국민대학교
전자정보통신대학
컴퓨터공학부

캡스톤 디자인 I

종합설계 프로젝트

프로젝트 명	커리어 한잔
팀 명	17조
문서 제목	결과보고서

Version	2.0
Date	2024-MAY-23

팀원	이 하람 (조장)
	김 승언
	김 혜은
	공 채연
	권 해담
	홍 지민

문서 정보 / 수정 내역

Filename	최종수행결과보고서-커리어한잔.docx
원안작성자	김승언, 김혜은, 이하람
수정작업자	공채연, 권해담, 김승언, 김혜은, 이하람, 홍지민

수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2024-05-10	김승언	1.0	최초 작성	보고서 양식 정리
2024-05-17	김혜은	1.1	내용 추가	수정된 연구내용 추가
2024-05-18	이하람	1.2	내용 추가	향후 추진 계획 수정
2024-05-20	공채연	1.3	내용 추가	활용/개발된 기술 추가
2024-05-21	권해담	1.4	내용 추가	기대효과 및 활용방안 추가
2024-05-21	김승언	1.5	내용 추가	테스트 케이스 추가
2024-05-22	공채연	1.6	내용 추가	UseCase Diagram 추가
2024-05-23	김혜은	1.7	내용 추가	사용자/운영자 매뉴얼 추가
2024-05-23	이하람	1.8	내용 추가	시스템 구조 및 설계도 추가
2024-05-23	홍지민	1.9	내용 추가	시스템 비기능(품질) 요구사항 추가
2024-05-23	권해담	2.0	내용 추가	추진 배경 및 필요성 추가

목 차

1 개요

□ 프로젝트 개요

'커리어 한잔' 프로젝트는 개발자들 간의 부담없는 실시간 커리어 대화를 제공하는 서비스이다. 이 프로젝트에서는 **'위치 지정 기능', '커피챗 매칭 기능', '채팅 기능', '리뷰 기능', '푸시알림 기능'** 등이 있다.

위치 지정 기능은 사용자가 실제 본인의 위치가 아닌, 주변 카페 중 한 곳을 자신의 위치로 지정하여 다른 사용자들에게 공개된다. 여기서 실제 카페 위치, 운영 정보로 카페를 선택하는데

도움을 줄 수 있다. 이 기능을 통해 직접적인 위치 노출을 최소화하며, 더불어 커피챗 매칭 후에는 약속 장소를 따로 찾아볼 필요 없이 해당 카페에서 커피챗을 진행할 수 있어 편리함을 제공한다.

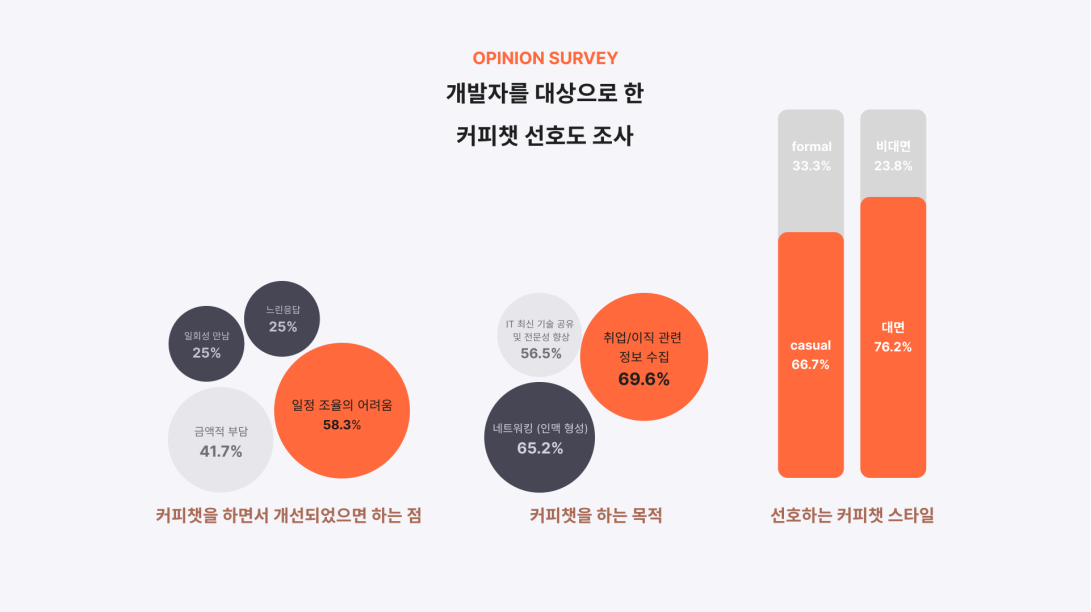
커피챗 매칭 기능은 유저의 반경에 있는 카페를 선택해 원하는 상대의 프로필을 확인하고, 매칭 요청을 보낼 수 있도록 한다. 사용자는 커리어 한잔에서 제공하는 여러 목적 중 하나를 골라 요청을 보내기 때문에, 요청을 받는 사람에게 손쉽게 더 많은 정보를 전달할 수 있다.

