우수인

참고자료

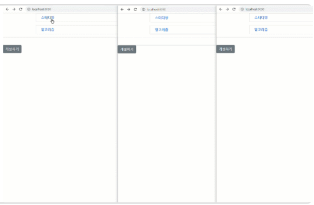
[Spring] webSocket으로 채팅서버 구현하기 (1)

서론 새로 개발할 앱을 플러터 + 스프링부트로 개발하기로 마음먹고나서, 이번 앱에 가장 중요한 부분이 채팅기능이기 때문에 가장먼저 스프링부트로 채팅서버를 구현해보려고한다.

https://learnote-dev.com/java/Spring-게시판-API-만들기-webSocket으로-채팅서버-구현하기/

Spring Boot Web Chatting : 스프링 부트로 실시간 채팅 만들기(1) stomp, socketjs, websocket

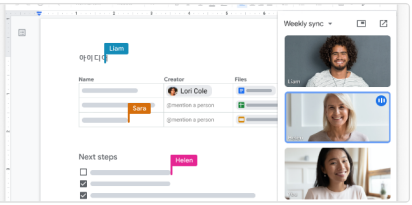
각 브렌치 및 포스팅 별 내용 - 일반 채팅 : master => 2 ~ 더보기 2022.09.01 - [토이 프로젝트/Spring&Java 갖고놀기] - Spring Boot Web Chatting : 스프링 부트로 실시간 채팅 만들기(2) chatDTO, DAO, Socket.js 코드 알아보기 - 시그널링 서버를 구현한 화상채팅 & 화면 공유(P2P) : master-webrtc-jpa => 6~7 번 포스팅 더



https://terianp.tistory.com/146

[Web] 웹소켓(WebSocket)이란? 웹소켓과 HTTP의 차이

웹소켓(WebSocket) 프로토콜이란? 웹소켓(WebSocket)은 클라이언트와 서버(브라우저와 서버)를 연결하고 실시간으로 통신이 가능하도록 하는 첨단 통신 프로토콜이다. 웹소켓은 하나의 TCP 접속에 전이중(duplex) 통신 채널을 제공한다. 쉽게 말해, 웹소켓은 Socket Connection을 유지한



https://code-lab1.tistory.com/300

정리

1. Session 이란?

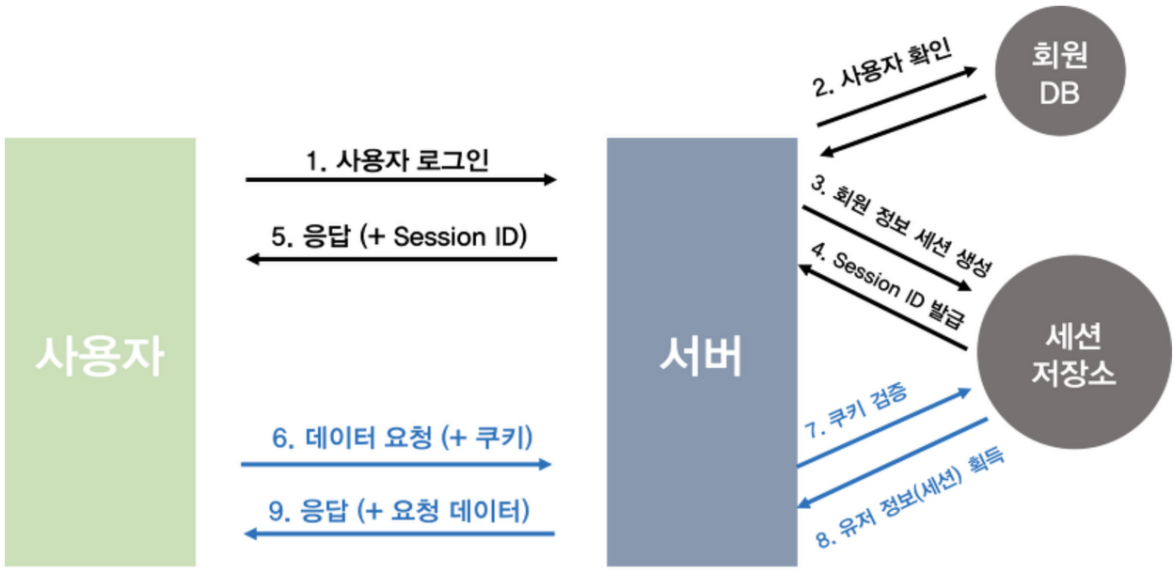
> **클라이언트**와 **서버** 간의 연결 상태를 의미하는 것이다.
>
> 좀 더 구체적으로 설명하면 클라이언트가 브라우저에 접속하여 서버와 **접속이 종료하기 전의 상태**를 의미한다.
>
> 쉽게 설명하면 사용자가 **웹사이트에 접속해 해당 창을 닫기 전 까지의 상태**라고 보면 된다.

