⚖️

HW4. Final Term Project Work

1. 프로젝트 팀 명

- 팀명: 고진감래
- 프로젝트 설명: 텔레그램과 같은 종단간 암호화(E2EE) 방식을 이용하는 채팅 서비스

2. 팀원 역할 진행 사항

- 팀원

  노종빈, 20180891, 소프트웨어학부, 4학년

  안시현, 20180502, 사회학과, 4학년

  조동후, 20192248, 정보보안암호수학과, 3학년

- 프로젝트에서 팀원의 역할

  노종빈 : 팀장, 백엔드, db관리

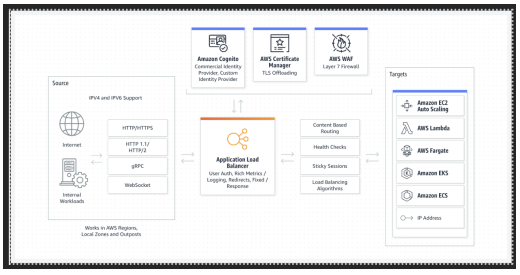
  안시현 : Docker를 활용하여 애플리케이션을 컨테이너화, AWS 다양한 서비스 이용하여 배포, 백엔드 및 프론트엔드 간의 효율적 소통 구축, 최종 보고서 총괄

  조동후 : 프론트엔드(App), 종단간 암호화(E2EE) 구현 프로세스 설계

3. 작업하였던 소프트웨어 시스템에 대한 목표

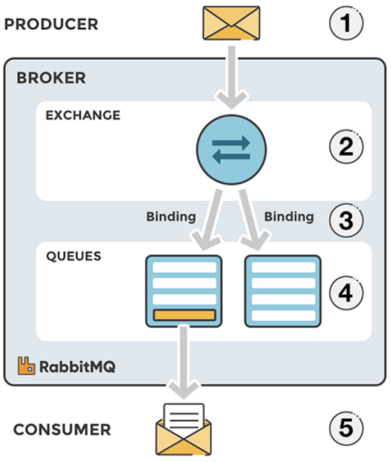
- 효율적 자원관리
  - vision - **Docker를 활용한 채팅 시스템 관리:** Docker를 사용하여 시스템을 쉽게 관리하고 확장하며 환경 간의 일관성을 유지
  - scope - Docker 이미지를 작성하고 필요한 의존성 및 설정을 포함한다.

- high - availability 시스템 만들기
  - vision - aws의 기능을 활용하여 시스템의 availability를 높임
  - scope
    - **Load-Balancer Architecture 구현 :** load balancer architecture구현 ⇒ 채팅기능의 availability 상승



- 채팅 - 개인 채팅 및 그룹 채팅 가능
  - vision - **실시간 커뮤니케이션**: message broken architecture를 활용한 채팅 시스템 구현한다.
  - scope
    - **실시간 채팅 기능**: 사용자는 실시간으로 채팅 메시지를 송수신한다.

- **메시지 브로커 구현**: RabbitMQ 메시지 브로커를 사용하여 메시지 전파 및 처리를 구현한다. 컨슈머와 메시지 브로커간의 결합도가 높지만 현 프로젝트 규모에 맞게 트래픽이 작기 때문에 선택 예정이다.



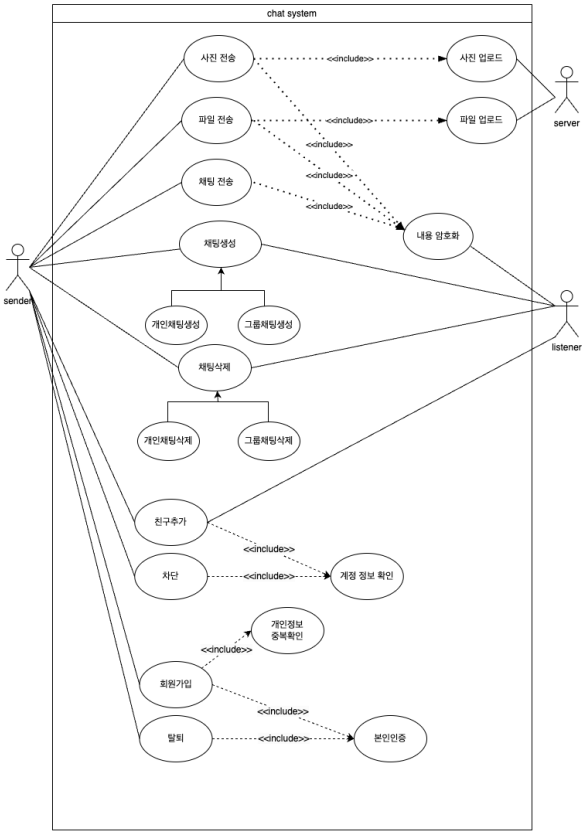
- **개인채팅/그룹채팅 구분:** 개인 채팅을 지원하기 위해 각 사용자는 고유한 식별자를 가진다. 채팅 메시지는 송신자와 수신자의 식별자를 사용하여 특정 대화 상대와 연결한다. 그룹채팅은 여러 사용자가 하나의 대화방에서 채팅하기 때문에 각 그룹 채팅은 고유한 식별자를 가진다.

- 종단간 암호화(E2EE)를 지원하는 개인 및 그룹 채팅 가능한 시스템 구현
  - vision - 제3자로부터 채팅 내용을 보호하기 위해 채팅 시스템에 암호화를 적용한다.
  - scope
    - 종단간 암호화(End to End Encryption): 기존 암호화 방식은 서버에서 복호화되기 때문에, 내용 탈취 우려가 있다. 이를 방지하기 위해 단말기(Client)에서 암/복호화가 이루어지는 종단간 암호화를 적용한다.
    - 하이브리드 암호 시스템: 대칭키 암호 시스템을 이용할 경우, 키 공유 등의 문제가 발생한다. 비대칭키 암호 시스템을 이용할 경우, 속도가 상대적으로 느리다는 문제가 발생한다. 따라서 대칭키와 비대칭키를 적절히 활용하여 장점을 극대화하는 하이브리드 암호 시스템을 사용하고자 한다.