채팅 기능은 커피챗 매칭이 성사된 사용자끼리 더 깊은 네트워킹을 진행할 수 있도록 연결시켜 주는 것이 주요 기능이다. 이 기능을 통해 약속 장소와 시간을 논의할 수 있을 뿐만 아니라 커피챗이 종료된 이후에도 연락을 이어나갈 수 있다.

리뷰 기능은 커피챗이 끝나고 두 사용자는 커피콩점 리뷰로 피드백을 남기게 된다. 사용자가 커피챗 요청을 보낼 때 커피 온도를 표시하게 되는데, 이 온도는 커피챗이 종료된 후 남긴 리뷰의 '커피콩점'이 반영된 것으로 상대방에게 나를 어필하는 수단이 되기도 한다. 따라서, 사용자는 자연스럽게 좋은 인상을 주기 위해 노력하게 될 것이고 이는 사용자 간의 신뢰성 있는 서비스를 제공한다.

푸시 알림 기능은 사용자가 커피챗 요청, 수락, 취소, 종료 알림을 받을 수 있도록 한다.

☐ 추진 배경 및 필요성



요즘 직장인들 간에는, 특히나 IT 분야에서는 이직을 위한 정보를 얻거나 서로가 가진 업무 경험과 인사이트를 공유하고 채용을 제안하는 등, 매우 다양한 목적으로 '커피챗'이라는 문화가 점차 발달하고 있다. 커피챗이란, 비즈니스 관계에서 커피 한잔하듯 가볍게 만나 인맥을 쌓고 정보를 공유하는 것을 말한다. 그러나 아직 우리나라에서는 이 문화가 서구권에 비해 활발하지 않고, 실제 커피챗이 이뤄지기까지의 절차가 다소 번거로우며 접근성이 떨어진다. 이러한 점에서 가볍고 캐주얼한 커피챗을 위한 새로운 접근 방식의 필요성을 느끼게 되었다.

이에 **모든 IT 분야의 현직자들 간의 부담없는 커리어 대화를 위한 커피챗 매칭 서비스**를 만들어보자는 의견으로 뜻을 모았다.

물론 커피챗 매칭에 활용할 수 있는 혹은 커피챗 매칭만을 위한 플랫폼들이 이미 존재하지만, 모두 온라인을 통한 대화를 지향한다는 한계가 있다. 그러나 코로나 팬데믹을 지나며 모두가 경험했듯이 온라인의 편리함만으로는 대체할 수 없는 오프라인 만남의 이점들이 있다. 이는 커피챗이라는 개방적이고 자유로움을 지향하는 문화에도 예외없이 적용될 것이다. 또한 우리에게 진짜 필요한 커피챗 문화는 당장 나에게 필요한 정보만을 얻기 위한 일회성 연락이 아닌, **서로 도움을 주고받으며 함께 발전해나가는 지속적 관계 형성**이며 이를 위해서는 **대면**

만남이 필수적이라고 판단했다.

이러한 동기로 '주변 개발자들과 대면 만남이 가능한 실시간 커피챗 매칭 서비스, **커리어 한잔**' 프로젝트를 시작하게 되었다.

2 개발 내용 및 결과물

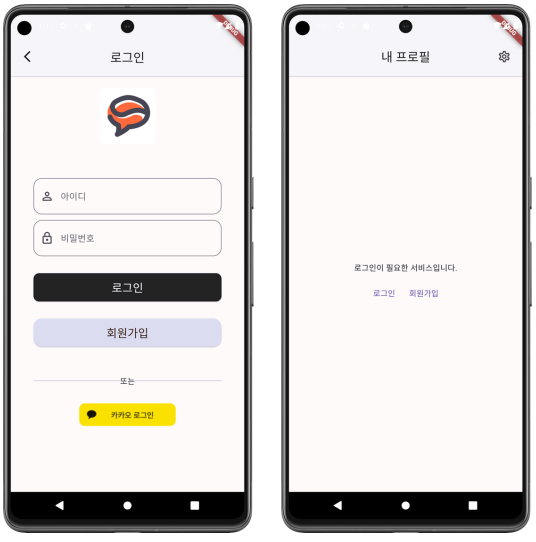
☐ 목표

'커리어 한잔' 프로젝트는 개발자들 간의 실시간 커리어 대화를 위한 서비스이다. 사용자에게 **1) 위치(카페) 지정**, **2) 커피챗 매칭**, **3) 채팅**, **4) 리뷰**, 그리고 **5) 푸시 알림**과 같은 기능을 제공하여 IT 분야 내 네트워킹을 형성하도록 돕는다. 이 프로젝트로 현직 개발자들 간에 더 부담 없이 정보를 공유하고 인맥을 형성할 수 있는 커피챗 문화를 활성화시키고자 한다. 사용자는 주변 카페를 선택하여 위치를 공유함으로써 안전하게 커피챗 요청을 주고받을 수 있고, 매칭 시 편리하게 약속 장소를 설정하도록 돕는다. 또한, 커피챗 후에는 상대방에 대한 리뷰를 작성하여 커피챗 노쇼 등의 문제를 방지한다. 이 프로젝트는 궁극적으로 온라인 대화만으로는 부족한 오프라인 만남의 가치를 강조하며, 사용자들 간의 지속적인 관계 형성을 지원한다.

