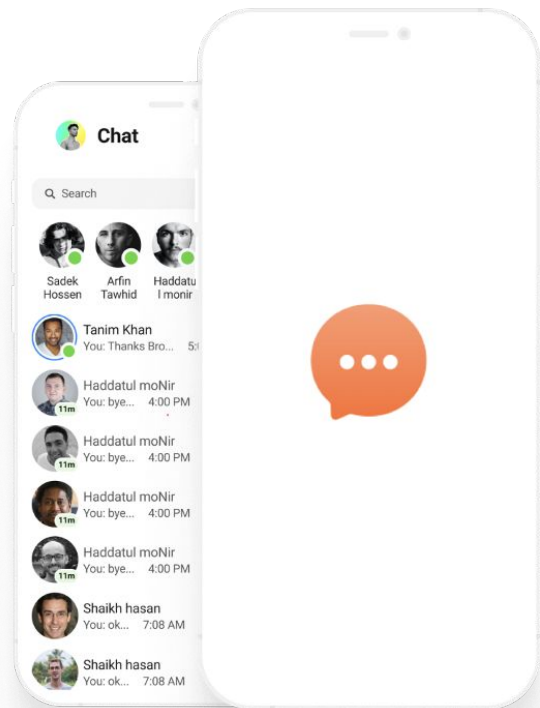


PLOP MESSENGER

경남 Plop 팀 최종발표



CONTENTS

01 팀 소개

02 주제

03 주요 구현 기능 및 시연

04 프로젝트 리뷰

TEAM PLOP



김주현 (팀장/Server)



이제호 (Server)



김호준 (iOS)



김가희 (Android)