- 단일 계층이 아닌 OSI layer의 여러 계층에서 보안 프로토콜을 이용하여 기밀성 보장 시스템 구현
  - vision - 중간자 공격 등 제 3자의 위협을 방지하기 위해 OSI layer의 여러 계층에 암호화를 적용한다.
  - scope
    - 4 Layer: TLS 프로토콜 이용
    - 7 Layer: HTTPS 프로토콜 이용

4. 시스템 개요 및 특성

- UseCase Diagram
  - 구현 부분: 채팅전송, 내용암호화, 채팅생성(개인,그룹), 채팅삭제(개인,그룹), 친구추가, 계정 정보 확인, 회원가입
  - 미구현 부분: 사진전송, 파일전송,탈퇴,차단,



useCase	하는역할(**Role and Responsibility**)	처리해야할 정보 (**Attributes**)	고려하는 **Functional Requirements**	고려할수있는**Architecture Style, GRASP,SOLID,DesignPatterns**	고려하는**Non-Functional Requirement**	**이를 반영하는 QAs**
회원가입	회원정보를 생성한다	아이디, 비밀번호, 개인 인증 수단(전화번호, email),	1. 회원가입시 정보들이 db에 저장이 되어야 한다. 2. 비밀번호 및 민감한 개인정보는 암호화가 되어야 한다. 3. 아이디 및 개인 고유 식별 정보(e-mail or 전화번호)가 중복 되면 안된다. 4. 개인 인증을 위해 전화번호, email 등으로 인증번호 발송 후 인증이 되어야 한다.	client - server architecture, multi-tiers architecture, observer pattern, high-cohesion	1. 도용된 개인정보인지 확인할 수 있어야 한다 2. 불필요한 개인정보는 최소화 한다 3. 개인정보 위탁에관한 법률을 준수해야한다.	보안(Security 신뢰성 (Reliability)
친구추가	다른 사람의 계정을 친구로 등록한다	친구로 등록하는 상대방의 id	1.친구관계에 대한 db에 정보가 들어가야 됨 2. 친구로 등록된 사람도 친구신청에 대한 정보를 받아봐야 한다 3. 친구로 등록되면 친구의 정보를 볼 수 있어야 한다. 4. 친구로 등록되	client - server architecture, Pub/sub model	1. 과도하게 친구등록을 하는 사람은 제한을 두어야 한다 2. 무차별적인 친구등록으로 타인의 개인정보를 보려는 경우를 막아야 한다	확장성 (Scalability), 용성(Availabi 보안(Security

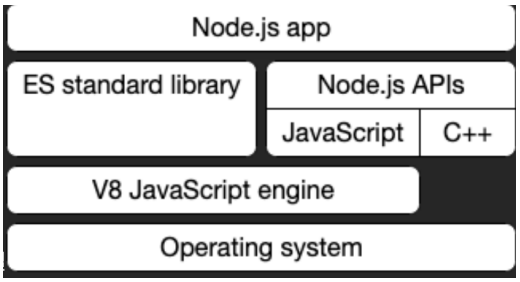
			면 채팅 초대 등의 기능을 사용가능 할수있게 되어야 한다			
차단	다른사람의 계정을 차단한다(친구관계를 해제한다)	차단하는 상대방의 id	1. 친구관계에 대한 db 수정 2. 차단당한 유저의 id가 차단한 유저의 정보에 접근 못하게 함 3. 차단db에 차단 정보 업로드	client - server architecture, Pub/sub model	1. 차단 당한 사람이 불법 행위를 하는 사람이라면 유저의 행위를 검토해야 한다 2. 차단 당한 유저가 차단한 유저에게 접근할 수 있는 방법이 없어야 한다	보안(Security
탈퇴	사용자의 계정을 탈퇴한다	탈퇴하는 사용자의 id	1. 탈퇴를 원하는 사용자의 개인정보 삭제 2.탈퇴하는 계정이 있는 채팅방의 처리 (단체톡방 : 나가기, 개인톡방 : 방폐쇄) 3. 탈퇴된계정과 연관된 정보 삭제 (친구관계 등)	client-server architecture, high-cohesion	1.탈퇴후 정보 보관기간을 얼마로 설정할 것인가(서비스 악용을 막기 위해) 2.탈퇴후 바로 즉시 재가입이 되게 할 것인가(악용을 막기위해)	보안(Security
개인채팅방 생성	개인채팅방을 만든다.	유저 id, 상대방 id	1. 사용자 인증: 개인채팅방 생성 위해 시스템 로그인 2. 채팅방 생성: 채팅방 이름지정, 대화 상대방 선택, 채팅방에 대화 기록 저장.	client - server architecture, Single Responsibility Principle	1. 사용자는 개인채팅방을 만들기 위해 보안을 위해 친구 추가가 되어있는 친구만 개인채팅할 수 있도록 해야 한다. 2. 빠른 채팅방 생성 가능해야 한다.	보안(Security 성능 (Performance 사용성(Usabi
그룹채팅방 생성	그룹채팅방을 만든다.	채팅방 id, 채팅방 소속 유저 id	1. 사용자 인증: 개인채팅방 생성 위해 시스템 로그인 2. 채팅방 생성: 채팅방 이름 지정, 대화 상대방 선택, 초대, 채팅방에 대화 기록 저장.	client - server architecture, Single Responsibility Principle	1. 다수의 사용자와 메시지를 관리해야 하기 때문에 스케일링 및 빠른 응답시간에 신경써야 한다.	확장성 (Scalability), 능(Performar
개인채팅방 삭제	개인채팅방을 삭제한다	유저 id, 상대방 id	1. 사용자 인증: 개인채팅방 생성 위해 시스템 로그인 2. 사용자에게 삭제 시 경고창 뜨게 하기 3. 채팅방 삭제: 채팅방에 대한 권한 취소	client - server architecture,	1. 삭제 요청은 적절한 권한 및 인증을 거쳐야 한다. 2. 삭제 요청에 대한 빠른 응답 시간이 필요 하다. 3. 삭제 시 권한 및 서버관리측면을 고려해 데이터를 남길지 삭제할지 고려해야된다.	보안(Security 능(Performar
그룹채팅방 삭제	그룹채팅방을 삭제한다	채팅방 id, 채팅방 소속 유저 id	1. 사용자 인증: 개인채팅방 생성 위해 시스템 로그인 2. 사용자에게 삭제 시 경고창 뜨게 하기 3. 채팅방 삭제: 채팅방에 대한 권한 취소, 그룹채팅방	client - server architecture,	1. 삭제 요청은 적절한 권한 및 인증을 거쳐야 한다. 2. 삭제 요청에 대한 빠른 응답 시간이 필요 하다.	보안(Security 성능 (Performance

			에 삭제 알림 자동 전송			
텍스트 전송	텍스트를 입력하여 전송한다.	채팅 id, 채팅방 id, 송신자 id, 보낸 시각, 채팅 내용	1. 텍스트 내용 및 채팅방 정보가 같이 저장되어 나중에 다시 채팅할 때 불러올 수 있어야 한다. 2. 텍스트 전송 전 단말기에서 암호화를 진행한다.	message broker architecture, observer pattern, mediator pattern	1. 암호화가 잘 되었는지 확인해야 한다. 2. 전송이 정상적으로 되었는지(누락되었는지 등)을 확인해야 한다. 3. 전송 완료 후 수신자가 채팅이 전송 되었음 확인할 수 있어야 한다. 4. 단말기 내 암호화 키 관련 정보가 안전하게 저장되어야 한다.	보안(Security 뢰성(Reliabili 무결성(Integr