☐ 연구/개발 내용 및 결과물

2. .1 연구/개발 내용

회원관리를 위한 일반 로그인 & 소셜 로그인 기능



- **프론트엔드**

일반 로그인 : 로그인 여부는 토큰의 유무로 결정된다. 서버에서 클라이언트로 전송된 토큰은 Flutter의 flutter_secure_storage 패키지를 사용하여 안전하게 저장된다. 이 패키지를 활용하여 토큰을 시스템의 내부 저장소인 keystore 영역에 저장함으로써, 앱이 루팅되더라도 토큰에 접근할 수 없도록 보안을 강화했다.

앱이 처음 실행될 때, 토큰이 존재하지 않는 경우에는 로그인 화면을, 토큰이 존재하는 경우에는 메인 화면을 보여준다. 만약 토큰이 있지만 기간이 만료된 경우, 마이페이지에 접속하여 서버에서 유저 정보를 요청했을 때 로그인 기간이 만료되었다는 안내 문구를 받게 된다. 로그아웃 시 마이페이지는 로그인과 회원가입 버튼이 있는 화면으로 전환되어, 여기서 재로그인을 할 수 있다.

카카오 로그인 : 소셜 로그인 기능으로 카카오 로그인을 구현했다. 이를 위해 카카오 네이티브 앱 키를 사용하여 flutter_kakao_sdk_user 라이브러리를 초기화했다. 사용자의 디바이스에 카카오톡이 설치되어 있으면 카카오톡을 통해 로그인하고, 그렇지 않으면 카카오 계정으로 로그인을 시도한다. 로그인에 성공하면 카카오 서버로부터 액세스 토큰을 받아서 서버에 전달한다. 카카오의 액세스 토큰은 카카오계정과 앱에 종속적인 값으로, 이것을 이용해 기존 로그인에서의 아이디와 비밀번호를 대체할 수 있었다.

- **백엔드**

일반 로그인 : JWT 토큰을 사용하여 로그인 인증 기능을 구현했다. 사용자가 로그인할 때 사용자의 UUID와 JWT 토큰을 저장한다. 생성된 JWT 토큰은 클라이언트에게 반환되고, 클라이언트는 이 토큰을 인증된 요청에 포함하여 서버로 전송한다.

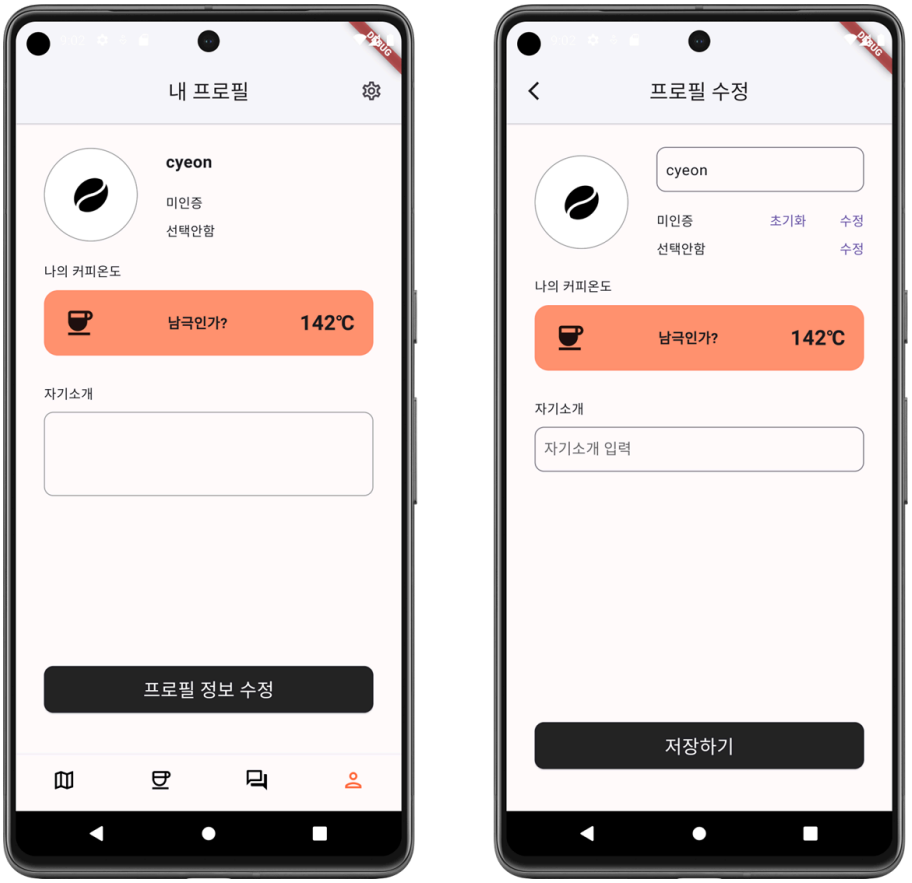
클라이언트는 JWT 토큰을 포함해서 요청을 보낸다. 이때, 서버에서 토큰의 유효성을 검증한다. 토큰이 유효한 경우, 요청을 계속 진행하고, 유효하지 않은 경우 예외를 발생시켜 요청을 거부한다. 토큰의 유효성은 토큰의 서명을 확인하고, 만료 시간을 확인하여 수행된다.

