STOMP 프로토콜 기반 신뢰적 메신저 App 서비스 개발 및 구현

권현수 이효동 한현민 오정석 김창묵 *정설영
경북대학교 컴퓨터학부
nowwater0225@gmail.com good79797979@naver.com um4825@naver.com
uh04049@naver.com asde130@naver.com *snowflower@knu.ac.kr

Developing and implementing STOMP protocol-based reliable messenger app services

Hyeon-Su Gwon Hyo-Dong Lee Hyun-Min Han Jeong-Seok Oh Chang-Muk Kim
*Seol-Yeong Jeong
Kyungpook National University

요약

현재 세계 모바일 App 시장에서는 메신저 서비스가 꾸준히 인기를 이어가고 있으며 국내 모바일 메신저 시장 또한 월간이용자 수가 꾸준히 유지되고 있다. 이에 따라 많은 App들이 메신저 기능을 제공하기 위해 자체 채팅 서비스를 도입하거나 외부 메신저 App을 연동하는 방법을 사용하고 있다. 하지만 중·소 IT 기업의 경우 자체 개발 인력 및 인프라를 마련하는데 한계가 있으며 외부 메신저 App 연동 시 이탈 역효과의 단점이 발생한다. 이에 본 논문은 중·소 IT 기업에 활용성을 높일 수 있는 STOMP 프로토콜 활용 자체 메신저 App 서비스를 개발하려 한다. 본 논문을 통해 App 자체에서 메신저 기능을 손쉽게 도입해 이용자 이탈율 최소화를 기대한다.

1. 서론

모바일 App 시장의 경우, 일시적인 변화와 유행에 민감하게 반응하며 다양한 App들의 다운로드 순위는 상승, 하강을 반복한다. 하지만 모바일 메신저 App 서비스의 경우 App 시장의 활성화 이후로 꾸준한 인기를 유지하고 있다. 세계 모바일 App 시장에서는 'WhatsApp', 'Facebook Messenger'와 같은 메신저 App 서비스들이 꾸준한 다운로드 순위 상위권을 차지하고 있으며, 한국 모바일 메신저 서비스 'Kakao Talk'의 경우 2020년 1분기 월간활성이용자수(MAU)가 4,500만 명에 도달하는 기록을 보였다[1].

모바일 메신저 서비스 시장의 성장에 따라, 많은 App 서비스들은 App 자체에서 메신저 채팅 기능을 도입하려는 움직임을 보인다. 최근 O2O(Online to Offline)[2] 서비스의 인기에 따라 온라인상에서 이용자와 서비스 제공자가 소통할 수 있는 공간이 필요해졌으며 자체 채팅 기능을 도입하는 사례가 늘고 있다. 하지만 중·소 IT 기업의 경우 자체 메신저 기능을 개발하기에는 인력과 인프라가 부족한 문제를 가지고 있다. 또한, 메신저 기능을 필요로 하는 기업들에 전문적으로 메세징 솔루션을 제공하는 플랫폼 기업의 경우 높은 비용 때문에 중·소기업에는 적합하지 않은 해결방안이다. 최근 이용자 수를 높게 유지하는 외부 메신저 App 계정을 연동해 메신저 기능을 대신하는 경우가 늘고 있지만, 이 경우 사용자 이탈 역효과를 유발한다. 실제로 O2O 서비스에서 사업자와 소비자 간의 연결을 위해 외부 메신저 App을 이용하게 되면 사용자 잔존효과가 떨어지는 현상을 겪고 있다.

이에 본 논문은 중·소 IT 기업에서 자체적으로 활용할 수 있는 STOMP 프로토콜 기반 메신저 App 서비스를 설계하고 구현한다. 자체 메신저 서버를 구축 후 STOMP 프로토콜을 활용하여 싱글. 멀티 채팅을 구현하고 신뢰적인 채팅 서비스를 개발해 패킷 로스 최소화를 기대한다.