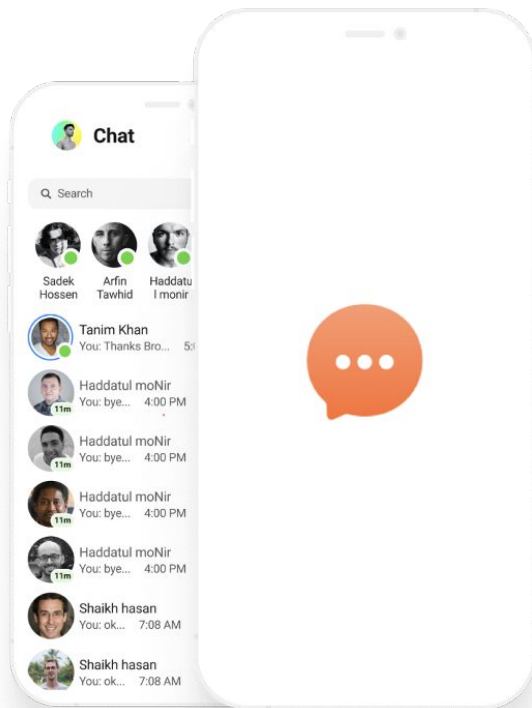
프로젝트 주제



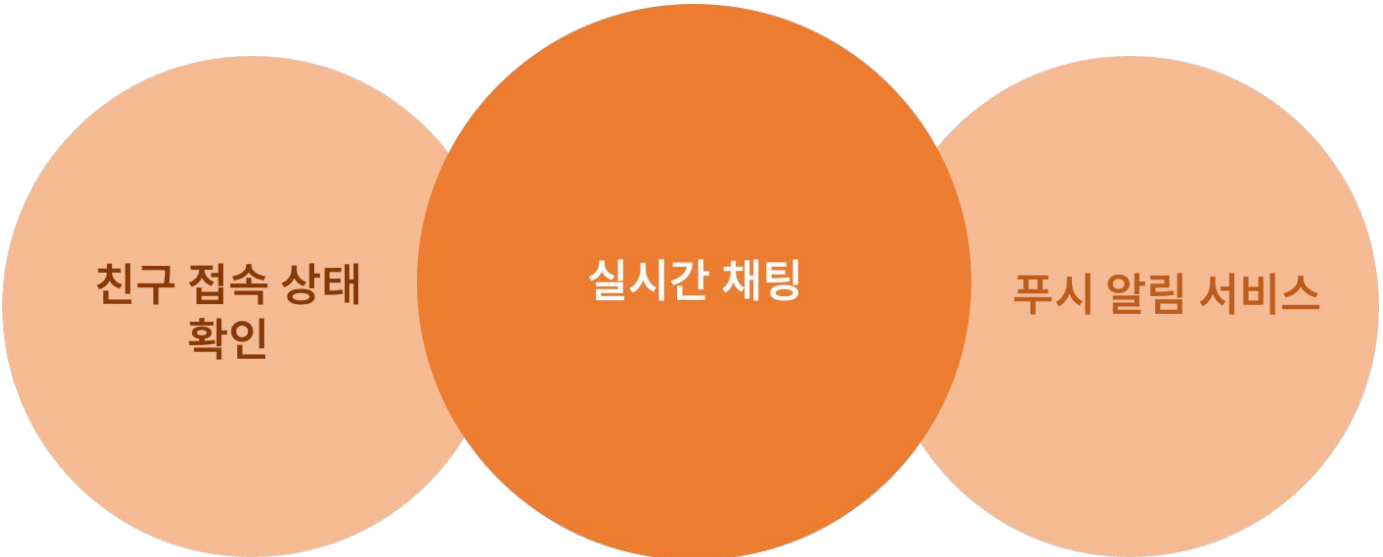
PLOP

MESSENGER

: Meta messenger 클론 프로젝트



주요 구현 기능



친구 접속 상태
확인

실시간 채팅

푸시 알림 서비스

시연 영상



친구1

그룹 채팅 기능

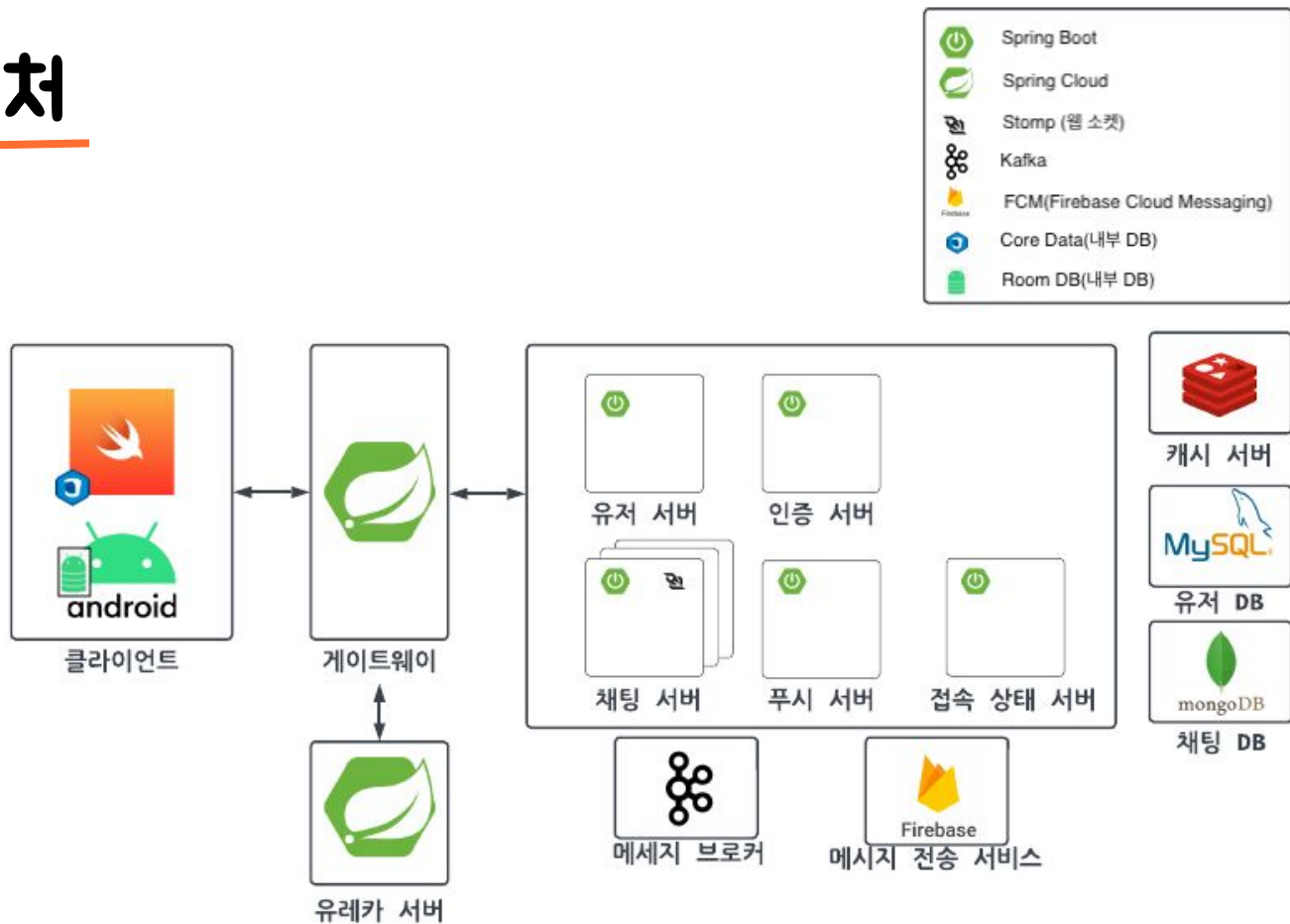
여러 사람과 서로 채팅 할 수 있습니다.



친구2

<https://www.youtube.com/watch?v=CHYVsNMhxLk>

아키텍처



배포 아키텍처

- Swift
- Core DB
- Android Studio
- Room DB
- Spring Boot
- Spring Cloud
- FireBase Cloud Messaging
- Kafka
- Redis
- Mongo DB
- MySQL
- Docker
- Mongo Atlas
- AWS EC2
- AWS RDS
- Grafana



IOS



Android



API Gateway
Load Balanced



Service Registry
Discovery



채팅 서버



접속 상태 서버



인증 서버



푸시 알림 서버



유저 서버



메시지 큐



캐시



모니터링



MySQL

유저 DB



채팅 DB



FCM

프로젝트 리뷰

프로젝트 리뷰 - 김호준(iOS)

목표 달성 현황

1. 코드를 하나의 파일에 몰아서 작성하지 말고 Domain Layer, Data Layer, Presentation Layer로 역할을 나누어 개발
2. 웹 소켓을 활용한 채팅 서비스 구현
3. XCode tool 중 하나인 Instruments을 사용할 수 있고, 이를 활용한 성능 개선 경험을 가진다.

✅ Domain(Entities, Usecase), Data(Network, Realm), Presentation(View, ViewModel, ViewController) 로 나누어서 하나의 객체가 너무 많은 일을 하지 않게 분리

⚠️ 1:1 채팅, 그룹 채팅을 구현하는 부분은 성공. 하지만, 에러 핸들링이 제대로 구현되어있지 않고, 많은 부분에서 생각한 대로 동작하지 않는 코드들을 개선하지 못함



프로젝트 리뷰 - 김호준(IOS)

문제 및 해결

😞 서버 -> 내부 저장소 -> 화면 으로 이어지는 로직을 어떻게 구현할까?