5. 소프트웨어 시스템에 대한 기술적 부분 및 Framework

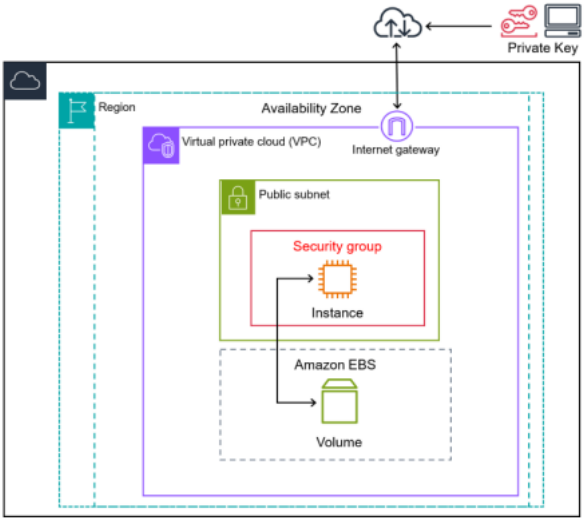
- 소프트웨어 시스템에 대한 개요 및 특성

1. Node.js



- **이벤트 기반:** Node.js는 이벤트 기반 아키텍처를 이용. 이는 비동기적인 이벤트 처리를 통해 효율적인 넌블로킹 I/O 작업을 가능케 함.
- **실행 엔진:** Node.js는 V8 JavaScript 엔진에서 실행됩니다. V8은 Google이 개발한 고성능 JavaScript 엔진으로, 빠른 코드 실행을 제공함.
- **WebSocket**: 채팅 서비스는 실시간 통신이 필수적인데, Node.js는 WebSocket과 같은 실시간 통신을 위한 기술을 지원. 실시간 업데이트를 쉽게 전송하고 수신할 수 있도록 도와주기 때문에 Node.js를 선택함.

2. AWS
    a. *<ec2 (Elastic Compute Cloud)>*

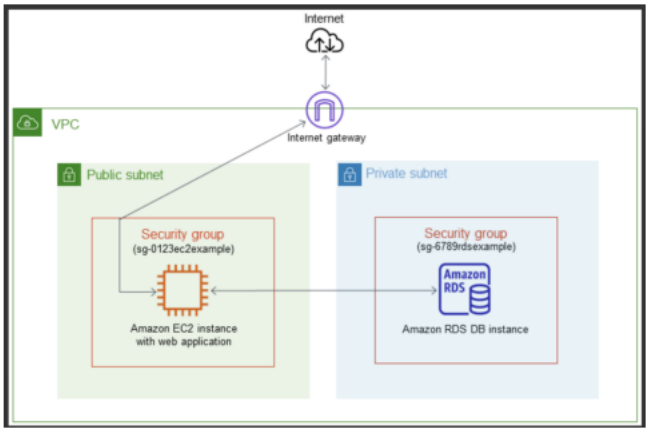


- **가상화 기술 활용:** 가상화를 통해 물리적 서버를 여러 개의 독립적인 가상 머신으로 분할함으로써 자원을 효율적으로 사용할 수 있음. 서버의 가상화를 통해 필요에 따라 가상 머신을 생성하고 크기를 조절할 수 있음.
- **유연한 스케일링:** 필요에 따라 가상 서버의 수를 동적으로 조절할 수 있어서 필요 이상으로 많은 물리적 서버를 구입할 필요가 없음.
- **쓰지 않는 자원의 해제:** EC2 인스턴스를 사용하여 필요한 만큼의 컴퓨팅 리소스를 구매하고, 필요 없어지면 인스턴스를 중지하거나 제거하여 비용을 절감할 수 있음.
- **EC2 인스턴스 생성 및 관리:** AWS EC2를 사용하면 필요에 따라 인스턴스를 생성하고 관리할 수 있어서 서비스의 확장성과 유연성이 향상
- EC2와 함께 다양한 AWS 서비스를 사용하여 서비스를 구성하고 최적화할 수 있는데 실제로 Amazon RDS를 사용하여 데이터베이스를 관리하고 Amazon S3를 사용하여 객체 스토리지를 활용할 수 있어서 소규모 팀프로젝트에 적합하다고 생각하였음.

b. *<ELB (Elastic Load Balancer)>*

- **다수의 서버에 대한 부하 분산:** Chat service에서 다수의 사용자 및 다양한 기능을 처리하는데 로드 밸런서를 사용하여 서버 간 부하를 균등하게 분산시킬 수 있음. 이는 대량의 동시 접속이나 다수의 채팅 메시지를 처리할 때 효과적임.
- **Health Check 활용:** 로드 밸런서의 Health Check를 통해 사용 가능한 서버만에 트래픽을 분산함으로써 안정성을 확보. 채팅 서비스의 각 서버가 정상적으로 작동하는지 주기적으로 확인 가능함.

c. *<RDS (Relational Database Service)>*



- **다양한 DB 엔진 선택:** Amazon RDS는 여러 가지 데이터베이스 엔진을 지원하며, Chat service의 요구 사항에 맞게 MySQL, PostgreSQL, 또는 Aurora와 같은 엔진을 선택하여 사용할 수 있음. 고진감래 팀은 MySQL 선택
- **자동 업데이트:** RDS는 데이터베이스 엔진에 대한 패치 및 업데이트를 자동으로 관리해줌. 이는 보안 문제나 성능 향상을 위한 업데이트를 자동으로 적용하여 채팅 서비스가 최신 및 안정된 환경에서 운영될 수 있도록 함
- RDS AZ(Availability Zone)

    - 데이터 보호를 위해서 db를 분산화 시킬 수 있음
    - 자동백업, 실시간 복제, 자동 장애 복구를 통하여 가용성과 내결함성을 높임

3. docker
    - **실행 환경의 용이성**: 도커 컨테이너를 이용하여 호스트 운영체제나 환경에 관계없이 일관된 실행 환경을 제공
    - **효율적인 자원환경**: 가상 머신에 비해 가볍고 빠르게 컨테이너를 생성하고 시작. 공통된 커널을 공유하므로 더 적은 자원을 소비

