nhắn mới

Socket sử dụng cho usecase U03

Tên	sendNewMessageNotification
Input	**Room**: `userId` (ID của người nhận tin nhắn). • Chỉ người nhận kết nối tới room này sẽ nhận được thông báo tin nhắn mới. **Message Data**: • `content` (string): Nội dung tin nhắn. • `senderId` (number): ID của người gửi. • `senderUsername` (string): Tên người gửi.

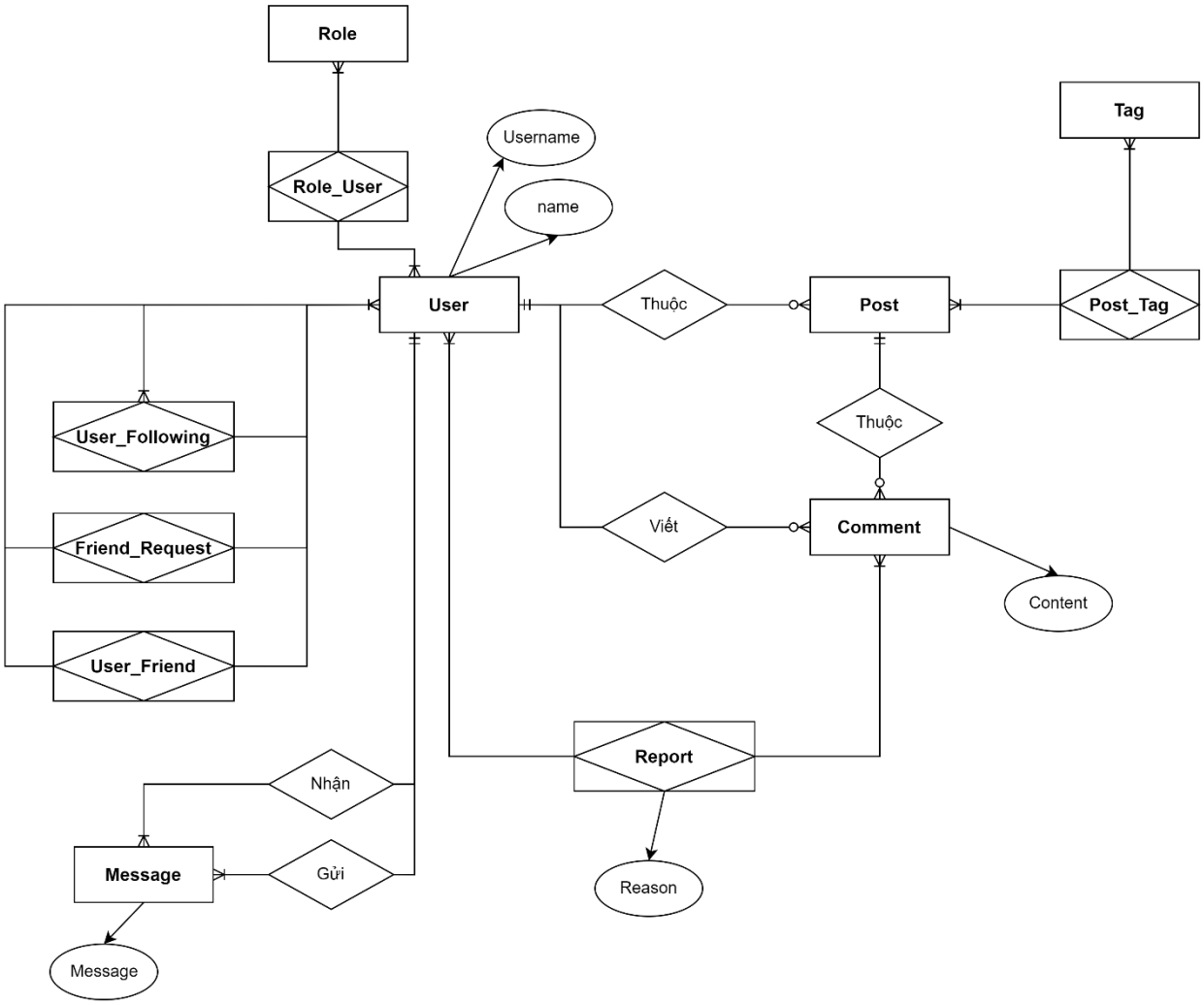
	• `createAt` (string - ISO datetime): Thời điểm tin nhắn được tạo.
Output	Không trả về giá trị cho server. Thông báo tin nhắn mới sẽ được hiển thị trên giao diện của người nhận, đồng thời danh sách tin nhắn sẽ được cập nhật tự động.
Mô tả	Socket event này được sử dụng để gửi thông báo real-time khi có tin nhắn mới đến một người dùng cụ thể. **Khi gửi tin nhắn mới**: Client phát sự kiện `messageClient` kèm theo `userId` của người nhận đến server Socket.IO. Server xử lý sự kiện, phát lại thông tin tin nhắn qua sự kiện `messageServer` đến room của người nhận. **Server phát sự kiện**: Sự kiện `messageServer` được phát đến client trong room `userId` để thông báo về tin nhắn mới. **Client-side nhận sự kiện**: Client người nhận lắng nghe sự kiện `messageServer`, sau đó tự động fetch API để cập nhật danh sách tin nhắn hiển thị trên giao diện. **Dòng chảy hoạt động** **Người gửi tin nhắn**: Gửi tin nhắn qua API (POST). Phát sự kiện `messageClient` lên server Socket.IO. **Server xử lý**: Nhận sự kiện `messageClient`. Phát sự kiện `messageServer` chứa thông tin tin nhắn mới đến room của người nhận. **Người nhận tin nhắn**: Nhận sự kiện `messageServer`. Fetch API để tải lại danh sách tin nhắn và hiển thị thông báo real-time.