1. 내부 저장소에 저장이 필요한 요청의 경우에는 응답이 오면 이를 바로 내부 저장소에 저장하게 구현
2. RxSwift와 Realm을 활용하여 Realm의 CRUD를 관찰할 수 있는 Observable을 생성
3. View Model이 이를 관찰하면서 Realm 내부에 변화가 생길 때 마다 UI 업데이트

```
func observeFetchAll() -> Observable<[ChatRoom]> {  
    return Observable.deferred {  
        let realm = self.realm  
  
        let objects = realm.objects(RMChatRoom.self)  
        return Observable.array(from: objects)  
            .map({ $0.map({ $0.asDomain() }) })  
    }  
    .subscribe(on: MainScheduler.instance)  
}
```

ex) 내부에 저장된 채팅방 목록의 변화를
관찰하는 Observable

프로젝트 리뷰 - 김호준(iOS)

좋았던 점

- 팀원들의 적극적인 도움
- 새로운 것들을 습득하기 위해 몰입했던 기간
- 개발자로서의 성장
- 직접적으로 느껴본 협업에서의 부족한 점

프로젝트 리뷰 - 김호준(iOS)

아쉬웠던 점

너무 현재 내 상태를 고려하지 않고 도달하고 싶은 상태에만 집중했던 게 아닌가

- Combine → RxSwift
- CoreData → Realm
- URLSessionWebSocketTask → Starscream
- [추가된 기술 스택] Moya, Alamofire, MessageKit

알은 기본 지식

- 내가 개발하는 플랫폼에 대한 이해도가 부족해 질문을 하기가 어려움
- 타 플랫폼에 대한 기본적인 지식이 부족하니 의사소통이 어려움

프로젝트 리뷰 - 김호준(IOS)

앞으로의 다짐 / 느낀점

- 좋은 계획과 목표는 현재의 나를 잘 알아야 나오지 않을까?
- 의사소통도 아는 게 있어야...

프로젝트 리뷰 - 김가희(Android)

목표 달성 현황

1. 개선 경험을 통해 내가 작성한 코드와 로직에 대한 이해 얻기
2. 테스트 코드 작성
3. 프로젝트 1차 완성
4. 소켓통신 경험으로 기능 폭 넓히기

✓ 비동기작업에 대한 이해와 이를 통한 개선 완료

✗ 일정상 실패..

✓ 목표로 잡았던 핵심 기능(채팅, 알람, 접속 상태) 구현 완료

✓ STOMP를 다루기 위한 추가적인 라이브러리를 사용하지 않고, 직접 구현하여 이해를 넓힘

프로젝트 리뷰 - 김가희(Android)

어려움

1. 비동기 처리라는 것이 직접적으로 와닿지 않았고, 지금까지 제대로된 이해없이 남의 코드를 따라하여 비동기 처리를 하였더니, 화면이 늦게 나타나는 문제가 발생
2. STOMP에 대한 자료 부족과 일반적으로 사용되는 라이브러리의 부재

프로젝트 리뷰 - 김가희(Android)

문제 및 해결

비동기 처리라는 것이 직접적으로 와닿지 않았고, 지금까지 제대로된 이해없이 비동기 처리를 하였더니 화면 로드가 느렸다.

1. 공식 문서와 여러 깃헙 코드를 참고하여 코루틴, Flow에 대한 이해

특히 Dispatchers를 이용한 스레드 관리, Scope지정을 통한 범위 관리, 병렬 처리 등에 대한 이해

2. 이 지식을 바탕으로 화면이 늦게 나타나는 문제 해결

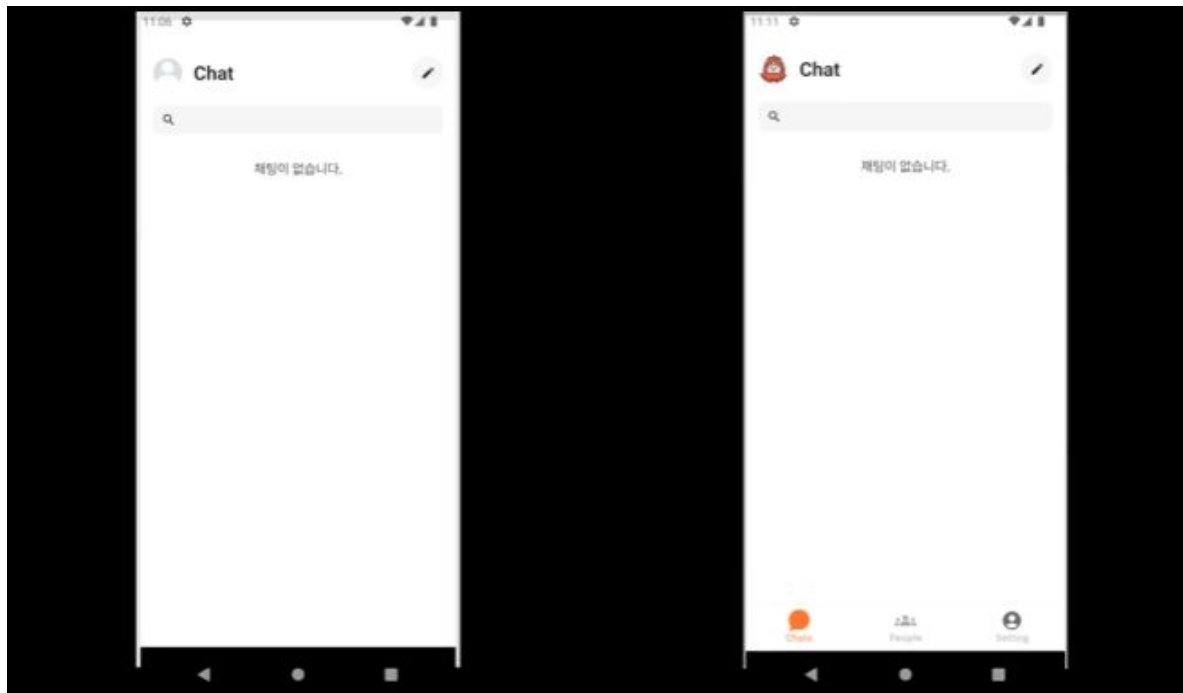
여러 종류의 데이터를 한 번에 불러올 때 이를 병렬적으로 처리하고 데이터를 불러오는 로직을 IO스레드에서 처리

```
init {  
    viewModelScope.launch { this: CoroutineScope  
        getRemoteChatRoom()  
        getLocalChatRooms()  
        getPresence()  
    }  
}
```



```
init {  
    viewModelScope.launch { this: CoroutineScope  
        launch { getRemoteChatRoom() }  
        launch { getLocalChatRooms() }  
        launch { getPresence() }  
    }  
}
```

프로젝트 리뷰 - 김가희(Android)



개선 후

개선 전

프로젝트 리뷰 - 김가희(Android)

문제 및 해결

STOMP에 대한 자료 부족과 일반적으로 사용되는 라이브러리의 부재로 인한 두려움, 그리고 처음 사용하는 기술에 대한 불안감

1. 왜 안드로이드에서 통용되는 STOMP 라이브러리가 없을까? -> 굳이 라이브러리가 필요하지 않아서 그렇지 않을까?