2. 관련 연구

2.1. WebSocket

본 과제에서 메시지 전송처리를 위해서는 클라이언트와 서버 간의 통신이 필요하다. 서버와 클라이언트의 통신을 위한 HTTP 프로토콜 중 하나인 Polling은 새로운 정보가 있는지 확인하기 위해 주기적으로 HTTP 요청을 보낸다. 하지만 이러한 방식은 지속해서 요청을 보내기 때문에 매번 커넥션을 위한 Handshake 비용이 발생해서 본 과제에 적합하지 않다고 판단했다. 따라서 본 과제에서는 메시지를 처리하는 방식을 WebSocket을 이용하여 구현하였다.

WebSocket[3]은 HTTP 요청의 방식 중 하나이지만, Polling과는 다른 방식으로 동작한다. WebSocket은 Polling과 달리 하나의 요청에 커넥션을 끊지 않고 유지할 수 있으며, 클라이언트 - 서버 간의 양방향 통신이 가능하다. 앞에서 언급한 반복적인 HandShake 과정에서 발생하는 비용을 줄일 수 있다. 하지만 WebSocket은 HTTP와 다르게 메시지 내용에 의미를 두지 않기 때문에 클라이언트-서버 간에 임의로 메시지에 의미를 부여하지 않으면 로직을 처리하기가 힘들다. 그래서 이러한 문제를 해결하기 위해 STOMP 프로토콜을 상위 프로토콜을 이용하였다.

2.2. STOMP

STOMP 프로토콜은 WebSocket 위에서 동작하는 프로토콜로서, 클라이언트와 서버가 전송할 메시지 유형, 형식, 내용을 정의하는 메커니즘이다. STOMP는 Simple Text Oriented Messaging Protocol의 약자로 TCP나 일반적인 WebSocket과 같은 양방향 네트워크 프로토콜 기반으로 동작한다. 하지만 이름에도 알 수 있듯이, STOMP는 텍스트 지향 프로토콜이지만 Message Payload에 Text뿐만 아니라 Binary 데이터를 포함할 수 있다. STOMP 프로토콜은 Publish-Subscribe 기반으로 동작하는데, 즉 이는 Message Broker를 통해서 다른 사용자들에게 메시지를 보내거나 서버가 특정 작업을 수행하도록 메시지를 보낼 수 있게 할 수 있다. 이러한 특징으로 인해서 메시지를 처리하는데 특별한 프로토콜을 정의할 필요도 없으며, 메시지 형식을 커스터 마이징 할 필요가 없이 메시지 송수신 기능을 구현할 수 있었다.

3. 시스템 설계

3.1 시스템 아키텍처 및 개발환경

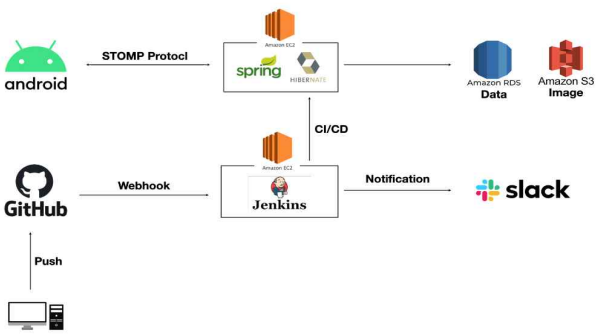

그림 1. 시스템 아키텍처

서버(스프링)의 운영 환경은 Amazon EC2 서비스 위에서 동작한다. 클라이언트(안드로이드) 는 Stomp Protocol을 이용해 서버와 메시지를 송/수신한다. 서버는 메시지 중 필요한 정보를 RBDORM의 구현체인 Hibernate를 통해 Amazon RDS에 Message, User, Room 등의 정보로 저장한다. 또한, 서버는 이미지 파일 정보를 Amazon S3에 저장한다.

CI/CD 환경 구축툴인 Jenkins는 또 하나의 AWS EC2 서비스 위에서 동작한다. Github의 Master 브랜치에 push 또는 merge 액션이 일어나면 webhook을 통해 Jenkins가 CI/CD 요청을 받게 된다. Jenkins는 서버가 운영 중인 EC2에 Jar 파일로 배포하고 그 결과를 Slack에 알린다.

3.2 시퀀스 다이어그램

본 연구에서는 채팅 서비스에 사용되는 시나리오와 객체 간 상호작용 과정을 다음과 같이 정의하였다.