컨트롤러 메서드 파라미터로 `@AuthenticationPrincipal` 어노테이션을 추가하여 JWT 토큰을 추출한다. 이를 바탕으로 사용자를 식별하고, 필요한 경우 해당 사용자 객체를

반환하여 컨트롤러 메서드에서 사용할 수 있도록 한다.

카카오 로그인 : 클라이언트에서 받은 카카오 토큰을 활용하여 Kakao API를 호출하고, 사용자 정보를 받아왔다. 이후 해당 사용자에게 JWT 토큰을 발급하여 클라이언트에게 반환했다.

마이페이지 (직무/자기소개/회사 수정)



- **프론트엔드**

초기에는 마이페이지에 접속할 때마다 매번 서버에 해당 유저의 정보를 요청하는 방식으로 구현하였으나, 이는 비효율적이었을 뿐만 아니라 프로필을 수정했을 때 다른

화면에 갔다오지 않는 한 프로필 화면이 갱신되지 않는다는 문제점이 있었다. 이를 해결하기 위해 provider를 사용했다.

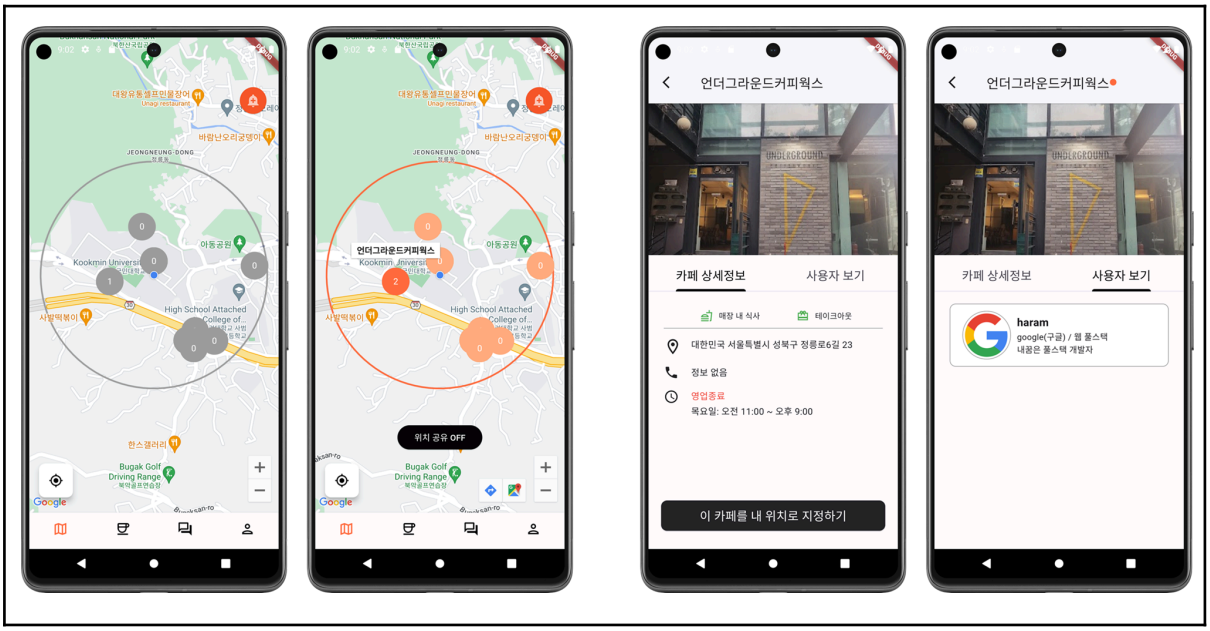
사용자가 로그인하여 새롭게 토큰을 발급받거나, 스토리지에 토큰이 존재하는 채로 앱을 실행할 때 해당 토큰을 이용해 서버로부터 닉네임, 회사, 직무 등의 유저 정보를 가져왔고, 이 유저 정보를 Provider를 사용해 전역으로 저장했다. 이후에는 매번 서버에 유저정보를 요청하지 않아도 로그인, 로그아웃 또는 유저 정보의 수정이 이루어질 때마다 유저 정보를 다른 위젯들에게 자동으로 notify할 수 있었다.

프로필 수정 화면에서는 닉네임과 자기소개를 입력한 후 저장하기 버튼을 눌러 수정할 수 있고, 또한 회사나 직무를 변경하는 화면으로 이동할 수도 있다. 이전에는 회사나 직무 변경 후 프로필 수정 화면으로 돌아오기 위해 모든 화면을 일일이 pop해야 했지만, 이는 사용자 편의성 측면에서 부적절하다고 판단되었다. 그래서 프로필 수정 페이지에 Routesettings 속성을 지정하고 popUntil() 함수를 사용하여, 회사/직무 변경 성공 안내 모달에서 확인 버튼을 누르면 프로필 수정 화면으로 자동으로 돌아오도록 개선했다.

- **백엔드**

  유저 정보를 불러오는 api (GET: auth/detail)를 제공해 인증된 유저에 한해서 자신의 유저 정보를 열람할 수 있는 api를 제작했다.
  이후 직무, 닉네임, 자기소개 등의 프로필을 수정하는 요청을 보낼 수 있도록 각각의 엔드포인트를 생성해 PUT 요청을 보내면 서버에서 유저의 정보를 수정한 뒤, 변경된 정보를 반환한다.