4.3 Thiết kế cơ sở dữ liệu

4.3.1 Xác định thực thể

- USER(**ID**, TEN, USERNAME, PASSWORD, EMAIL, NGAYTAO)
- ROLE(**ID**, TEN)
- POST(**ID**, IDUSER, TIEUDE, NOIDUNG, NGAYTAO)
- TAG(**ID**, TEN)
- MESSAGE(**ID**, IDNGUOIGUI, IDNGUOINHAN, NOIDUNG, NGAYGUI)
- COMMENT(**ID**, IDPOST, IDUSER, NOIDUNG, IDCHA, NGAYTAO)

4.3.2 Mô hình ERD



4.3.3 Mô hình cơ sở dữ liệu quan hệ

- KHÓA CHÍNH: **IN ĐẬM GẠCH CHÂN**
- KHÓA NGOẠI: GẠCH CHÂN

- USER(**ID**, TEN, USERNAME, PASSWORD, EMAIL, NGAYTAO)
- ROLE(**ID**, TEN)
- USER_ROLE(IDUSER, IDROLE)
- POST(**ID**, IDUSER, TIEUDE, NOIDUNG, NGAYTAO)
- TAG(**ID**, TEN)
- MESSAGE(**ID**, IDNGUOIGUI, IDNGUOINHAN, NOIDUNG, NGAYGUI)
- POST_TAG(IDPOST, IDTAG)
- POST_DISLIKE(IDPOST, IDUSER)
- POST_LIKE(IDPOST, IDUSER)
- COMMENT(**ID**, IDPOST, IDUSER, NOIDUNG, IDCHA, NGAYTAO)
- COMMENT_LIKE(IDCOMMENT, IDUSER)
- COMMENT_DISLIKE(IDCOMMENT, IDUSER)
- FRIEND_REQUEST(IDNGUOIGUI, IDNGUOINHAN)
- USER_FRIEND(IDUSER, IDFRIEND)
- USER_FOLLOWING(IDUSER, IDUSERFOLLOW)
- REPORT(ID, IDUSER, IDCOMMENT, REASON)