2. 다른사람들의 STOMP라이브러리와 공식문서를 보면서 작동방식을 이해

3. 프로젝트 작동방식에 맞게 OkHttp3의 web socket listener를 이용하여 직접 구현

=> 웹 소켓 방식에 대해 이해하고 새로운 기술에 대한 두려움을 많이 줄었습니다.

프로젝트 리뷰 - 김가희(Android)

그 외 얻게 된 것

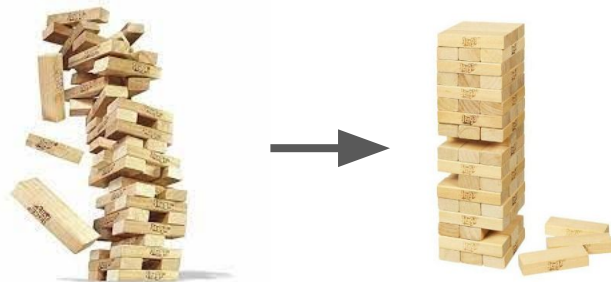
1. 이해 없이 사용한 라이브러리에 대한 제대로된 이해!

- Dagger-Hilt를 통한 의존성 주입, 더 나아가 멀티 바인딩 까지 이해하고 작성
- Retrofit, Okhttp3, Interceptor등 통신 관련 라이브러리에 대한 이해 후 사용
- 내부 DB(Room)와 관련된 이해와 후 사용

=> 왜 이런 코드를 작성했는지 설명할 수 있습니다!!

2. 개발 공부 방법, 문제 해결 방법의 확립

- 개발하다가 궁금한 부분이 있으면 기록하고 해결
- 공식문서는 여러 번 봐도 손해가 없다는걸 알게 되었습니다.
- 남의 코드를 그저 복붙하기 보다는 따라쓰더라도 직접 쳐보자.



무너질 뻔한 제 알팍한 지식!.. 조금 단단해졌습니다 ...

궁금한 것 ...

Aa 이름	태그	해결 여부
Retrofit - client?	Data Retrofit Interceptor	<input checked="" type="checkbox"/>
ExperimentalCoroutinesApi::class?	Coroutine	<input checked="" type="checkbox"/>
@HiltAndroidApp	DI	<input checked="" type="checkbox"/>
ViewModel 주입	ViewModel	<input checked="" type="checkbox"/>
Navigation to Activity	Compose Navigation	<input checked="" type="checkbox"/>
StringRes	Android	<input checked="" type="checkbox"/>
get()문법	Kotlin	<input checked="" type="checkbox"/>
from.lifecyclesResumed()		<input checked="" type="checkbox"/>
정규 표현식	Kotlin	<input checked="" type="checkbox"/>
slice vs substring	Kotlin	<input checked="" type="checkbox"/>
@Immutable	Kotlin	<input checked="" type="checkbox"/>
MessageFormatter	Kotlin	<input checked="" type="checkbox"/>
레이징 되지 않는 데이터에서 Lazy 와 그 밖 Column의 차이	Compose	<input checked="" type="checkbox"/>
companion object	Kotlin	<input type="checkbox"/>
viewmodel 예 factory		<input type="checkbox"/>
OkHttp vs retrofit	Retrofit Interceptor	<input checked="" type="checkbox"/>
Interceptor	Interceptor	<input checked="" type="checkbox"/>
dispatcher IO		<input checked="" type="checkbox"/>
Network Interceptor class	Interceptor	<input checked="" type="checkbox"/>
DI Scope		<input type="checkbox"/>
Entity naming		<input checked="" type="checkbox"/>

프로젝트 리뷰 - 김가희(Android)

아쉬운점

- 예외 처리에서 보일러 플레이트 코드가 생겼고, 이에 대해 더 나은 개선점을 찾았지만 시간상 적용하지 못한점.
- 테스트 코드 작성을 하지 못한점.

앞으로의 다짐 / 느낀점

- 공식 문서는 정말 중요하고, 공부한것을 스스로 사용할 줄 알아야한다.
- 코드를 한 줄 한 줄 이해하며 기본지식을 쌓아가는것이 중요하다.
- 새로운 기술을 사용할 때 너무 큰 두려움을 갖지말자.
- 초기 설계를 단단히 하고가자!

프로젝트 리뷰 - 이제호(Server)

목표 달성 현황

1. MSA 설계 및 구현 능력 함양
2. 경험해보지 못한 아키텍처와 기술들을 활용하여 문제 해결 능력에 대한 폭 확장
3. 협업을 통한 팀 단위 문제해결 능력 향상

✓ MSA 아키텍처 설계 및 게이트웨이, 인증, 유저, 푸시 서버 개발

✓ MSA 기반 아키텍처 상에서 캐시 (Redis), 푸시 알림 (FCM), 클라우드(AWS) 활용

✓ Git을 통한 형상 관리, 정기 회의를 통한 지속적인 의사소통

프로젝트 리뷰 - 이제호(Server)

목표

- MSA 설계 및 구현 능력 함양
- 경험해보지 못한 아키텍처와 기술들을 활용하여 문제 해결 능력에 대한 폭 확장
- ✓ MSA 아키텍처 설계 및 게이트웨이, 인증, 유저, 푸시 서버 개발
- ✓ MSA 기반 아키텍처 상에서 캐시 (Redis), 푸시 알림(FCM), 클라우드(AWS) 활용
- 게이트웨이
 - JWT 검증 및 각 서비스 연결
- 인증
 - JWT 기반 인증 구현
 - Redis - refresh token , 이메일 인증 코드
- 유저 서버
 - AWS S3 - 프로필 이미지 저장
- 푸시 서버
 - FCM - 푸시 알림 전송

프로젝트 리뷰 - 이제호(Server)

목표

- 협업을 통한 팀 단위 문제해결 능력 향상

 Git을 통한 형상 관리 + 정기 회의를 통한 지속적인 의사소통

Git 브랜치 전략 : main - develop - feature 브랜치로 분리하여 단일 저장소를 팀원들이 효과적으로 활용

Git 커밋 컨벤션 : 커밋 컨벤션 통일하여 커밋 내용 전달에 용이

정기 회의 : 다양한 배경의 사람들이 모여 하는 회의를 경험함으로써 소통에 대한 방법 고찰

프로젝트 리뷰 - 이제호(Server)

문제 및 해결

- 보안 관련 설정을 어떻게 관리할 것인가?