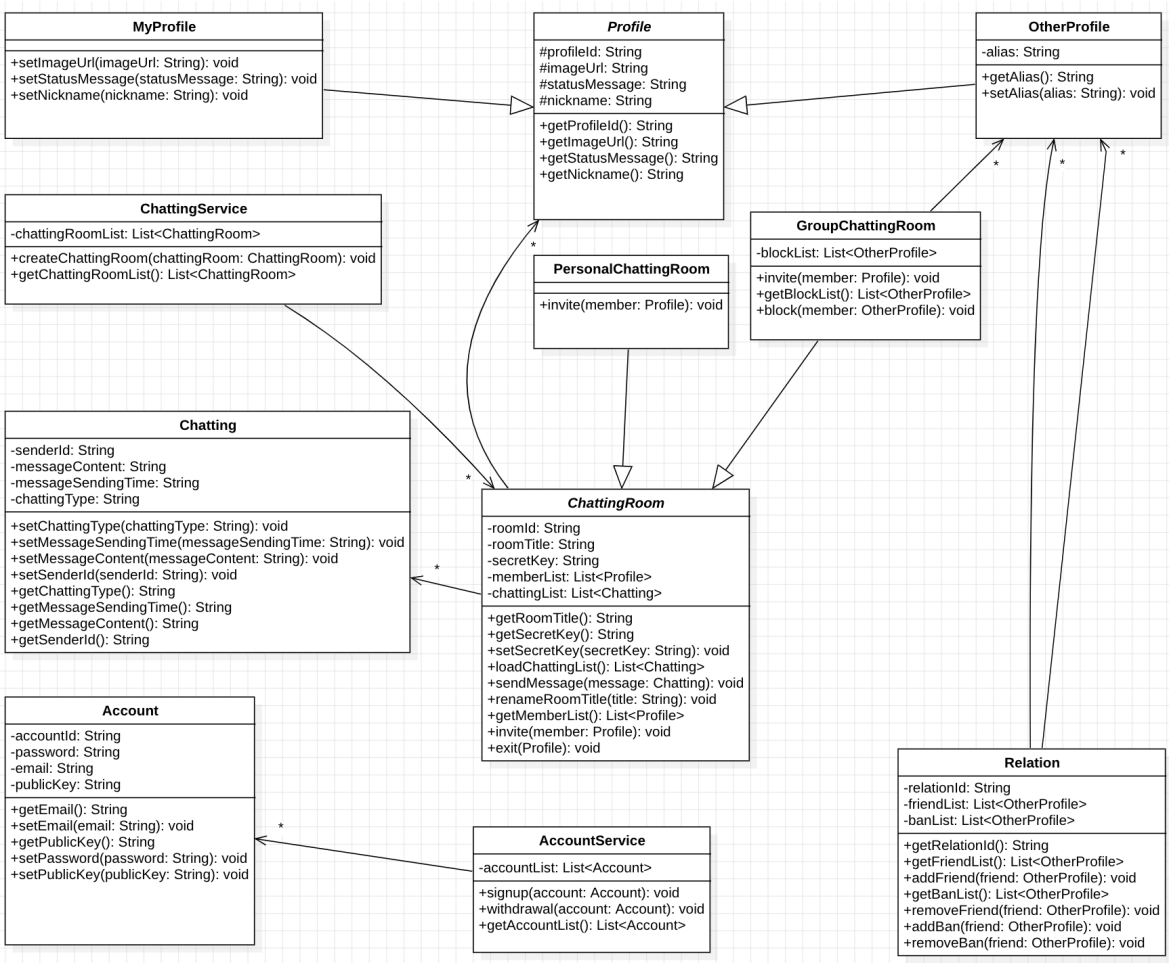
4. MongoDB
    - **문서 지향적 저장:** MongoDB의 문서 지향적인 구조는 채팅 메시지를 저장하기에 이상적임. 각 채팅 메시지를 MongoDB 콜렉션에 저장.
    - **스키마의 유연성:**MongoDB의 유연한 스키마는 다양한 구조의 메시지를 저장할 수 있도록 함. 예를 들어, 그룹 채팅 메시지는 개인 메시지와는 다른 속성을 가질 수 있음.
    - **빠른 검색을 위한 인덱스:** MongoDB의 인덱스를 사용하여 채팅 메시지를 빠르게 검색. 타임스탬프, 발신자, 또는 채팅방과 같은 필드에 인덱싱을 적용하여 쿼리 성능을 향상
5. Rabbit MQ
    - 오픈소스 메시지 브로커 소프트웨어
    - AMQP(Advanced Message Queing Protocol) 지원
    - 특징
        - 메시지 지향

        - AMQP 지원
        - 큐와 메시지 라우팅
        - 배달보장 (높은 신뢰선)
        - 유연한 라우팅
        - 클러스터링
        - 플러그인 시스템
        - 다양한 언어 및 프로토콜 지원
        - 액세스 제어 및 보안

- Software Architecture Style 을 가지고 있는 소프트웨어 시스템
    - Telegram
        - software architecture
            - client-server model
                - 클라이언트-서버 모델은 텔레그램의 아키텍처의 핵심이며, 여기서 클라이언트(사용자)는 텔레그램 서버와 상호 작용하여 메시지를 보내고 받습니다. 서버는 중간 매개체 역할을 하며, 클라이언트와 메시지를 중계하고 클라우드에 저장하여 전달될 때까지 보관합니다.
            - P2P model
                - 텔레그램은 또한 메시징 속도를 최적화하고 서버 부하를 감소시키기 위해 피어 투 피어 네트워크를 사용합니다. 두 클라이언트가 동일한 네트워크에 연결되어 있을 때 서버를 통하지 않고 직접 통신할 수 있습니다. 이로써 지연 시간이 감소하고 메시징 속도가 향상됩니다.\
        - Security
            - End - to -End Encryption
                - 텔레그램은 종단 간 암호화를 사용하여 메시지를 가로채지 못하도록 보호하고 의도한 수신자만이 그 내용을 읽을 수 있도록 합니다. 이는 메시지가 발신자의 기기에서 암호화되고 수신자의 기기에서만 해독되며, 어떠한 제3자(텔레그램 자체를 포함하여)도 메시지 내용에 액세스할 수 없음을 의미합니다.
    - Telegram의 어떠한 부분(모듈, 서버, 서비스 등)을 사용했는지에 대하여
        - client-server model을 이용하여 chat service 구현
        - 종단 간 암호화를 통해 메시지의 안전성 보장
        - HTTPS를 통한 통신은 데이터의 무결성과 기밀성을 강화하여 보안 수준을 높임

6. 시스템 분석 및 설계도

- 자신이 하는 작업의 FR 에 대한 Class Diagram



- NFR 에 대하여 고려하는 Quality Attributes 을 반영한 내용을 Class 에 설명 (양식 활용)

Class	하는역할(**Role and Responsibility**)	정보(**Attributes**)	해당하는 **Functional Requirements**	반영한 Architecture Style, GRASP, SOLID, Design Patterns, Tactics	반영한 **Non-Functional Requirement** 과 적용하는 **Quality Attributes**
Chatting	채팅 메세지 및 관련 정보 저장	송신자 ID, 메세지 내용, 메세지 송신 시각, 채팅 유형	채팅 서비스를 제공하기 위한 정보 제공	단일책임원칙(SRP), 데코레이터 패턴	종단간 암호화를 위해서 서버에 채팅 정보를 보내기 전에 암호화 해야한다. (security)
ChattingService	채팅방 관리	채팅방 목록	채팅 서비스를 이용하기 위한 공간 생성 및 관리	creator 패턴	채팅방 생성 시 방 별 특성이 다르므로 인원 수를 고려하여 채팅방을 생성해야 한다. (suitabillity)
ChattingRoom	채팅을 주고 받을 환경을 제공	채팅방 제목, 채팅방 비밀키, 채팅 참여자 목록, 채팅 내역	채팅방 초대 및 나가기, 채팅 전송 및 불러오기	controller 패턴	종단간 암호화를 위해서 각 채팅방은 고유한 비밀키를 가지고 있어야하며, 안전하게 공유되어야 한다. (security)
PersonalChattingRoom	개인채팅방 정보 담당	채팅방 제목, 채팅방 비밀키, 채팅 참여자 목록, 채팅 내역	채팅방의 주인을 설정. 채팅방의 상대방을 설정	observer 패턴: 메시지 전송 시, 관련된 객체들에 알림을 제공하여 상호작용을 지원.	채팅 내용은 안전하게 저장되어야 하며, 사용자 인증 등의 보안 기능을 제공한다.(security)