→ **클라이언트가 서버에 접속하면 세션ID를 할당 받음.**

2. Session 활용 방법

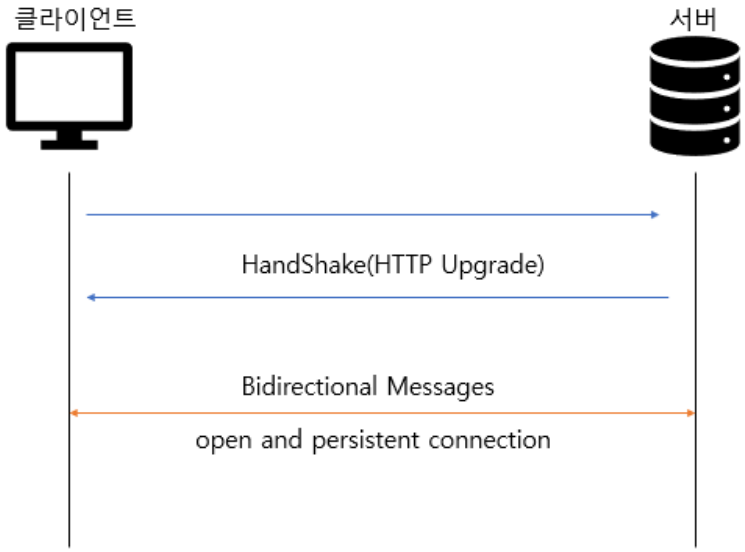
1. **사용자 인증 및 세션 식별**: 사용자가 화상 채팅 또는 채팅 서버에 접속할 때, 사용자를 인증하고 세션을 생성하여 식별자로 사용 가능
2. **상태 관리**: 이를 통해 화상 채팅에서는 사용자의 온라인/오프라인 상태, 마이크/카메라 활성화 여부 등을 추적할 수 있고, 채팅 서버에서는 각 사용자의 채팅방 참여 상태 등을 관리할 수 있음
3. **메시지 교환**: 세션은 사용자 간에 메시지를 교환하는 데 사용 가능
4. **타임아웃 및 세션 종료**: 일정 시간 동안 사용자의 비활동이 감지되면 세션을 종료할 수 있음.

5. **보안 및 권한 부여**: 세션을 사용하여 사용자의 보안 및 권한 부여를 관리 가능



3. WebSocket 이란?

> **웹소켓(WebSocket)**은 클라이언트와 서버(브라우저와 서버)를 연결하고 **실시간으로 통신이 가능**하도록 하는 첨단 통신 프로토콜이다. 웹소켓은 **하나의 TCP 접속에 전이중(duplex) 통신 채널을 제공**한다.
>
> 쉽게 말해, 웹소켓은 Socket Connection을 유지한 채로 **실시간으로 양방향 통신 혹은 데이터 전송이 가능한 프로토콜**이다.



4. 채팅 구현 로직(SpringBoot와 WebSocket 사용)