➡ 로컬 환경 및 배포 환경에서 환경변수를 통해 관리

```
spring:
  application:
    name: auth-service
  jpa:
    show-sql: true
  # hibernate:
  #   ddl-auto: create-drop
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://plop-rds.cyjccp4psnuz.ap-northeast
    username: ${PLOP_DB_USER} # plop-db-user
    password: ${PLOP_DB_PWD} # plop-db-pwd
  redis:
    host: 127.0.0.1
    port: 6379

mail:
  host: smtp.gmail.com
  port: 587
  username: ${MAIL_USER}
  password: ${MAIL_PWD}
  properties:
    mail:
      debug: true
      smtp:
        auth: true
        starttls:
          enable: true
```

GitGuardian

[sgdevcamp2022/plop] SMTP credentials exposed on GitHub

google-cloud-compli.

Potentially Compromised Credentials for plop-df864

GitGuardian

[sgdevcamp2022/plop] AWS Keys exposed on GitHub

```
export ACCESS_KEY=AKIA
export SECRET_KEY=wfe/
export PLOP_DB_USER=pl
export PLOP_DB_PWD=pas
export FCM_KEY_PATH=pl
export MAIL_USER=jh181
export MAIL_PWD=nrlaw1
```

프로젝트 리뷰 - 이제호(Server)

문제 및 해결

- 모바일에서 서버와 어떻게 통합할 것인가?

➡ AWS에 배포를 통한 Public IP로 연결 및 테스트

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
AUTH-SERVICE	n/a (1)	(1)	UP (1) - ip-172-31-12-151.ap-northeast-2.compute.internal:de3e94c314edd19908c6833f26ded4d3
CHAT-SERVICE	n/a (1)	(1)	UP (1) - ip-172-31-12-151.ap-northeast-2.compute.internal:2e2836881941f1eef5720733251e4210
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - ip-172-31-12-151.ap-northeast-2.compute.internal:gateway-service:8000
PRESENCE-SERVICE	n/a (1)	(1)	UP (1) - ip-172-31-12-151.ap-northeast-2.compute.internal:3199db47d9d463d1afa72a2f461159f2
PUSH-SERVICE	n/a (1)	(1)	UP (1) - ip-172-31-12-151.ap-northeast-2.compute.internal:5c97ed454bfff63d5c30493e43da9e0b4
USER-SERVICE	n/a (1)	(1)	UP (1) - ip-172-31-12-151.ap-northeast-2.compute.internal:763d15f8d2e9c991477c214dc012313e

- 각 서버의 로그를 어떻게 관리할 것인가?

➡ nohup 명령어를 통해 인스턴스에 접속 중이지 않을 때도 서버가 동작

➡ nohup.out 파일에 로그가 기록됨

⚠ 임시 해결 => 도커를 통한 컨테이너별 로그, ELK를 통한 로그 통합 관리

```
[ec2-user@ip-172-31-12-151 server]$ tree -L 2
.
├── auth
│   ├── build
│   ├── build.gradle
│   ├── dump.rdb
│   ├── gradle
│   ├── gradlew
│   ├── gradlew.bat
│   ├── nohup.out
│   ├── settings.gradle
│   └── src
└── chat
    ├── build
    ├── build.gradle
    ├── gradle
    ├── gradlew
    ├── gradlew.bat
    ├── nohup.out
    ├── settings.gradle
    └── src
```

프로젝트 리뷰 - 이제호(Server)

성찰

- **좋았던 점**
 - 서버, 특히 스프링에 대한 경험이 부족했지만 자신감을 가지게 되었다.
 - 에러코드 및 예외처리 중요성을 깨닫고 적용해볼 수 있었다.
 - 개발뿐만 아니라 배포해볼 수 있었다.
- **아쉬운 점**
 - MSA임에도 DB 분리 X
 - 테스트에 대한 고려가 미흡 -> 주요 로직 중 일부만 테스트 코드 작성
 - 로그에 대한 전반적인 요소가 미흡 -> 로그 수집, 저장, 조회

프로젝트 리뷰 - 이제호(Server)

앞으로의 다짐 / 느낀점

- 과거 프로젝트에서 놓치고 있던 것
 - 프로젝트의 목표 혹은 목적
 - 일정 관리를 통한 프로젝트의 진행
 - 예외처리, 로그
- 전공 지식의 필요성
 - 기술에 대한 습득력
 - 원활한 의사소통
- 소통의 중요성
 - 생각하는 관점이 다름.
 - 문제 해결에 큰 도움이 됨.