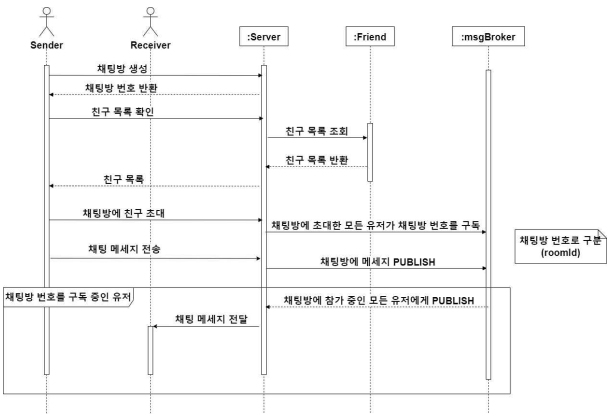

그림 2. Sequence Diagram

먼저 메시지 Sender가 서버에게 채팅방 생성을 요청하면 서버는 생성된 채팅방의 번호를 반환한다. 이후 Sender는 친구 목록을 조회하여 친구 상태인 유저들을 채팅방에 초대할 수 있다. 초대받은 유저는 자동으로 채팅방 번호를 구독(Subscription)함으로써 채팅방에 참가한다. 이후 채팅방에 초대된 모든 유저는 해당 채팅방 번호를 주제(Topic)로 발행(Publication)되는 모든 메시지를 받아온다. 이러한 상태는 채팅방 번호를 구독하고 있는 동안 계속해서 유지된다. 만약 유저가 채팅방을 나간다면 구독하고 있던 채팅방 번호를 더 이상 구독하지 않고, 해당 채팅방 번호로 발행되는 메시지를 받아오지 않는다. 이러한 과정은 Publisher와 Subscriber의 중간에서 Message Broker가 Message filtering을 수행하여 이루어진다. Message filtering은 크게 두 가지 방식이 존재한다. 첫 번째는 Topic-based filtering 방식이고, 두 번째는 Content-based filtering 방식이다. Topic-based system에서는 발행인이 메시지의 주제(Topic)를 먼저 정의하고 나서 해당 주제로 메시지를 발행한다. 그러면 같은 주제를 구독하고 있는 모든 구독자는 해당 주제로 발행되는 메시지를 전송받을 수 있다. 반면 Content-based system에서는 구독자가 받아올 메시지에 대해 제약조건들을 정의해두고, 그 제약조건을 만족하는 메시지만 발행인으로부터 받아온다. 본 연구에서는 두 가지 Message filtering 방식 중 Topic-based filtering 방식을 선택하여 개발을 진행하였다. 이렇게 발행인과 구독자, 그리고

Message Broker를 서로 분리해 줌으로써 높은 Scalability를 끌어낼 수 있다.

3.3 데이터베이스

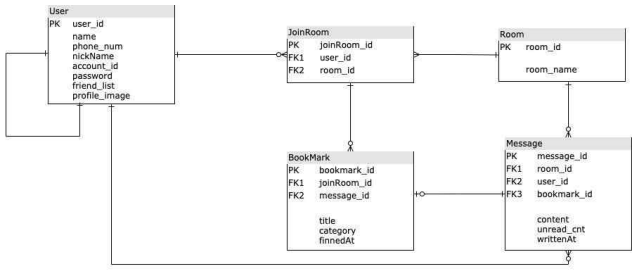

그림 3. 데이터베이스 구조

본 연구에서는 그림 3과 같이 데이터베이스 구조를 설계하였다. Message Broker를 통해서 메시지 통신을 수행하지만, 메시지와 관련된 부가적인 서비스를 제공하기 위해 데이터베이스에도 사용자 간에 주고받은 메시지를 저장한다. 먼저 User와 Room은 joinRoom을 통해 서로 연관 관계를 맺을 수 있다. joinRoom은 각각 하나의 User와 Room을 서로 연결해주며, Bookmark와 연관 관계를 맺는다. 이를 통해 사용자는 채팅방별로 특정 메시지에 대해 북 마크를 설정할 수 있고, 북 마크가 설정된 메시지들을 확인할 수 있다. 그리고 User는 다른 User와 친구 관계를 맺을 수 있으며, 자기 순환 참조를 통해 연관 관계를 맺도록 설계하였다.

