마이크로 서비스 명세

채팅 송수신

▼ 1. B: 채팅 페이지

▼ 2. C-1: 채팅 송신

1. 이름: 채팅 송신
2. 버전: 0.0.1
3. 입력 타입: string
4. 크기: `(추후 입력)`
5. 에러 처리 정책: `미정`
6. 연결 가능 서비스
    a. 채팅 저장
7. 램: `미정`
8. 작성언어: JAVA

▼ 3. C-2: 채팅 저장

1. 이름: 채팅 저장
2. 버전:0.0.1
3. 입력 타입: string
4. 크기: 미정
5. 에러 처리 정책
6. 연결 가능 서비스
7. 램
8. 작성 언어: JAVA

▼ 4. C-3: 채팅 수신

1. 이름: 채팅 수신
2. 버전: 0.0.1
3. 입력 타입: string
4. 크기:
5. 에러 처리 정책:
6. 연결 가능 서비스:
7. 램:
8. 작성 언어: JAVA

▼ 5. E: 채팅 DB

1. 이름: 채팅 DB
2. 버전: 0.0.1
3. 입력 타입: -
4. 크기:
5. 에러 처리 정책:
6. 연결 가능 서비스:

7. 램:
8. 작성 언어: JAVA

▼ 6. 명세표(마이크로서비스 풀 명세)

명세 항목 | 서비스	채팅 페이지	채팅 송신	채팅 수신	채팅 저장DBConnector	채팅방 생성	채팅
BCE 패턴	B	C	C	E	C	C
이름	채팅 페이지	채팅 송신	채팅 수신	채팅 저장	채팅방 생성	채팅
버전	0.0.1	0.0.1	0.0.1	0.0.1	0.0.1	0.0.
입력 타입	-	아래 표 참조	아래 표 참조	아래 표 참조	아래 표 참조	아래
출력 타입	아래 표 참조	아래 표 참조	아래 표 참조	아래 표 참조	아래 표 참조	아래
에러 처리 정책	1. 예외처리	1. 예외처리	1. 예외처리	1. 예외처리	1. 예외처리	1. 예
연결 가능 서비스	-	채팅 저장	Finish	채팅 수신	채팅방 저장	채팅
Method	GET	POST	GET	POST	POST	DEL
URL	{base_url}/ws/chat/	{servicename}/send/	{base_url}/ws/chat/{roomid}/	{servicename}/chatsave/	{base_url}/chatroom/	{bas
작성 언어	Javascript	JAVA	JAVA	JAVA	JAVA	JAV

▼ 7. 조합 명세(메시지 전송)

	채팅 페이지(B)	채팅 송신(C)	채팅 저장(C)	채팅 DB(E)	채팅 수신(C)
서비스 순서	1	2	3	3-1	4
이름	채팅 페이지	채팅 송신	채팅 저장	-	채팅 수신
버전	0.0.1	0.0.1	0.0.1	-	0.0.1
입력 타입	-	String	-	-	String
크기	-	미정	미정	-	미정
에러 처리 정책	-	미정	-	-	미정
연결 가능 서비스	-	채팅 저장	채팅 수신	-	Finish
램	-	미정	미정	-	미정
작성 언어	Javascript	JAVA	JAVA	MySQL	JAVA

▼ 수정사항

1. 입력타입 정리 구체화 시킬것 ex) Rest API와 같은 입력 타입들 or 포트
   a. 채팅 송신(채팅 페이지 → 채팅 송신)

```
// [POST] {base_url}/chat/{roomid}/
{
  "user" : username:string,
  "message": string
}
```

   b. 채팅 저장(채팅 송신 → 채팅 저장:DB connector)

```
// [POST] {servicename}/chatsave/
{
  "sessionKey": redisSessionKey:string,
  "user": username:string,
  "room": roomId:string,
  "message": string,
  "createdAt": string
}
```

c. 채팅 수신(채팅 저장 → 채팅 수신)

```
// [POST] {servicename}/send/
{
  "sessionKey": redisSessionKey:string,
  "user": username:string,
  "room": roomId:string,
  "message": string
}
```

d. 채팅방 생성

```
// [POST] {base_url}/chatroom/
{
  "user": username,
  "room": roomId,
}
```