Class	하는역할(**Role and Responsibility**)	정보(**Attributes**)	해당하는 **Functional Requirements**	반영한 Architecture Style, GRASP, SOLID, Design Patterns, Tactics	반영한 **Non-Functional Requirement** 과 적용하는 **Quality Attributes**
GroupChattingRoom	그룹채팅방 정보 담당	채팅방 제목, 채팅방 비밀키, 채팅 참여자 목록, 채팅 내역, 채팅 차단 목록	그룹채팅방에 참여자를 추가. 그룹 채팅방의 참여자를 차단.	observer 패턴: 메시지 전송 시, 관련된 객체들에 알림을 제공하여 상호작용을 지원.	빠른 참여자 추가 및 삭제, 메시지 전송 및 수신 기능을 제공하여 사용자 간의 즉각적인 소통을 향상시킨다.(performance)
Profile	사용자의 소개 정보 저장	프로필 이미지 URL, 상태 메세지, 닉네임	사용자의 프로필 관련된 정보 제공	expert 패턴, 개방폐쇄원칙(OCP), 데코레이터 패턴	프로필은 사용자를 소개하는 책임을 가지나, 민감한 정보가 포함되지 않도록 한다. (security)
MyProfile	사용자 본인의 소개 정보 저장 및 서비스 제공	프로필 이미지 URL, 상태 메세지, 닉네임	프로필 이미지, 상태 메세지 및 닉네임 변경	개방폐쇄원칙(OCP), 데코레이터 패턴	사용자는 본인의 프로필을 변경할 수 있어야 한다. (changeability)
OtherProfile	사용자 본인 외 소개 정보 저장	프로필 이미지 URL, 상태 메세지, 닉네임, 별칭	프로필 별칭 조회 및 설정	개방폐쇄원칙(OCP), 데코레이터 패턴	사용자는 본인 외 프로필에 관하여 별칭을 부여할 수 있으며 설정한 본인만 별칭을 조회할 수 있어야 한다. (changeability)
Relation	사용자 간 관계 정보 서비스 제공	친구 목록, 차단 목록	친구 추가 및 삭제, 친구 차단 등록 및 해제, 친구 및 차단 목록 조회	expert 패턴	사용자는 편리하고 쾌적한 환경에서 채팅 서비스를 이용하기 위해 다른 사용자와의 관계를 설정할 수 있어야 한다. (usability)
Account	사용자의 자격증명(credential) 정보 저장	이메일, 비밀번호, 공개키	로그인 및 암호화를 위한 정보 제공		종단간 암호화를 위해 채팅방의 비밀키를 안전하게 공유하기 위해 각 사용자는 고유한 공개키를 가지고 있어야 한다. (security)
AccountService	사용자 계정 관리 서비스 제공	사용자 계정 목록	본 프로그램을 이용하기 위한 계정 추가, 삭제	expert 패턴	사용자 본인 외 계정에 접근할 수 없도록 권한을 고려해야 한다. 또한 사용자당 1개의 계정을 가지도록 제한해야 한다. (security)

- Architecture Driver (FR, QA 등) 에 대한 Tactics 을 반영 시 설명하기

  Tactic1. Deployability

  - **설명:** 앱의 모든 구성 요소를 독립적인 도커 컨테이너로 패키징하여 각각의 컨테이너가 독립적으로 실행되고 확장될 수 있도록 함.
  - **적용 방법:** 서버는 도커 이미지로 패키징되며, 각 컨테이너는 필요에 따라 스케일링될 수 있음. 컨테이너는 운영체제의 종속성을 포함하므로 환경에 상관없이 일관되게 동작할 수 있음.