카페 지정 및 위치 공유 기능



커피챗을 요청하거나 받기 위해서는 '위치 공유 on' 상태여야 한다. '위치 공유 on' 상태란, 사용자가 자신의 위치를 지정하면 주변의 다른 사용자들과 위치가 공유되어 커피챗 요청을 주고 받을 수 있는 상태를 말한다. 여기서 사용자의 안전을 고려하여 실제 사용자의 위치를 실시간으로 공유하도록 하지 않고, 사용자의 반경 500m 내 카페 중 한 곳을 자신이 보일 위치로 지정하도록 했다. 또한 사용자가 '위치 공유 OFF' 버튼을 클릭하거나, 지정한 카페에서 반경이 벗어나는 경우, 그리고 앱을 완전히 종료하거나 로그아웃하는 경우에 위치 공유를 끄도록 했다.

- **프론트엔드**

  동작 과정: 지도 페이지(메인)에서 사용자의 반경과 반경 내의 카페들(각 카페를 자신의 위치로 지정한 사용자 수)을 표시한다. 사용자가 카페의 위치나 사용자 수를 보고 마음에 드는 카페를 터치하면 카페의 상세페이지로 이동한다. 카페 상세페이지에는 카페의 주소, 영업 시간 등을 볼 수 있는 '상세정보 탭'과 해당 카페를 자신의 위치로 지정한 사용자들의 목록을 볼 수 있는 '사용자 보기 탭'이 있다. '사용자 보기 탭'은 기본적으로 블러 처리되어 있어 사용자들의 정보를 볼 수 없다. 사용자가 카페 상세페이지 하단에 있는 카페 지정 버튼을 클릭해야 '위치 공유 on' 상태가 되며 블러 효과가 사라진다. 사용자들의 프로필 로고, 닉네임, 회사/직무, 자기소개 한 줄을 볼 수 있으며, 한 사용자를 클릭하면 보다 상세한 정보와 커피챗 요청 버튼이 있는 모달창이 뜬다.

개발 내용:

카페 지정/해제, 사용자 목록 보기 - 지도에서 사용자의 주변 카페 목록이 가져와질 때, 먼저 그 카페들에 대해 서버에 http 요청을 보내 전체 사용자 목록을 가져오도록 했다. 같은 카페들에 대해 웹소켓(stomp) sub 요청을 보내고, 사용자가 특정 카페를 자신의 위치로 지정할 때는 해당 카페에 사용자를 추가하는 pub 요청을, 반대로 해제할 때는 삭제하는 pub 요청을 보내도록 했다. 그렇게 서버를 통해 실시간으로 변경 사항이 반영되는 주변 카페들의 사용자 목록을 화면에서도 즉각적으로 보여줄 수 있도록 플러터의 ChangeNotifierProvider를 사용해 사용자 목록을 관리했다. 이를 통해 사용자는 자신의 위치를 지정하는 동시에 다른 사용자들에게 보일 수 있으며, 다른 사용자들이 주변 카페에 들어오거나 나가는 것을 바로 확인할 수 있다.

위치 공유 온/오프 UI - 우선 위치 공유 온/오프 상태를 관리하기 위한 MyCafe 모델을 ChangeNotifierProvider로 생성하여 위치 공유를 켤 때 즉, 카페를 지정할 때는 myCafe에 해당 카페 정보(카페 ID, 위도, 경도)를 저장하고, 카페 지정해제될 때 myCafe를 다시 null로 초기화하도록 했다. myCafe에 저장된 카페 ID가 있으면 온라인, 없으면 오프라인으로 구분하여 프론트 단의 UI를 다르게 표시하도록 했다. 오프라인이면 지도에 사용자의 반경과 카페 마커의 색상을 흑백으로 표시하고 위치 공유 끄기 버튼을 없앴으며, 카페 상세페이지의 사용자 보기 탭을 블러 처리하였다. 온라인이면 지도에 색상을 넣고, 위치 공유 끄기 버튼을 활성화시켰으며 사용자 보기 탭에서 사용자들의 프로필 정보를 확인하고 탭하여 커피챗 요청을 보낼 수 있게 하였다.

자동 위치공유 오프라인 처리 - 사용자가 직접 '위치공유 OFF' 버튼을 클릭하는 상황 외의 상황들에서 자동으로 위치 공유를 끄도록 처리하였다. 위치공유 오프라인 처리는 서버에 카페에서 사용자 삭제 pub 요청을 보내는 것뿐만 아니라, 프론트 단 처리를 위해 myCafe를 null로 초기화하는 것을 포함한다. 이것을 간편하게 재사용하고자 서비스 클래스를 만들어 내부에 실제 오프라인 처리 서비스를 추가하고 만들어진 서비스 객체를 Provider로 전역으로 관리하였다. 먼저 사용자의 반경에서 기존 지정한 카페가 벗어나는 경우는 타이머를 통해 구현했다. 카페를 지정할 때 해당 카페의 위치와 사용자의 위치를 5분마다 체크하는 타이머를 함께 구동하여, 거리가 500m를 넘을 때 오프라인 처리를 한다. 로그아웃할 때는 로그아웃 로직에 오프라인 처리를 포함시키고, 앱을 종료할 때는