프로젝트 리뷰 - 김주현(Server)

목표 달성 현황

MSA 구조에서의 백엔드 개발

웹소켓 기술을 사용하여 채팅 기능을 구현

부하테스트를 통해 트래픽 처리를 해보며 성능 개선하기

✓ Feign 을 적용하여 서비스들간 통신

✓ 소켓 통신을 STOMP를 이용해 pub/sub 구조로 개발

✓ nGrinder 사용, 리팩토링과 톰캣 설정으로 성능 개선

프로젝트 리뷰 - 김주현(Server)

목표

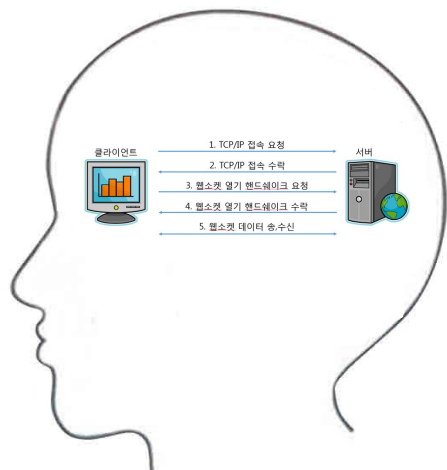
웹소켓 기술을 사용하여 채팅 기능을 구현

WebSocket

STOMP

웹소켓 프로토콜

프로젝트 리뷰 - 김주현(Server)



WebSocket

```
@Configuration
@RequiredArgsConstructor
@EnableWebSocket
public class WebSocketConfig implements WebSocketConfigurer {

    private final ChatHandler chatHandler;

    @Override
    public void registerWebSocketHandlers(WebSocketHandlerRegistry registry) {

        registry.addHandler(chatHandler, "ws/chat").setAllowedOrigins("*");
    }
}
```

STOMP

```
@RequiredArgsConstructor
@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfig implements WebSocketMessageBrokerConfigurer {

    1 usage 11:51:06
    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        // '/chatting/queue/room/{room_id}'
        registry.enableSimpleBroker("/chatting/queue", "/chatting/topic");
        registry.setApplicationDestinationPrefixes("/chatting/pub");
    }

    1 usage 11:51:06
    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry
            .addEndpoint("/ws-chat")
            .setAllowedOrigins("*");
        registry.addEndpoint("/ws-chat").setAllowedOrigins("*").withSockJS();
    }
}
```

프로젝트 리뷰 - 김주현(Server)

STOMP

SimpleBroker.. 이용자가 증가할 수록
서버 메모리에 부담,,

외부 메세지 브로커




 RabbitMQ


 kafka


프로젝트 리뷰 - 김주현(Server)




채팅 기능은 처음이라..

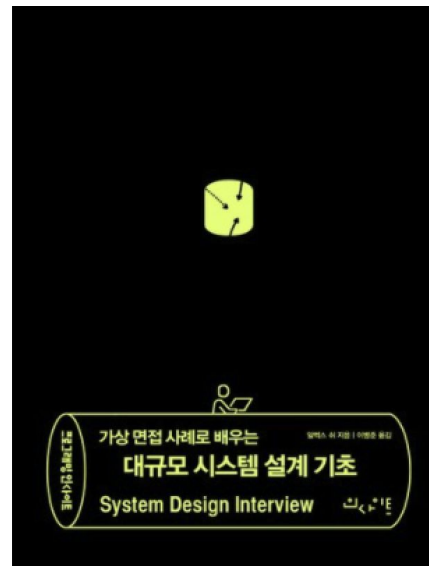


**정소희** 오후 11:58
아키텍처 설계 중 참고하고 있는 책입니다.
<http://www.yes24.com/Product/Goods/102819435>

 **YES24**
가상 면접 사례로 배우는 대규모 시스템 설계 기초 - YES24
“페이스북의 뉴스 피드나 메신저, 유튜브, 구글 드라이브 같은 대규모 시스템은 어떻게 설계할까?” IT 경력자라도 느닷없이 대규모 시스템을 설계하려고 하면 막막하다고 느낄 수 있다. 특히나 면접을 보는 상황이라면 더욱 눈앞이 캄캄해질 것이다. 복잡한 시스템을 설...



 11  2 



12장 채팅 시스템 설계

프로젝트 리뷰 - 김주현(Server)

채팅 기능은 처음이라..

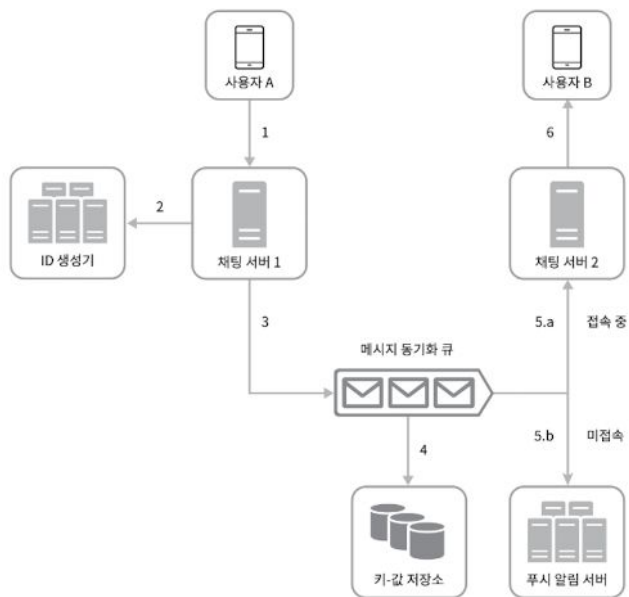
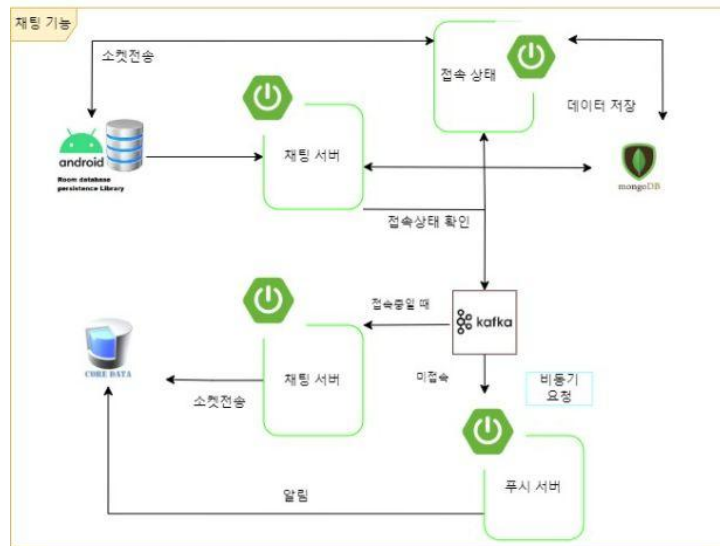
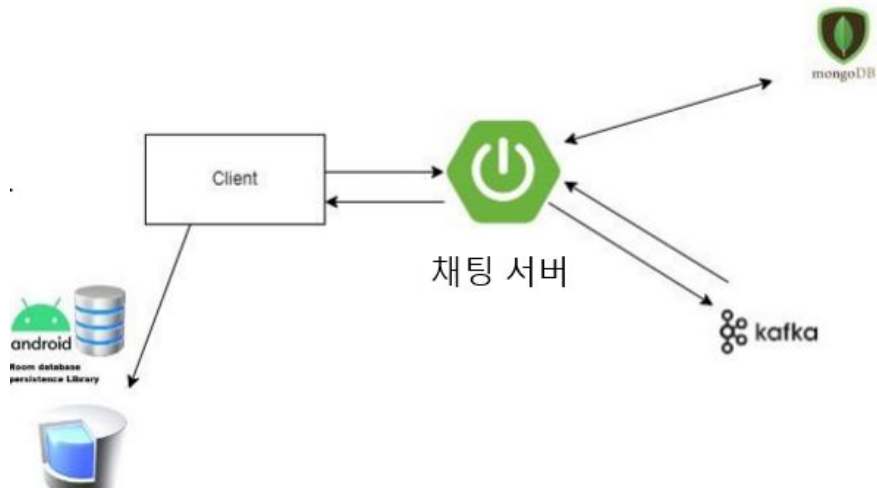