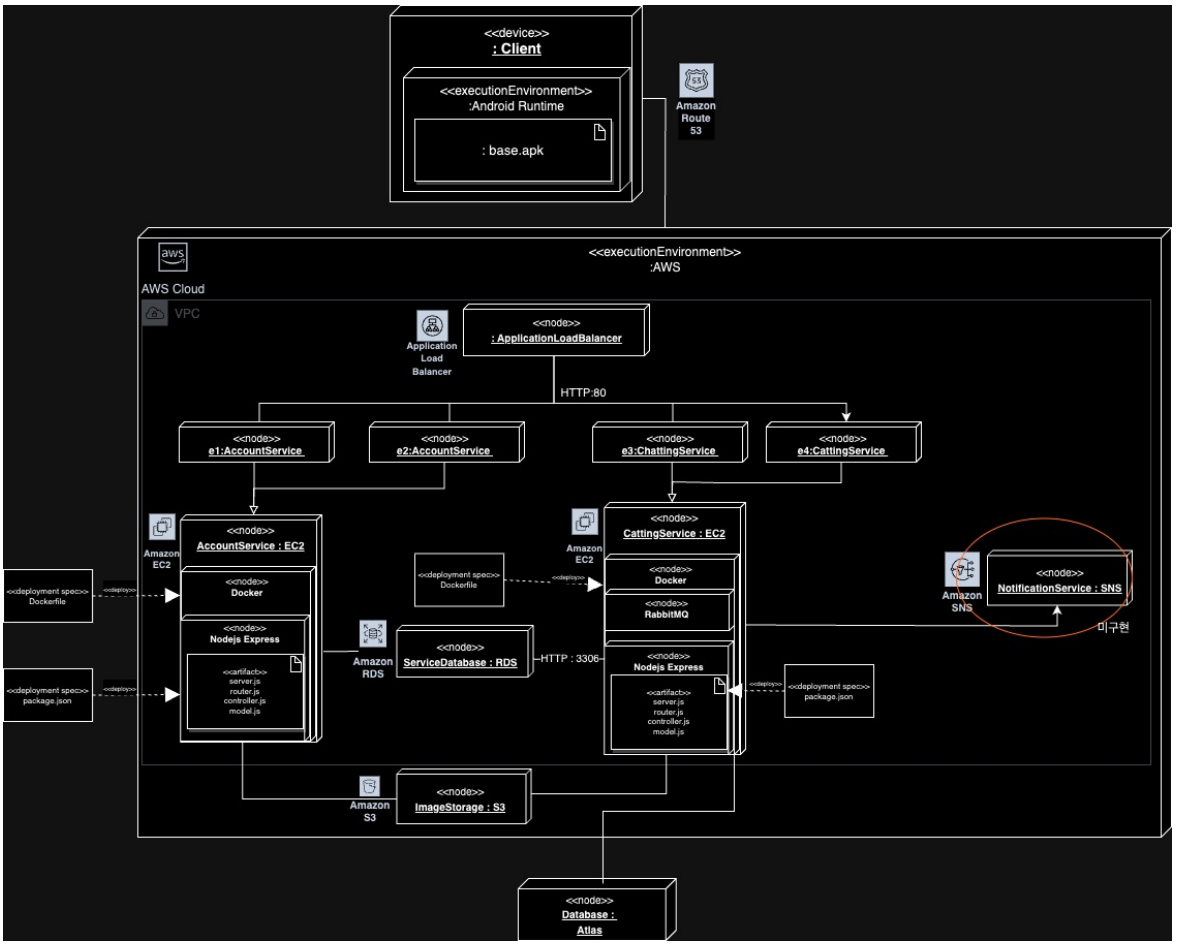
  Tactic 2: Availability

  - **설명:** 로드 밸런서는 클라이언트의 각 요청을 여러 서버로 분산하여 트래픽을 균형있게 분배함. 이로써 서버로 몰리는 트래픽을 효과적으로 분산함으로써 서비스의 성능을 향상시키고 부하를 분산시킴.
  - **적용 방법:** 클라이언트는 도메인에 대한 단일 엔드포인트(로드 밸런서 주소)를 통해 여러 서버에 연결할 수 있다. 로드 밸런서는 각 서버의 상태를 모니터링하고 새로운 요청을 받아들일 수 있는 서버로 트래픽을 분배함. 이때 서버는 필요에 따라 동적으로 추가될 수 있어야 함.

  Tactic 3: modifiability

- **설명:** 소프트웨어를 모듈로 나누어 각 모듈이 독립적으로 개발 및 유지보수될 수 있도록 함. 모듈화된 코드는 특정 기능이나 책임을 담당하므로 해당 모듈을 수정하거나 업그레이드하는 데 다른 부분에 영향을 미치지 않음.
- **적용 방법:** user, chat, chatroom 등 라우팅을 처리.새로운 채팅 관련 기능이 추가되거나 수정되더라도 한 모듈 안에서만 처리되면 됨.

- Deployment Diagram



- 비용 문제로 현재는 ec2 인스턴스 하나만 사용 (e1객체만 실행중)
- 시간 부족으로 sns, Aws elastic beanstock 구현 실패
- 비용문제로 Aws documment대신 mongodb atlats를 사용

8. Architecture style 설명

사용한 Architecture Style	설계에서 필요한 역할	관련된 Class들과의 연관성	수행한 Functional Requirements	수행한 Non-Functional Requirement과 Quality Attributes	장점과 단점
Client-Server	서버, 클라이언트 분리	클라이언트 측에서 각각의 기능에 대한 api에 맞게 데이터를 보내주면 서버에서 데이터를 처리하여 저장혹은 결과를 전송한다.	client : 각종 요청을 보내고 결과를 수신한다 server : 각종 요청을 수신하고 결과를 송신한다. 전체 채팅, 계정 정보등을 저장한다.	서버와 클라이언트의 역할을 분담한다. 종단간 보안을 적용하여 사용자의 정보를 보호할 수 있다.	장점: 역할분담을 통해 서버,클라이언트의 부담을 줄일 수 있다. 단점: 서버-클라이언트 통신간 패킷탈취등의 보안문제가 생길 수 있다.

사용한 Architecture Style	설계에서 필요한 역할	관련된 Class들과의 연관성	수행한 Functional Requirements	수행한 Non-Functional Requirement과 Quality Attributes	장점과 단점
Multi-Tier	클라이언트, 로드벨런서, 서버, 저장소 분리	클래스의 기능 중 데이터를 전송할부분, 받은 데이터를 처리할 부분, 데이터를 저장할 부분을 나누어 각각 역할을 수행한다	client : 각종 요청을 보내고 결과를 수신한다. load balancer : 요청에 대한 트래픽을 분산킨다 server : 요청을 처리한다 DB server : 데이터를 저장한다	AWS내의 여러기능을 사용함으로써 **확장성**을 증대시킨다.	장점: AWS내의 여러 구현되어있는 기능을 사용함으로써 서버개발의 부담을 줄인다. 각 Tier로 구성을 하기때문에 하나의 Tier의 변경에 용이하다. 단점: Tier간의 통신에 신경을 써야한다. Tier별로 instance가 나뉘어있어 관리가 힘들 수 있다.
Load Balancer	다중 EC2 loadBalancing에 사용	채팅 기능의 경우 트래픽이 몰리는 가능성이 커, load balancer를 통해 적절히 트래픽을 분산해야한다.	서버로 몰리는 각종 요청의 트래픽 분산 한다. 하나의 도메인으로 여러개의 서버 접속을 가능하게 한다	**가용성** 및 **확장성** 향상시킨다. 하나의 도메인으로 여러 서버 사용가능하다.	장점 : 하나의 서버 인스턴스가 다운되더라도 서비스전체에는 큰 지장 없다. 단점 : AWS ELB 사용 시 강제로 HTTPS프로토콜을 사용해야한다.(도메인 구매시 비용청구) LoadBalancer가 다운될 경우 서비스 전체를 사용못할 수 있다.
VM	Docker적용시 필요	리눅스 컨테이너에 여러 기능을 추가하며 앱을 좀 더 쉽게 사용할 수 있다.	Containerization: 앱의 모든 구성 요소를 독립적인 도커 컨테이너로 패키징한다.	**가용성** 및 **확장성** 향상을 위하고 배포를 쉽게 한다.	장점 : AWS ELB 사용, 다중 EC2를 구성 할때, 환경구성에 편할 수 있다. 단점 : VM을 적용하지 않은 구조보다 속도가 느리다.
Layered Architecture	Client : android platform Server : AWS EC2 인스턴스 상에서 Docker위에 RabbitMQ, Nodejs 탑제	각 Class는 표준화된 프레임워크에서 일관되게 동작하며, Docker를 통한 컨테이너 환경에서 OS 등의 종속성이 없는 상태에서 동작할 수 있다.	사용자 환경에 관계 없이 채팅 서비스를 이용할 수 있다.	레이어 별로 세팅을 진행하여 **재사용성** 및 **독립성**을 높인다.	장점: 실행환경 세팅을 해놓으면 재사용성이 증가한다. 상위계층, 하위계층 간의 독립성 향상 단점 : 런타임 성능저하 에러처리에 불리
MessageBroker Architecture	chatting system 구현시 RabbitMQ사용	ChattingRoomClass 의 채팅 보내기 기능을 이용하면 pub 할 수 있으며, 채팅방에 참여함으로써 sub 할 수 있다.	pub / sub 매커니즘을 기반으로 메세지 큐에 송신된 채팅을 해당하는 사용자에게 수신할 수 있게한다.	**가용성** 및 **사용성**을 향상 할수있다. 메시지 처리기능의 **신뢰성**을 향상시킨다.	장점: 메시지 처리에 용이하다. 한순간에 몰리는 트래픽 처리를 하기쉽다. 단점: 복잡성이 증가한다.