• Thực thể USER: Thông tin user

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	ID	Id	bigint	Khóa chính	X
2	TEN	Tên người dùng	varchar(255)		X
3	USERNAME	Tên đăng nhập	varchar(20)		X
4	PASSWORD	Mật khẩu	varchar(64)		X
5	EMAIL	Email	varchar(320)		
6	NGAYTAO	Ngày tạo	datetime		X

• Thực thể ROLE: Thông tin các quyền

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	ID	Id	bigint	Khóa chính	X
2	TEN	Tên quyền	varchar(20)		X

• Thực thể USER_ROLE: Quyền của user

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull

1	IDUSER	Id user	bigint	Khóa ngoại	X
2	IDROLE	Id quyền	bigint	Khóa ngoại	X

- Thực thể POST: Thông tin bài viết

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	ID	Id bài viết	bigint	Khóa chính	X
2	IDUSER	Id người đăng	bigint	Khóa ngoại	X
3	TIEUDE	Tiêu đề	varchar(80)		X
4	NOIDUNG	Nội dung	text		X
5	NGAYTAO	Ngày tạo	datetime		X

- Thực thể TAG: Thông tin tag

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	ID	Id	bigint	Khóa chính	X
2	TEN	Tên tag	varchar(255)		X

- Thực thể MESSAGE: Thông tin tin nhắn

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	ID	Id tin nhắn	bigint	Khóa chính	X
2	IDNGUOIGUI	Id người gửi	bigint	Khóa ngoại	X
3	IDNGUOINHAN	Id người nhận	bigint	Khóa ngoại	X
4	NOIDUNG	Nội dung	text		X
5	NGAYTAO	Ngày tạo	datetime		X

- Thực thể POST_TAG: Tag của bài viết

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	IDPOST	Id bài viết	bigint	Khóa ngoại	X
2	IDTAG	Id tag	bigint	Khóa ngoại	X

- Thực thể POST_LIKE: Like của bài viết

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	IDPOST	Id bài viết	bigint	Khóa ngoại	X
2	IDUSER	Id người like	bigint	Khóa ngoại	X

- Thực thể POST_DISLIKE: Dislike của bài viết

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	IDPOST	Id bài viết	bigint	Khóa ngoại	X
2	IDUSER	Id người dislike	bigint	Khóa ngoại	X

- Thực thể COMMENT: Thông tin bình luận

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	ID	Id bình luận	bigint	Khóa chính	X
2	IDUSER	Id người đăng	bigint	Khóa ngoại	X
3	IDCHA	Id bình luận cha	bigint	Khóa ngoại	X
4	NOIDUNG	Nội dung	text		X
5	NGAYTAO	Ngày tạo	datetime		X

- Thực thể COMMENT_LIKE: Like của bình luận

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	IDCOMMENT	Id bình luận	bigint	Khóa ngoại	X
2	IDUSER	Id người like	bigint	Khóa ngoại	X

- Thực thể COMMENT_DISLIKE: Dislike của bình luận

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	IDCOMMENT	Id bình luận	bigint	Khóa ngoại	X
2	IDUSER	Id người dislike	bigint	Khóa ngoại	X

- Thực thể FRIEND_REQUEST: Lời mời kết bạn

Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	IDNGUOIGUI	Id người gửi	bigint	Khóa ngoại	X
2	IDNGUOINHAN	Id người nhận	bigint	Khóa ngoại	X

- Thực thể USER_FRIEND: Bạn bè

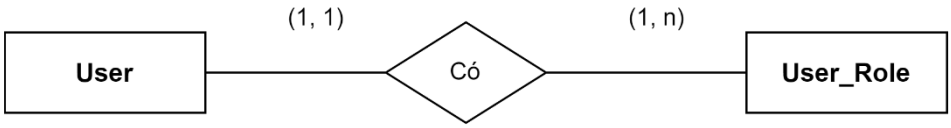
Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	IDUSER1	Id người dung	bigint	Khóa ngoại	X
2	IDFIEND	Id bạn bè	bigint	Khóa ngoại	X