그림 12-12



프로젝트 리뷰 - 김주현(Server)

채팅 기능은 처음이라..



메시지 처리 흐름

프로젝트 리뷰 - 김주현(Server)

목표

부하테스트를 통한 트래픽 처리를 해보며 성능 개선하기

성능 테스트

프로젝트 리뷰 - 김주현(Server)

무엇을? 어떻게 ?

자주 쓰는 API에 nGrinder를 사용하자



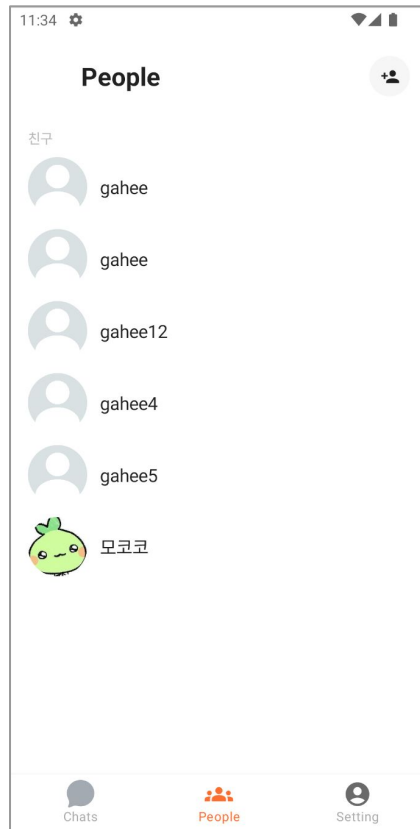
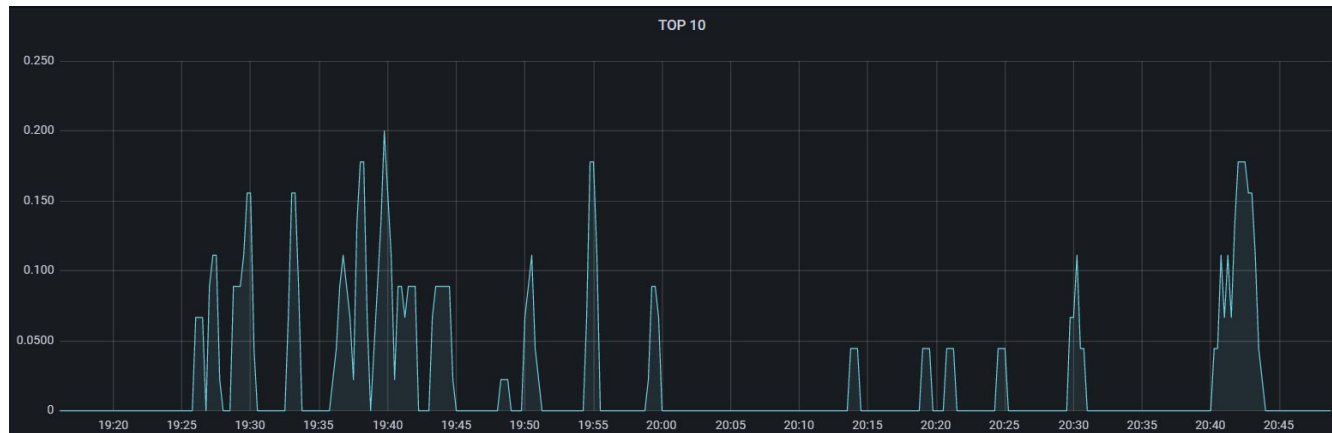
프로젝트 리뷰 - 김주현(Server)

측정 환경



프로젝트 리뷰 - 김주현(Server)

GET /friend
친구 목록 조회



프로젝트 리뷰 - 김주현(Server)

첫 테스트

Vuser: 300, localhost, AWS RDS, API: /friend

에이전트 1

총 Vuser 300

약 2~5분간

총 Vuser	300
에이전트	1
프로세스 스레드	2 / 150
샘플링 무시 횟수	0
TPS	20.8
최고 TPS	27
평균 테스트시간	13,623.72 ms
총 실행 테스트	2,376
성공함 테스트	2,376
에러	0
성능 리포트	
테스트 대상 서버	
127.0.0.1	

시작시간	2023-02-21 16:10:02	종료시간	2023-02-21 16:12:02
테스트 기간	00:02:00 HH:MM:SS	동작 시간	00:02:00 HH:MM:SS
설명	Connections Size : 10		

Performance

CSV 다운로드

TPS



TPS : 20.8

프로젝트 리뷰 - 김주현(Server)

그럼 어떻게 개선하지?