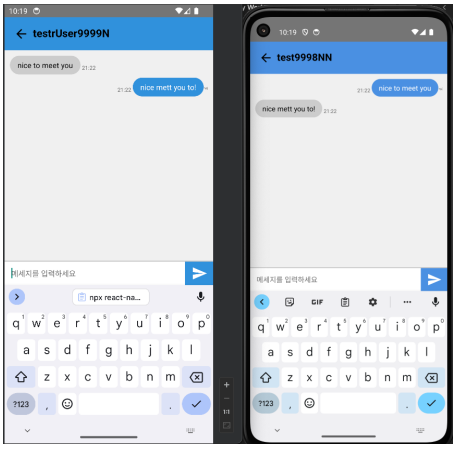
9. 결과와 결론

- 시스템의FR특성과QA에따라서코드의주요부분을캡처하고결과화면캡처
  하여 설명하기

1. 종단간 암호화(E2EE) 채팅 서비스 제공
   - 종단간 암호화를 적용한 채팅 서비스 구현으로 아래와 같은 FR과 QA를 충족한다.
     - **텍스트 전송**에 대한 요구 사항을 만족한다.
     - 안전한 키 공유로 비밀 통신을 실현하여 **보안(Security), 신뢰성(Reliability)**을 제공한다.
     - 키 생성을 위한 SEED 값으로 Hash 함수를 사용하며, 단말기 내 개인키가 변조되지 않는 공간에 보관되기 때문에 **무결성(Integrity)**을 제공한다.
   - 종단간 암호화 프로세스를 요약하자면 아래와 같다.
     1. 회원가입 시 공개키/개인키 생성, 공개키는 서버, 개인키는 단말기에 저장
     2. 방 생성 시 비밀키 생성, 비밀키는 저장 후 상대방의 공개키로 비밀키를 암호화하여 서버 저장
     3. 각 사용자는 단말기에 저장한 개인키로 비밀키를 획득
     4. 안전하게 공유된 비밀키로 비밀 통신
   - 본 프로젝트에서는 아래 이미지와 같이 종단간 암호화가 적용된 채팅 서비스를 제공한다.

```
const encryptSHA256 = data => {
  return CryptoJS.SHA256(data).toString();
};

export const encryptAES256 = (data, secretKey, iv) => {
  return CryptoJS.AES.encrypt(data, CryptoJS.enc.Utf8.parse(secretKey), {
    iv: CryptoJS.enc.Utf8.parse(iv),
    padding: CryptoJS.pad.Pkcs7,
    mode: CryptoJS.mode.CTR,
  }).toString();
};

export const decryptAES256 = (data, secretKey, iv) => {
  return CryptoJS.AES.decrypt(data, CryptoJS.enc.Utf8.parse(secretKey), {
    iv: CryptoJS.enc.Utf8.parse(iv),
    padding: CryptoJS.pad.Pkcs7,
    mode: CryptoJS.mode.CTR,
  }).toString(CryptoJS.enc.Utf8);
};

export const encryptRSA2048 = (data, publicKey) => {
  const crypto = new JSEncrypt({default_key_size: RSA_DEFAULT_KEY_SIZE});

  crypto.setPublicKey(publicKey);

  return crypto.encrypt(data);
};

export const decryptRSA2048 = (data, privateKey) => {
  const crypto = new JSEncrypt({default_key_size: RSA_DEFAULT_KEY_SIZE});

  crypto.setPrivateKey(privateKey);

  return crypto.decrypt(data);
};
```

종단간 암호화를 위한 암호화/복호화 함수 구현



성공적으로 암호화 통신을 하고 있는 화면

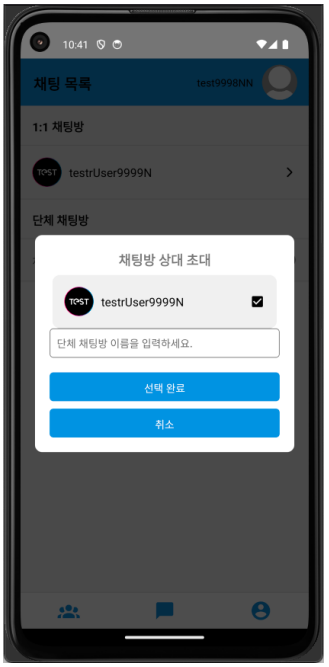
```javascript
export const generateKeyRSA2048 = () => {
  const crypto = new JSEncrypt({default_key_size: RSA_DEFAULT_KEY_SIZE});

  // RSA 키 쌍 생성
  crypto.getKey();

  const publicKey = crypto.getPublicKey();
  const privateKey = crypto.getPrivateKey();

  return {publicKey, privateKey};
};

export const generateSecretKey = data => {
  const secretKey = encryptSHA256(data);

  console.log(secretKey);

  return secretKey;
};
```

종단간 암호화를 위한 키 생성 함수 구현

회원가입 시 최초 1회 암호키 생성

2. 개인 채팅방 및 그룹 채팅방 생성
   - 채팅방(1:1 채팅방, 단체 채팅방) 생성 서비스 구현으로 아래와 같은 FR과 QA를 충족한다.
     - **개인 채팅방 생성, 그룹 채팅방 생성**에 대한 요구 사항을 만족한다.
     - 하나의 전체 채팅방이 아닌, 원하는 사용자만 초대하여 채팅 서비스를 이용할 수 있도록 **보안(Security)**을 제공한다.
     - 채팅방 별로 채팅이 조회되며 관련 기능이 동작하기에, **성능(Performance), 사용성(Usability)**이 좋다.
     - 채팅방을 쉽게 생성할 수 있으며, 다른 종류의 채팅방 필요 시 쉽게 추가할 수 있기 때문에 **확장성(Scalability)**이 좋다.
   - 본 프로젝트에서는 아래 이미지와 같이 다양한 유형의 채팅방 생성 서비스를 제공한다.

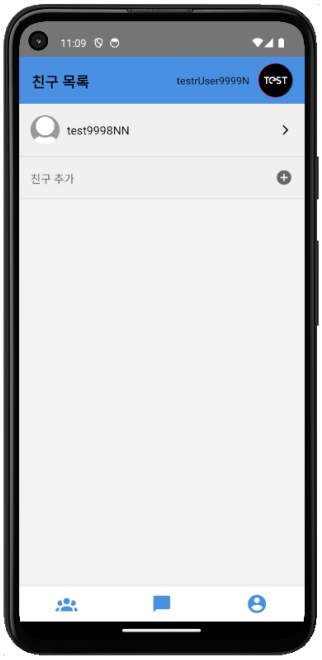
원하는 유형의 채팅방 생성 가능 화면



채팅방 생성 시 원하는 유저만 초대 가능

3. 친구 추가 및 목록 조회
   - 친구 추가 및 목록 조회 서비스 구현으로 아래와 같은 FR과 QA를 만족한다.
     - **친구 추가**에 대한 요구 사항을 만족한다.