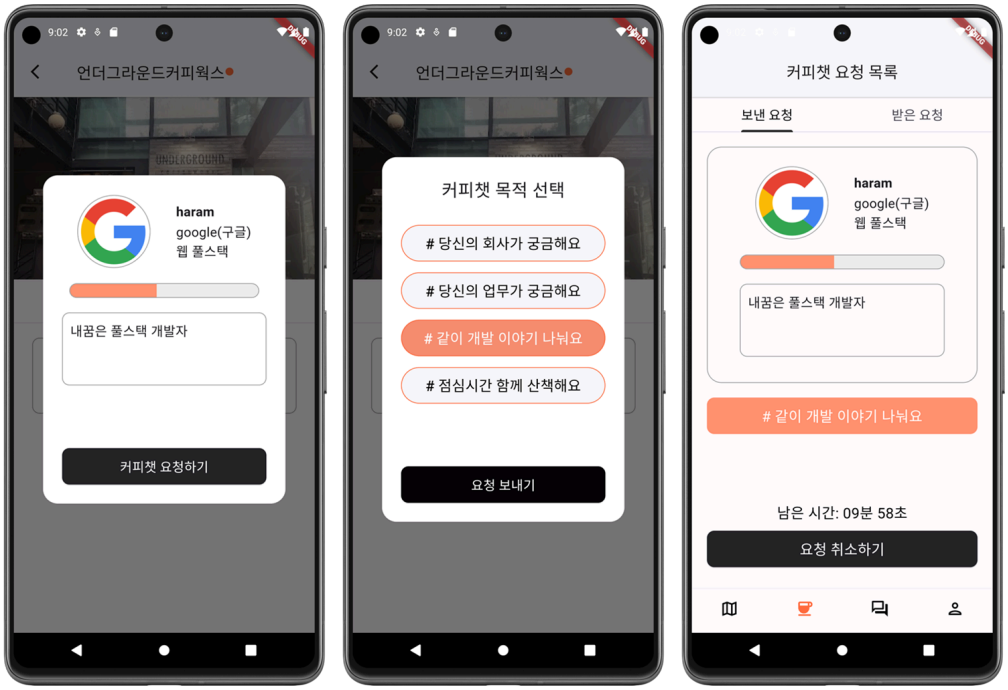
서버에서 웹소켓 연결 끊어짐을 감지하여 서버에서 처리하도록 해 프론트에서는 따로 처리해주지 않았다.

- **백엔드**

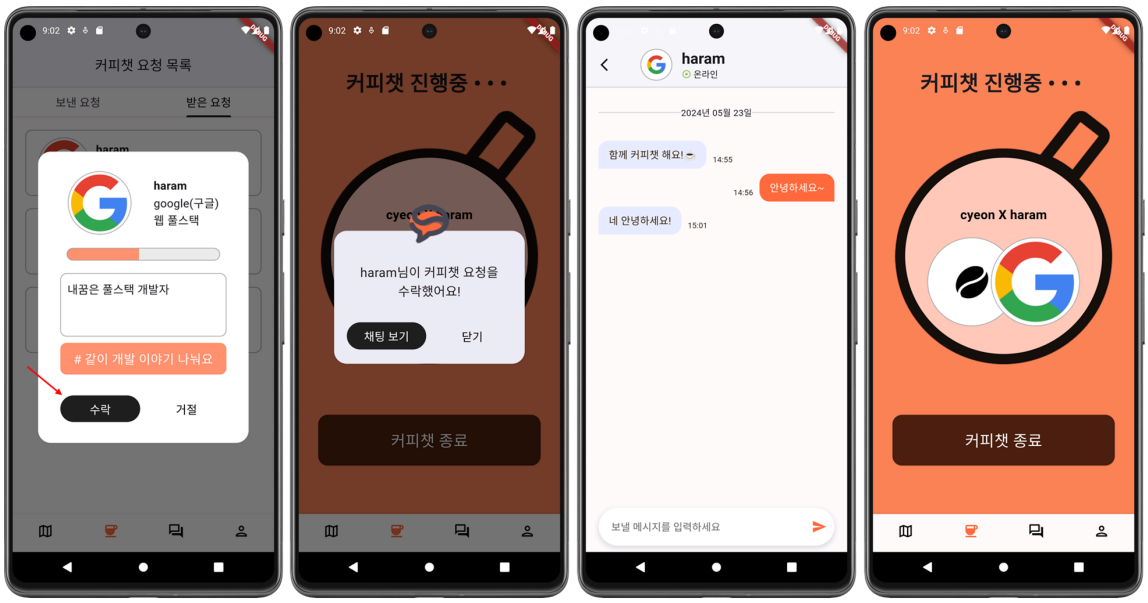
동작 과정 : 초기 지도 화면에 접속 했을 때, HTTP GET 요청으로 사용자 반경 500m 이내의 카페 목록을 request 로 받고, Redis에서 카페별로 할당된 사용자 목록을 조회해 response 로 반환한다. 이후 사용자가 특정 카페를 선택 또는 해지하겠다는 pub 요청을 STOMP 프로토콜로 보내면, 이 요청 정보를 Redis에 저장하고, 해당 카페를 sub 하고 있는 모든 사용자에게 이 사용자가 추가 또는 삭제 되었다는 메세지를 보낸다.