소스코드분석 -> 리팩토링

1. @Transactional(readOnly = true) 추가

```
Hibernate: select userentity0_.id as id1_1_, userentity0_.created_at  
Hibernate: select friendenti0_.receiver_id as receiver1_0_, frienden  
Hibernate: select userentity0_.id as id1_1_0_, userentity0_.created_  
Hibernate: select userentity0_.id as id1_1_0_, userentity0_.created_  
Hibernate: select userentity0_.id as id1_1_0_, userentity0_.created_  
Hibernate: select userentity0_.id as id1_1_0_, userentity0_.created_  
Hibernate: select userentity0_.id as id1_1_0_, userentity0_.created_  
Hibernate: select userentity0_.id as id1_1_0_, userentity0_.created_
```

2. 쿼리 변경

```
if (friend.getSenderId().equals(senderEntity.getId())) {  
    user = userRepository.findById(friend.getReceiverId()).orElse(null);  
} else {  
    user = userRepository.findById(friend.getSenderId()).orElse(null);  
}
```

친구들을 한명씩 조회 → 친구 여러명 한번에 조회

```
userRepository.findByIdIn(userIdList);
```

```
select * from friend where id in (?, ?, ?)
```

```
Hibernate: select userentity0_.id as id1_1_, userentity0_.created_at as created_2_1_, userentity0_.updated_at as updated_3_1_, userentity0_.access_at as access_a4_1_, use  
Hibernate: select friendenti0_.receiver_id as receiver1_0_, friendenti0_.sender_id as sender_i2_0_, friendenti0_.status as status3_0_ from friend friendenti0_ where (frie  
Hibernate: select userentity0_.id as id1_1_, userentity0_.created_at as created_2_1_, userentity0_.updated_at as updated_3_1_, userentity0_.access_at as access_a4_1_, use
```

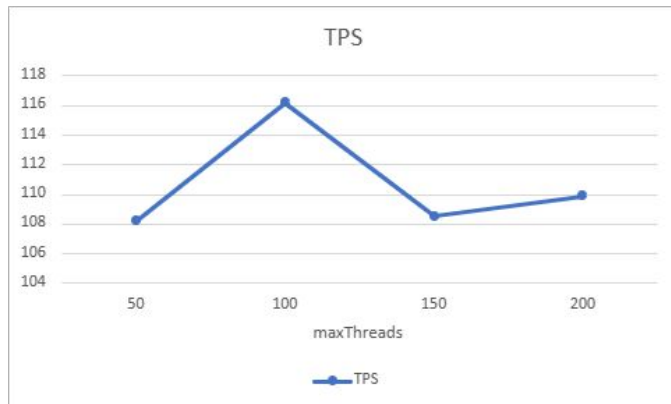
프로젝트 리뷰 - 김주현(Server)

톰캣 설정

- maximumPoolSize : Connection Pool 이 가질 수 있는 최대 커넥션 개수
10 , 20, 30

- maxThreads (최대 스레드풀 개수): 기본값 200
서버 사양에 따라 가장 많이 좌우되는 설정값
직접 성능 테스트를 실행하여 최적의 값을 찾을 필요가 있다.

50개, 100개, 150개, 200개



프로젝트 리뷰 - 김주현(Server)

부하 테스트
Vuser: 300
5분

성능 개선 결과

개선 사항
readOnly
리팩토링

maxThreads: 100

maximumPoolSize : 20

중 Vuser	300
에이전트	1
프로세스 스레드	2 / 150
샘플링 무시 횟수	0
TPS	186.6
최고 TPS	200
평균 테스트시간	1,605.68 ms
총 실행 테스트	32,601
성공한 테스트	32,594
에러	7
성능 리포트	
테스트 대상 서버	
127.0.0.1	

시작시간	2023-02-21 22:40:30	종료시간	2023-02-21 22:43:30
테스트 기간	00:03:00 HH:MM:SS	동작 시간	00:03:00 HH:MM:SS
설명	@Transactional(readOnly = true) 히카리클 20 스레드 100		

Performance

CSV 다운로드

TPS



TPS: 20.8 → 186.6 (약 797% 증가!)

프로젝트 리뷰 - 김주현(Server)

좋았던 점

STOMP를 이용한 웹소켓 통신과 HTTP 통신의 차이점을 이해

성능 테스트

- 리팩토링을 통해서 충분히 성능 개선을 할 수 있다!
- 툴킷 설정을 하되 서버 환경을 고려해야 한다!

프로젝트 리뷰 - 김주현(Server)

아쉬운 점

- 개발 환경만 고려하여 배포 시점이 늦어져 모바일과 연동이 늦어짐
- 모니터링 툴을 충분히 활용하지 못함

앞으로의 다짐 / 느낀점

- 협업할 때 개발 초기 가데이터를 넣어 FE와의 연동을 위한 배포, 통합시 에러 최소화 (코드리뷰를 통해 알게됨!)
- 성능 지표, 자원(CPU 사용률, disk, 메모리)수치 등을 보며 성능 개선을 할 수 있는 안목을 길러야겠다!
- 카프카, 래빗MQ와 같은 외부 라이브러리, 툴 도입시 배포 환경에서의 운영을 고려하여 도입

캠프를 마치며,,



김주현

값진 경험 잘 소화해서 더욱 성장하겠습니다. Smilegate 너무 감사해요~!



이제호

실력 상승 + 자신감 상승 + 좋은 팀원들..!
아낌없는 지원과 조언 감사합니다!



김가희

좋은 팀원들 덕에 프로젝트를 완성할 수 있었고
이를 통해 지식을 단단하게 다질 수 있었습니다. 감사합니다!



김호준

어려울 때 많이 도와주셔서 고맙고, 앞으로도 좋은 인연으로 남았으면
좋겠네요 감사합니다!

감사합니다