- Thực thể USER_FOLLOWING: Theo dõi

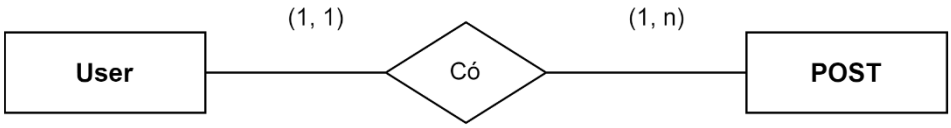
Stt	Thuộc tính	Mô tả	Kiểu dữ liệu	Khóa	Notnull
1	IDUSER	Id người dung	bigint	Khóa ngoại	X
2	IDUSERFOLLOW	Id người người follow	bigint	Khóa ngoại	X

4.3.4 Mô hình dữ liệu

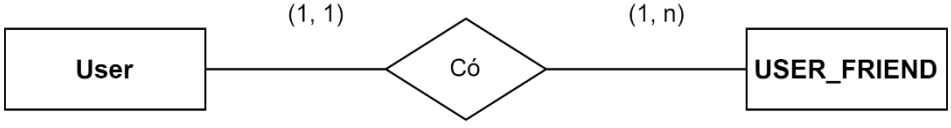
4.3.4.1 Xét quan hệ USER-USER_ROLE



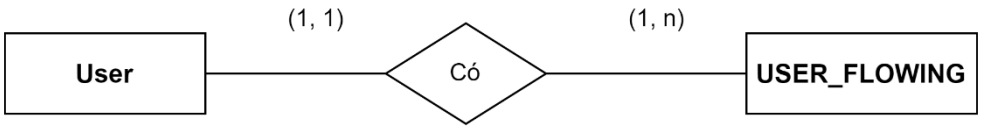
4.3.4.2 Xét quan hệ USER-POST



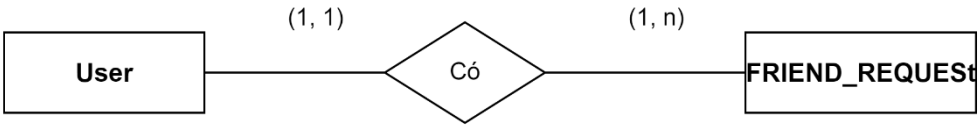
4.3.4.3 Xét quan hệ USER-USER_FRIEND



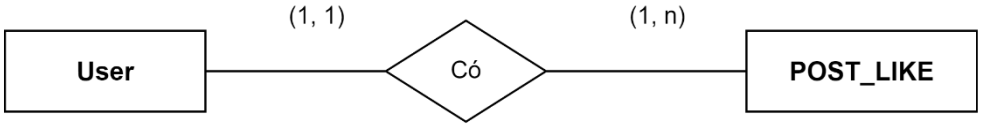
4.3.4.4 Xét quan hệ USER-USER_FOLLOWING



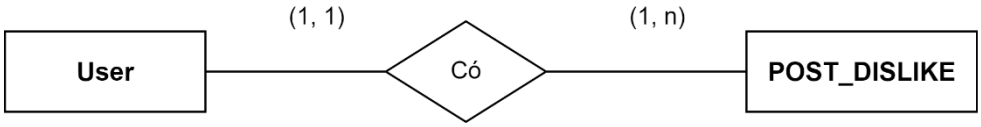
4.3.4.5 Xét quan hệ USER-FRIEND_REQUEST



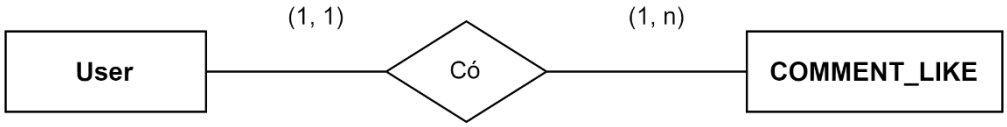
4.3.4.6 Xét quan hệ USER-POST_LIKE



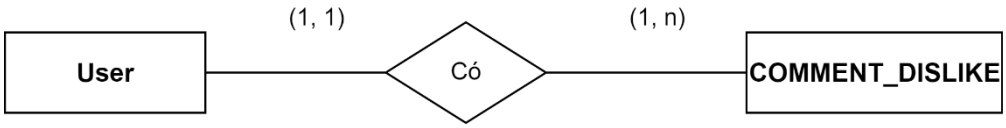
4.3.4.7 Xét quan hệ USER-POST_DISLIKE



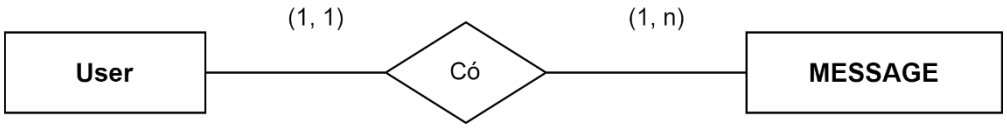
4.3.4.8 Xét quan hệ USER-COMMENT_LIKE



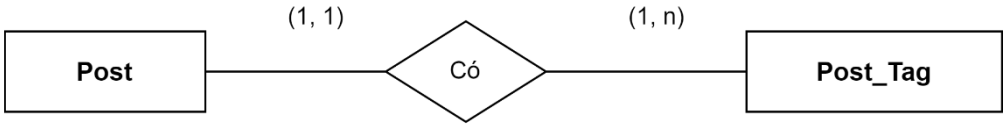
4.3.4.9 Xét quan hệ USER-COMMENT_DISLIKE