3.4 정보구조도

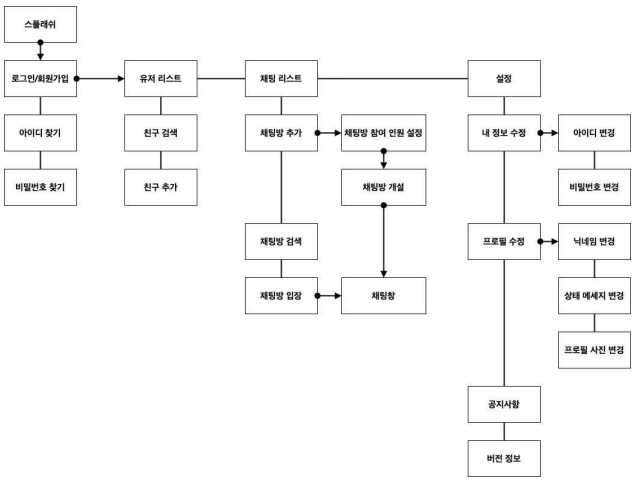

그림 4. Information Architecture

본 논문의 사용자 인터페이스 흐름을 알 수 있는 정보구조도이다. 스플래쉬 뷰를 지나 로그인 화면으로 이동한다. 로그인 과정을 완료하면 유저 리스트, 채팅 리스트, 설정 프래그먼트 화면을 볼 수 있는 홈 화면으로 이동한다. 유저 리스트 화면의 경우 나의 프로필과 친구 목록을 확인하고 친구 검색, 친구 추가 기능을 수행할 수 있다. 채팅 리스트 화면은 채팅방을 추가하고 채팅방 검색, 입장이 가능하다. 채팅 리스트에서 특정 채팅방을 입장하면 채팅창 액티비티가 시작되고 채팅창 뷰에서 채팅을 진행할 수 있다. 설정 화면의 경우 내 정보 수정과 프로필 수정, 공지사항, 버전 정보 등의 세부 항목들에 접근할 수 있다.

4. 시스템 구현 및 테스트

4.1 안드로이드의 정보구조

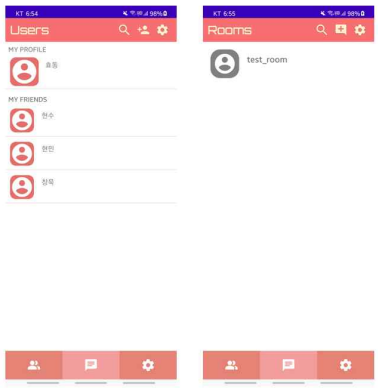

그림 5. 친구목록 및 채팅방 목록 UI

그림5는 친구목록과 채팅방 목록 기능을 구현한 UI이다. 친구목록 화면에서는 사용자의 친구들의 닉네임, 상태 메시지와 같은 공유정보를 리스트 형태로 확인할 수 있다. 채팅방 목록 화면 역시 자신이 참여하고 있는 채팅방들의 목록과 정보를 리스트 형태로 보여준다. 두 화면에 표시되는 데이터는 AWS RDS 서버 데이터베이스에 저장되어 있으며 정보가 수정될 경우 갱신된 정보가 화면에 표시된다.

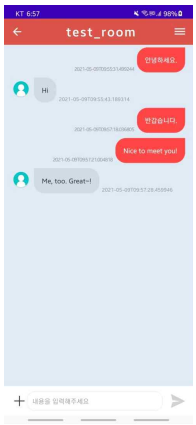

그림 6. 채팅방 내부 UI

그림6은 채팅방 내부 UI이다. 채팅방 내에서는 해당 채팅방을 구독하고 있는 사용자 모두에게 메시지를 전송할 수 있다. 전송된 메시지는 실시간으로 갱신되며 각 사용자의 화면에서 확인할 수 있다. 또한, 북 마크 기능을 이용하여 자신이 원하는 메시지가 있는 곳을 따로 저장 및 확인할 수 있다.