    - 친구에 대한 추가 정보 필요 시 추가할 수 있어 **확장성(Scalability)**이 좋다.
- 본 프로젝트에서는 아래 이미지와 같이 친구 추가 및 목록 조회 서비스를 제공한다.



친구 목록을 확인할 수 있는 화면

친구 추가를 할 수 있는 화면

- RABBIT MQ와 socket io 를 사용한 채팅시스템 (서버)

```
const mongoConnect = () => {
    if (NODE_ENV !== "production") {
        mongoose.set("debug", true);
    }
    mongoose
        .connect(MONGO_URL, {
            dbName: "softAch",
        })
        .then(() => {
            const rabbitmq = new RabitMQconnection();
            rabbitmq.consumeMessage();
            console.log("소비시작");
        });
};
```

    - mongoDB 접속할때 메시지 큐에있는 data를 가져와서 업데이트하게 설정

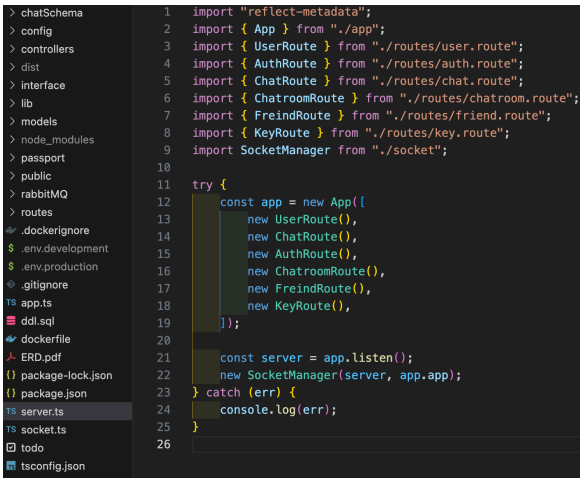
```
private async connect() {
    const connection = await ampq.connect({
        hostname: this.rabbitMQHOST,
        port: Number(this.rabbitMQPort),
        username: this.rabbitMQUser,
        password: this.rabbitMQPassword,
    });
    const channel = await connection.createChannel();
    this.channel = channel;
}

public getChannel = async () => {
    await this.connect();
    return this.channel;
};

public consumeMessage = async () => {
    const queueName = "chat";
    await this.connect();
    await this.channel!.assertQueue(queueName, { durable: true });
    this.channel!.consume(
        queueName,
        async (message) => {
            const data = JSON.parse(message!.content.toString());
            try {
                const chatDataModel = new chat(data);
                await chatDataModel.save();
                console.log("rabbitMQ success");
            } catch (e) {
                console.error("mongofail ", e);
            }
            this.channel?.ack(message!);
        },
        { noAck: false }
    );
};
}
```

- rabbitmq 설정 코드

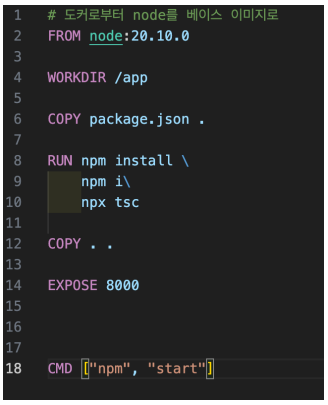
```
socket.on("chat", async (data) => {
    try {
        const chatdata = new chat({
            room: data.room,
            sender: data.sender,
            message: data.message,
            sendAt: Date.now(),
        });

        const message = JSON.stringify(chatdata);
        // rabbitMQ로 메시지 전송
        this.rabbitChannel.sendToQueue(
            "chat",
            Buffer.from(message)
        );
        // const collection = mongoose.connection.collection(
        //     data.room
        // );
        // await collection.insertOne(chatdata);

        this.io.to(data.room).emit("chat", chatdata);
    } catch (e) {
        console.log(e);
    }
});
```

- socket io를 활용한 채팅에서 데이터를 받고, 받은데이터를 rabbitmq로 전송
- 응답이느린 mongodb에 저장하는 내용은 Rabbitmq에게 맡기고 socket통신에 집중함
- **신뢰성**을 높이고, mongodDB에 저장하는 부분을 분리 하기 편하게 만들어 확장성, **배포용이성**을 높임

- MVC 패턴을 적용해 **확장성, 수정용이성 증가**



- docker를 이용한 deployment 증가



```
# 도커로부터 node를 베이스 이미지로
FROM node:20.10.0

WORKDIR /app

COPY package.json .

RUN npm install \
    npm i\
    npx tsc

COPY . .

EXPOSE 8000



CMD ["npm", "start"]
```

- 시스템 아키텍트로써 고려한 내용
  - **AWS 리소스 구성:**
    - **고려 사항:** 어떤 AWS 서비스를 사용할지 고민함. 특정 요구 사항과 예산 제한을 고려해야 하기 때문임.
    - **결론:**
      - 서비스의 규모에 따라 적절한 AWS 리소스를 선택함. 현재 서비스가 그리 크지 않기 때문에 여러 ec2를 사용하지는 않았지만 채팅 서비스가 커질수록 분산 서비스를 이용하는 게 맞다고 판단됨.
      - 현재 실제 서비스중인 어플이아님, 또한 수익이날 수 없기때문에 최대한 비용을 적게 만들수 있도록 노력함
  - **보안 및 개인정보 보호:**
    - **고려 사항:** chat service는 보안이 핵심적이기 때문에 이에 대해 고민함.
    - **결론:** 종단간 암호화를 이용하여 사용자의 데이터를 보호하여 중간자 공격 및 불법 접근으로부터 안전을 제공.
  - **서비스의 신뢰성**
    - 고려사항 : 사용자의 데이터, 활동이 누락되면 안됨
    - 결론 :
      - Message Queue, fault 및 error처리 복구에대한 방법을 세워 신뢰성을 높어야된다

- 참고문헌
  - aws deployment diagram

    https://medium.com/@kachmarani/uml-deployment-diagram-in-modern-word-caee0a2ecaa3
  - sns sqs

    https://aws.amazon.com/ko/blogs/architecture/best-practices-for-implementing-event-driven-architectures-in-your-organization/

    https://channel.io/ko/blog/tech-backend-aws-sqs-introduction
  - aws rabbitmq

    https://docs.aws.amazon.com/ko_kr/amazon-mq/latest/developer-guide/getting-started-rabbitmq.html
  - aws eb

    https://catalog.us-east-1.prod.workshops.aws/workshops/3fd6c80b-39f2-4534-b69c-c400aed50c67/ko-KR/beanstalk/deploy-new-app
  - Rabbit MQ 공식문서

    https://www.rabbitmq.com/#features