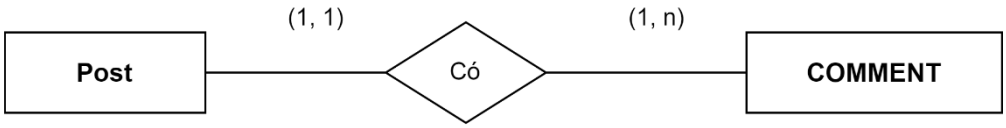
4.3.4.10 Xét quan hệ USER-MESSAGE



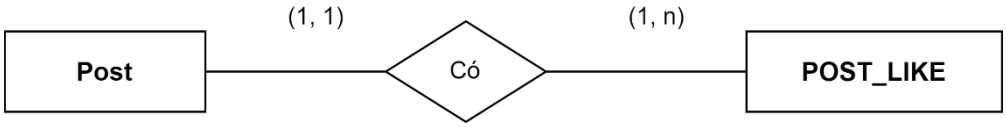
4.3.4.11 Xét quan hệ POST-POST_TAG



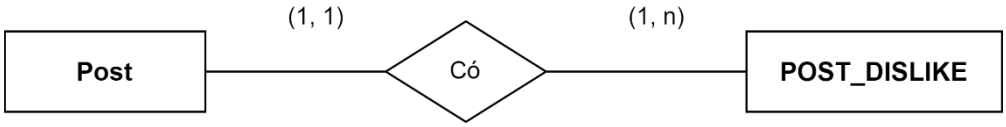
4.3.4.12 Xét quan hệ POST-COMMENT



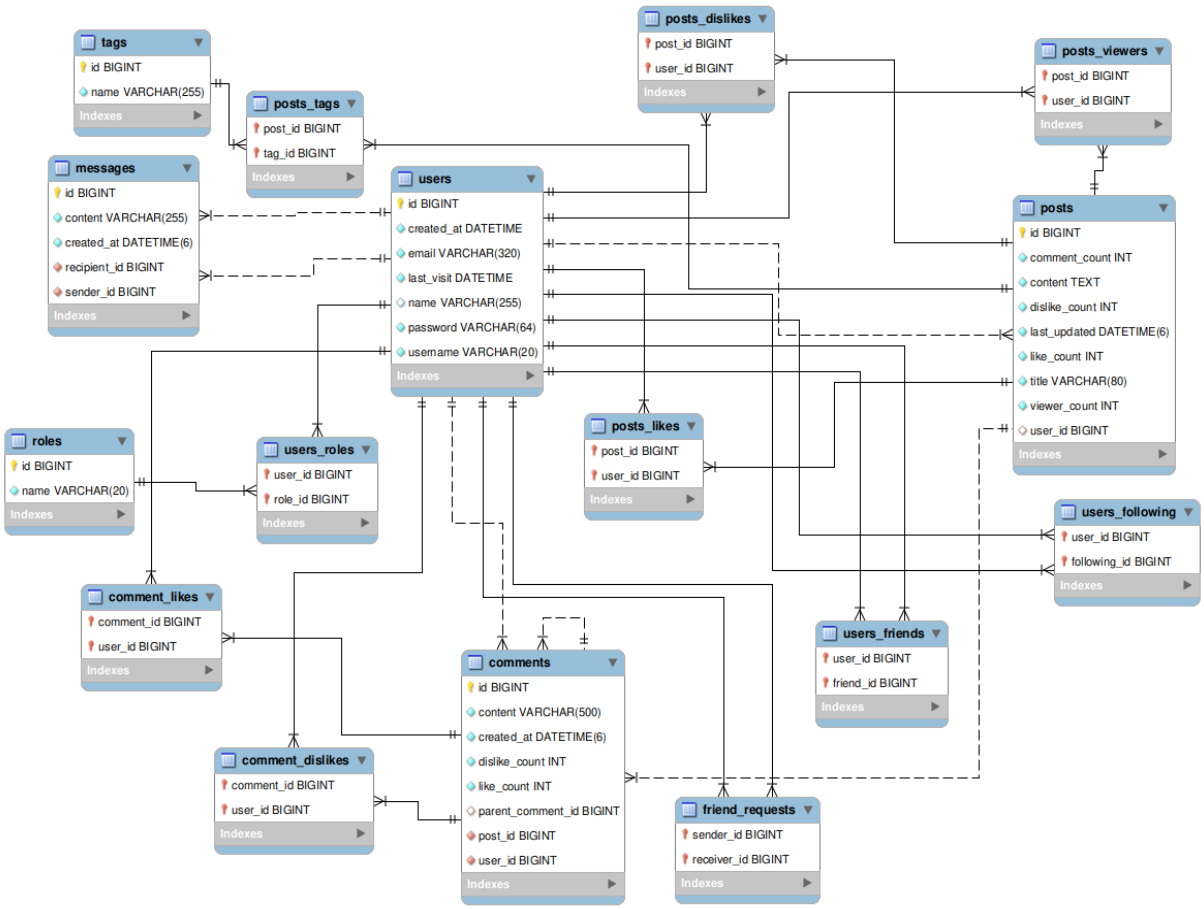
4.3.4.13 Xét quan hệ POST-POST_LIKE



4.3.4.14 Xét quan hệ POST- POST_DISLIKE



4.3.5 Phân tích và hoàn thiện mô hình quan hệ



TÀI LIỆU THAM KHẢO

[1] “Spring,” [Online]. Available: https://spring.io/guides/gs/spring-boot. [Accessed 21 08 2024].

[2] Hach, "Spring Security 6 with JWT, Oauth2," Viblo, [Online]. Available: https://hocvienptit.edu.vn/tong-quan-ve-spring-security-va-cach-thuc-bao-mat-trong-ung-dung-java. [Accessed 20 7 2024].

[3] N. H. Nam, "Hướng dẫn Spring Boot JPA + MySQL," Viblo, [Online]. Available: https://viblo.asia/p/spring-boot-11-huong-dan-spring-boot-jpa-mysql-GrLZD8dgZk0. [Accessed 21 8 2024].

[4] “SocketIO,” [Online]. Available: https://socket.io/docs/v4/tutorial/introduction. [Accessed 21 08 2024].