개발 내용 : 우리 서비스는 타 서비스와의 차별점으로 '실시간'을 가져갔기에, 특정 카페 선택시에 해당 카페 상세 페이지를 보고 있는 다른 사용자에게 변동사항이 실시간으로 반영되는 것을 목표로 했다. 이를 구현하기 위해 실시간 데이터 전송이 가능한 websocket과 특정 카페(채널)의 변동사항을 보내기에 적합한 pub/sub 구조를 사용했고, 카페 선택 데이터는 변동이 빈번하고, 휘발성 데이터라는 측면에서 MySQL이 아닌, Redis에 저장했다.
또한 pub 요청시에 연결한 웹소켓 sessionId를 user DB에 저장해서 이후에 해당 사용자가 앱을 종료했을 시에 Redis 에서 해당 사용자가 선택한 카페를 삭제하여, 오프라인을 감지하도록 했다.
사용자가 선택한 카페 정보를 저장하는 Redis 에는 key를 cafeId로 갖고, userId 들이 담긴 set을 value를 저장하도록 구현했다. 이는 특정 카페의 사용자 목록을 조회할 때 cafeId를 기준으로 할당된 사용자 목록을 빠르게 조회할 수 있도록 하기 위함이다.

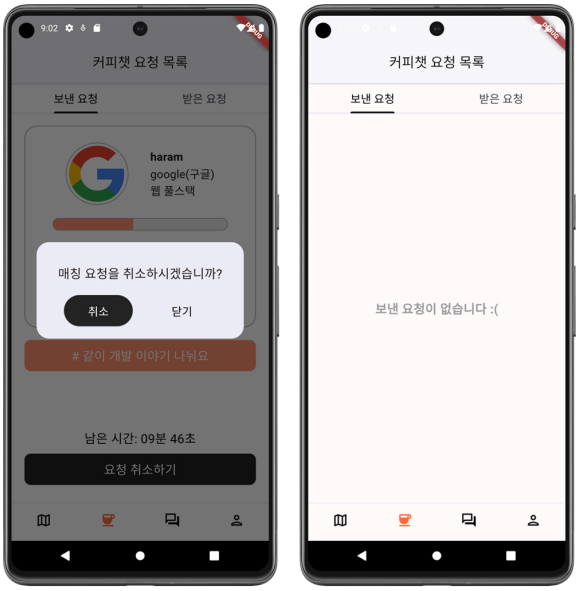
커피챗 매칭 기능 (요청)



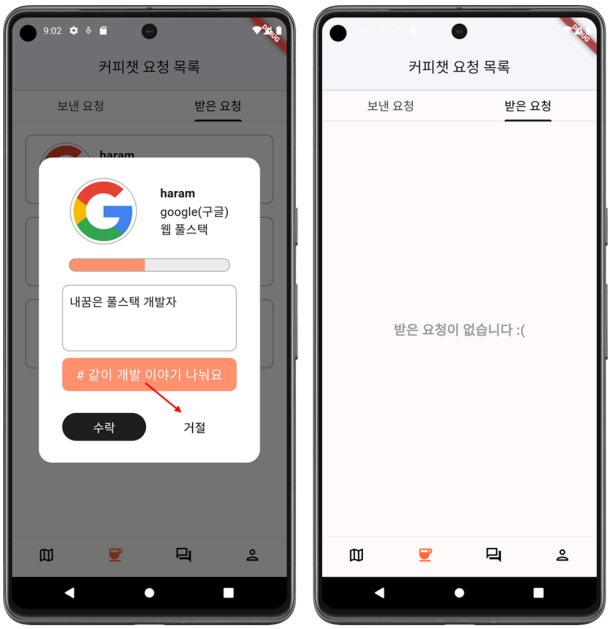
커피챗 매칭 기능 (수락)