e. 채팅방 삭제

```
// [DELETE] {base_url}/chatroom/{roomId}/
```

2. 출력 타입

a. 채팅 송신

```
{
  "status": statusCode(int),
  "message": string(에러 메시지)
}
```

b. 채팅 저장

```
// 채팅저장 서비스 -> DB connector
INSERT INTO MESSAGE (USERNAME, ROOMID, MSG, CREATEDAT)
  VALUES (VARCHAR, VARCHAR, VARCHAR, DATETIME)
```

```
{
  "status":statusCode(int),
  "message": string(에러 메시지)
}
```

c. 채팅 수신

```
{
  "status":statusCode(int),
  "message": string(에러 메시지)
}
```

d. 채팅방 생성

```
INSERT INTO CHATROOM (USERNAME, ROOMID)
  VALUES (VARCHAR, VARCHAR)
```

e. 채팅방 삭제

```
DELETE FROM CHATROOM WHERE USERNAME, ROOMID
```

3. 조합 명세에서 단계적으로 Sequnce diagram을 그릴것

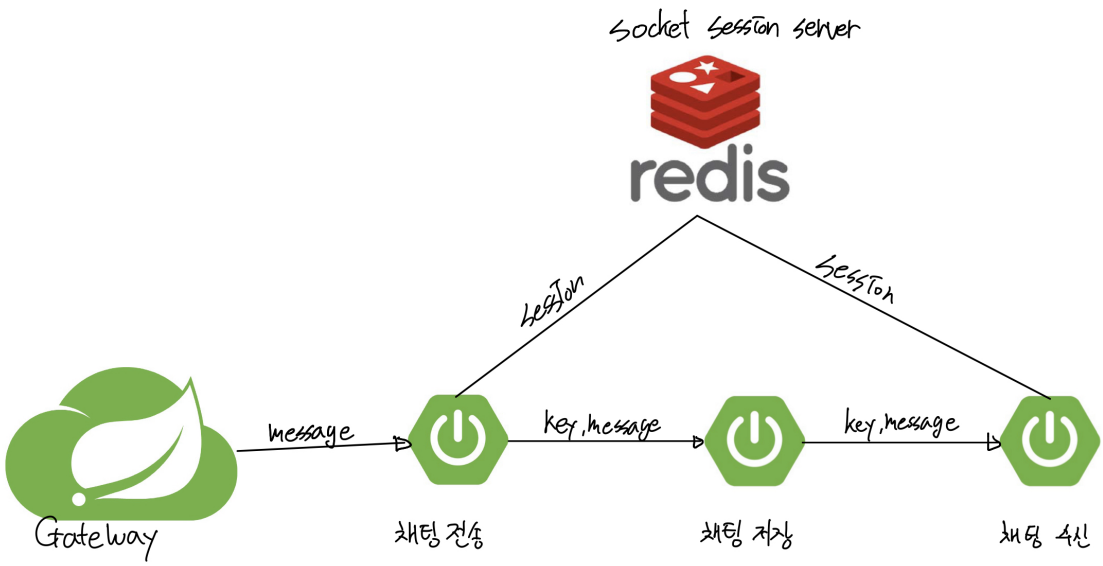
a. 채팅 페이지 → 채팅방 생성 : Username과 RoomId를 key로 저장

b. 채팅 송신 → 채팅저장 : data{message} 전송 및 채팅 저장에서 POST

c. 채팅 저장 → 채팅 송신

4. 크기와 RAM은 실행 pool 로 가는것을 추천, 크기같은 경우는 loc(line of code)로 설정 무관
   a. 램은 최대 램이 아닌 실행중 차지하는 공간
5. 채팅 DB를 Entity로 하지말고 DB Connector를 추가하여 Entity로 만들어 줄것
6. 에러처리정책 중요도를 나눠서 적기, 성능적면과 가용성적인 면을 비교하여 나눌 것
   a. 예외 처리 또는 로그
   b. 성능적면, 가용성적인면
      i. 로그를 생성하면 성능에는 좋지만 가용적이지 않다.
      ii. 서킷 브레이커 사용하면 성능은 안 좋지만 가용성 높다.
   c. 채팅방 생성, 채팅방 삭제, 채팅 송신
      a. transaction을 사용하여 전체 실패 처리
   d. 채팅 저장
      a. 저장 실패 로그 생성
   e. 채팅 수신
      a. 사용가능한 동일 기능 서비스 호출
      b. 사용가능한 서비스 없으면 동일 서비스 생성후 메시지와 소켓 세션 키 전달
7. 명세 부분에 추가적으로 들어갈 부분 생각 ex) endpoint
   a. {base_url}/chatting 처럼 엔드포인트

채팅 서비스 구조



→ 위 그림의 구조와 세션 데이터를 전달해주는 구조