4.2 안드로이드의 Stomp Protocol 사용

토픽	받는 메시지 타입	설명
/message/{room ID}	Message	room의 새로운 메시지
/invite/room/{ID}	Room	방으로 초대
/invite/friend/{ID}	Friend	친구 초대

표 1. 구독 토픽별 설명

안드로이드는 Stomp Protocol을 통해 아래 표처럼 3종류의 토픽을 구독한다.

구독한 토픽에서 새로운 데이터를 수신하면 해당 타입으로 변환 후, 로컬 데이터베이스(SQLite)에 저장한다. 사용자가 오프라인 상태여서 받지 못한 정보는 온라인 상태로 바뀔 때, 로컬 데이터베이스에서 각 테이블의 엔티티 중 가장 늦게 삽입된 엔티티의 시간을 각각 가져온다. 테이블에 삽입할 새로운 데이터를 Stomp Protocol을 통해 마지막 시간 이후의 정보를 요청하여 해당 정보를 수신한다.

특정 방의 Message를 보여주는 MessageFragment에서는 로컬 데이터베이스에서 roomID가 일치하는 Message들을 가져와 View를 그리게 된다. LiveData를 통해 새로운 Message 정보가 로컬 데이터베이스에 추가되었는지 관찰한다. 새로운 Message가 추가되면 View를 새로 그리게 된다. 현재 참여 중인 방 목록을 보여주는 RoomFragment에서는 로컬 데이터베이스에서 자신의 ID 와 일치하는 방 목록들을 가져와 View를 그리게 된다. LiveData를 통해 새로운 Room 정보가 로컬 데이터베이스에 추가되었는지 관찰한다. 새로운 Room 정보가 추가되면 View를 새로 그리게 된다. 친구 목록을 보여주는 FriendFragment에서는 로컬 데이터베이스에서 자신의 ID 와 일치하는 친구 정보들을 가져와 View를 그리게 된다. LiveData를 통해 새로운 Friend 정보가 로컬 데이터베이스에 추가되었는지 관찰한다. 새로운 Friend 정보가 추가되면 View를 새로 그리게 된다.

메시지 전송, 친구 추가 요청, 방 생성 정보 및 친구 초대 기능은 Stomp Protocol을 사용하며 표 1. 의 토픽으로 전송하게 된다. Message 전송 버튼을 누르면 해당 메시지를 Message 타입으로 변환 후, /message/{roomID}로 전송한다. 친구 추가 버튼을 누르면 해당 정보를 Friend 타입으로 변환 후, /invite/friend/{friendID}로 전송한다. 해당 방으로 친구 초대는 해당 정보를 Room 타입으로 변환 후, /invite/room/{roomID}로 전송한다.

5. 결론

본 논문에서는 모바일 애플리케이션 환경에서의 신뢰성 있는 메시지 전송을 위하여 STOMP 프로토콜에 기반한 메신저 웹 애플리케이션을 개발 및 제안하고, 테스트베드 구현을 통해 이를 검증하였다. 다른 솔루션, 프로토콜에 기반한 메신저 앱 사용 시 일부 메시지가 누락되는 문제점을 본 논문에서 제안하는 방법을 통해 해결할 수 있을 것으로 기대한다. 또한, 중소기업이나 스타트업과 같은 기업에서 신뢰성 측면에서 경쟁력 있는 자체 메신저 애플리케이션을 개발할 수 있을 것으로 예상된다.

ACKNOWLEDGEMENT

"본 연구는 2021년 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학사업의 연구결과로 수행되었음"(2021-0-01082)

참 고 문 헌

[1] 전자신문, “진격의 카톡, 3개월 연속 4억 시간 넘었다”, https://m.etnews.com/20200522000186, 2020

[2] 김한준, “O2O(Online to Offline) 비즈니즈 모델의 서비스 동향 연구” 한국정보처리학회 춘계학술발표대회, 2018, pp.198-201

[3] 서준오, 김철원, “IOT환경에서 MQTT와 WebSocket을 활용한 실시간 사물제어 시스템 설계 및 구현” 한국전자통신학회 논문지, 2018, pp.517-524