1. **클라이언트**:
   - 클라이언트는 WebSocket을 사용하여 서버와 연결합니다.
   - WebSocket 클라이언트 라이브러리를 사용하여 서버와의 연결을 설정하고, 채팅 메시지를 전송하고 수신하는 로직을 구현합니다.
   - 연결된 클라이언트는 서버에 사용자 정보를 전달하여 식별될 수 있도록 합니다.
2. **서버**:
   - Spring Boot에서 WebSocket을 사용하기 위해 `@EnableWebSocket` 어노테이션을 사용하여 WebSocket 활성화합니다.
   - 클라이언트와의 연결을 관리하기 위해 WebSocket 핸들러를 구현합니다.
   - 클라이언트가 새로운 채팅 메시지를 전송하면, 해당 메시지를 상대방 클라이언트로 브로드캐스트합니다.
   - 서버는 연결된 클라이언트 목록을 유지하고, 메시지를 수신한 클라이언트를 제외한 다른 클라이언트에게 메시지를 전송하는 로직을 구현합니다.
3. **메시지 전송**:
   - 클라이언트는 WebSocket 연결을 통해 서버에 채팅 메시지를 전송합니다.
   - 클라이언트는 WebSocket 클라이언트 라이브러리를 사용하여 메시지를 서버로 보내는 요청을 생성하고 전송합니다.
4. **메시지 수신**:
   - 서버는 WebSocket 핸들러를 통해 클라이언트로부터 채팅 메시지를 수신합니다.
   - 서버는 메시지를 브로드캐스트하거나, 필요한 경우 수신한 메시지를 데이터베이스에 저장하는 등의 로직을 수행합니다.
5. **개인 식별**:
   - 클라이언트는 서버에 사용자 정보를 전달하여 개인을 식별할 수 있도록 합니다.

- 서버는 개별 클라이언트를 식별하여 해당 클라이언트에게 메시지를 전송하거나 개인 설정을 관리합니다.

5. SpringBoot + MySQL + Redis

1. **레디스 설정**:
   - `spring-boot-starter-data-redis` 라이브러리를 추가하여 Spring Boot와 레디스를 통합합니다.
   - `application.properties` 파일에서 레디스 호스트, 포트, 인증 정보 등을 구성합니다.
2. **WebSocket 핸들러**:
   - WebSocket 핸들러를 구현하여 클라이언트 간의 실시간 메시지 전송을 처리합니다.
   - 메시지를 수신하면 레디스에 해당 메시지를 저장하고, 받은 클라이언트와 상대방 클라이언트에게 메시지를 브로드캐스트합니다.
3. **대화 기록 저장**:
   - 채팅 메시지를 레디스에 저장하는 동시에, MySQL 등의 관계형 데이터베이스에도 대화 기록을 저장합니다.
   - 메시지는 사용자 식별자, 메시지 내용, 타임스탬프 등의 정보와 함께 저장될 수 있습니다.
4. **대화 기록 조회**:
   - 대화 기록을 조회할 때는 필요에 따라 레디스 또는 MySQL에서 조회할 수 있습니다.
   - 레디스를 사용하여 최근 대화 기록을 캐싱하고, 필요한 경우 MySQL 등의 데이터베이스에서 과거 대화 기록을 조회할 수 있습니다.

6. MySQL + Redis를 통한 채팅 로그 관리

1. **레디스를 사용한 최근 목록**:
   - 레디스의 Sorted Set(정렬된 집합)을 활용하여 최근 목록을 저장합니다.
   - 메시지를 수신할 때마다 해당 메시지를 레디스 Sorted Set에 추가합니다. 정렬 기준은 타임스탬프 또는 메시지의 순서 등을 사용할 수 있습니다.
   - 최근 목록을 조회할 때는 Sorted Set에서 가장 최근 메시지부터 일정 개수를 가져옵니다.
2. **MySQL을 사용한 과거 대화 기록**:
   - MySQL 등의 관계형 데이터베이스에 과거 대화 기록을 저장합니다.
   - 대화 기록을 데이터베이스에 저장할 때는 레디스에 저장하는 것과 함께 데이터베이스에도 저장합니다.
   - 과거 대화 기록을 조회할 때는 데이터베이스에서 필요한 범위의 대화 기록을 쿼리하여 가져옵니다. 쿼리 조건으로는 사용자 ID, 대화방 ID, 타임스탬프 범위 등을 사용할 수 있습니다.

7. 내가 생각한 최종 통신 방법

1. SpringBoot에서 WebSocket 활성화
2. 클라이언트가 WebSocket에 연결 요청
3. SpringBoot Main Server는 WebSocket에 연결을 수락하고 클라이언트와 연결 설정
4. 클라이언트는 WebSocket을 통해 다른 연결된 클라이언트와 메세지를 주고받음
5. 메세지가 종료되었을 때, Redis와 MySQL에 대화 내용을 저장함

6. 다음 번에 같은 2명의 클라이언트가 대화할 때 일정 기간의 대화 목록을 Redis에서 받아오고, 더 이전 대화를 보려고 한다면 MySQL에서 지난 대화를 가져와서 프론트에 보여줌