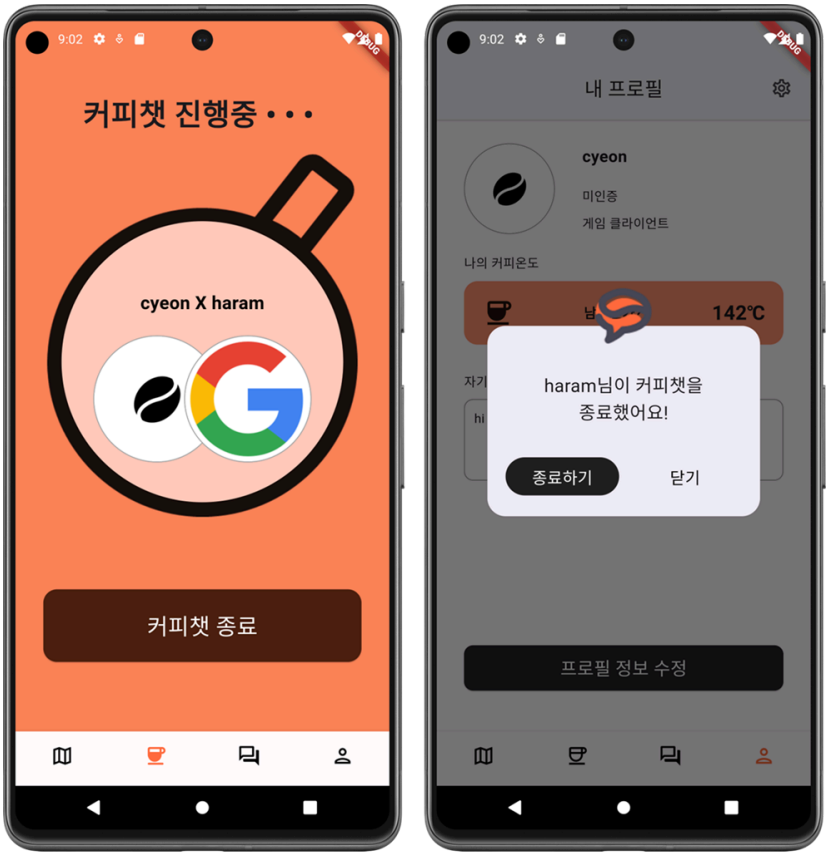
커피챗 매칭 기능 (취소)



커피챗 매칭 기능 (거절)



커피챗 매칭 기능 (종료)



종료 보내기 / 상대가 종료

- **프론트엔드**

요청: 커피챗 목적을 선택했다는 가정 하에 요청을 보낼 수 있다. 커피챗 목적을 선택하지 않았다면 요청 보내기 버튼은 비활성화 되어있고, 클릭 했을 때 “커피챗 목적을 선택하세요”와 같은 Dialog가 뜬다. 목적을 선택한 후 서버에 matchRequest 요청을 보낸다. 이 때 로그인 한 유저의 Id, 상대방의 Id, 그리고 선택한 목적을 index로 넘겨준다.

수락

1. Provider.selectedindex는 하단바의 탭 인덱스이다.
2. 받은 요청 탭으로 이동하여 userItem에 접근한다.
3. 나에게 요청을 보낸 유저의 detail 화면을 보기 위해 ReceivedReqDialog에 접근한다.
4. 수락한다.

두 개의 단계가 중첩된 3단계에서 accept api 요청을 했다. api 요청까지는 잘 이루어졌지만, 총 두개의 페이지와 모달창을 닫으려하니 Looking up a deactivated widget's ancestor is unsafe 라는 에러가 발생했다.

따라서 selectedIndex로 이동하던 부분에서 하단바가 필요 없는 부분은 Navigator.push로 이동하도록 구현했다. 또한 api호출과 Navigator.push를 동시에 구현할 때 위와 같이 조상 위젯이 없다는 에러가 발생했다. 요청화면에서 채팅방 클래스로 이동한 후에 accept request를 보내서 해당 에러를 해결할 수 있었다.

거절: 받은 요청 list에서 상대방의 프로필을 보고 거절하는 기능이다. 거절의 과정에서도 위와 동일한 에러가 발생하였다. 거절의 경우 다른 창으로 이동이 필요하지 않고 페이지의 리로드가 필요한상황이었다. 따라서 콜백함수를 사용해 거절 버튼의 click event를 감지했다. callback 함수인 handelreject에 fetchReceiveList()를 호출하여 받은 요청에서 거절 데이터가 적용되도록 리로드했다.

취소: 자신이 보낸 요청을 취소한다. 요청을 보낸 사용자 입장에서 수행할 수 있는 기능이다. 요청을 취소할 때 handelRequestCancel 함수를 호출해서 api를 호출했다. 요청 취소가 정상적으로 이루어 졌을 때 보낸 요청 정보를 다시 호출했다. 리로드 된 상황에서 보낸 요청은 없기때문에 “보낸 요청이 없습니다” 화면으로 업데이트 된다.

종료: 매칭이 수락된 경우에 수행할 수 있는 기능이다. 상대방과의 매칭을 끝내고 싶을 경우 “매칭 종료하기”를 누르면 커피 콩점 리뷰를 남기고, 매칭을 종료한다. 서로 평점을 남길 수 있도록 상대방이 리뷰를 먼저 남긴 경우에만 matchFinish api를 요청할 수 있도록 구현했다.

- **백엔드**

요청 : 먼저 요청이 유효한지 확인하기 위해 사용자 정보를 검증하도록 했다. 이를 통해 본인에게 요청을 보내는 경우나 유효하지 않은 사용자가 요청을 보내는 경우를 방지한다. 요청이 유효하면 Redis 데이터베이스를 사용하여 매칭 요청 정보를 저장하도록 했고, 이 정보에는 matchId(매칭ID), senderId(발신자ID), receiverId(수신자ID), requestType(요청목적), expirationTime(만료시간), status(매칭 상태)가 포함된다. 매칭 요청이 보내졌으므로 status는 초기에 pending으로 설정했다. 또한, 요청을 보내면 해당 사용자에게 lock을 10분 간 걸어 여러 명에게 커피챗 요청을 보내지 못하도록 설정했다. 이는 Redis의 TTL 기능을 활용하여 자동으로 expire 되도록 설정했다.

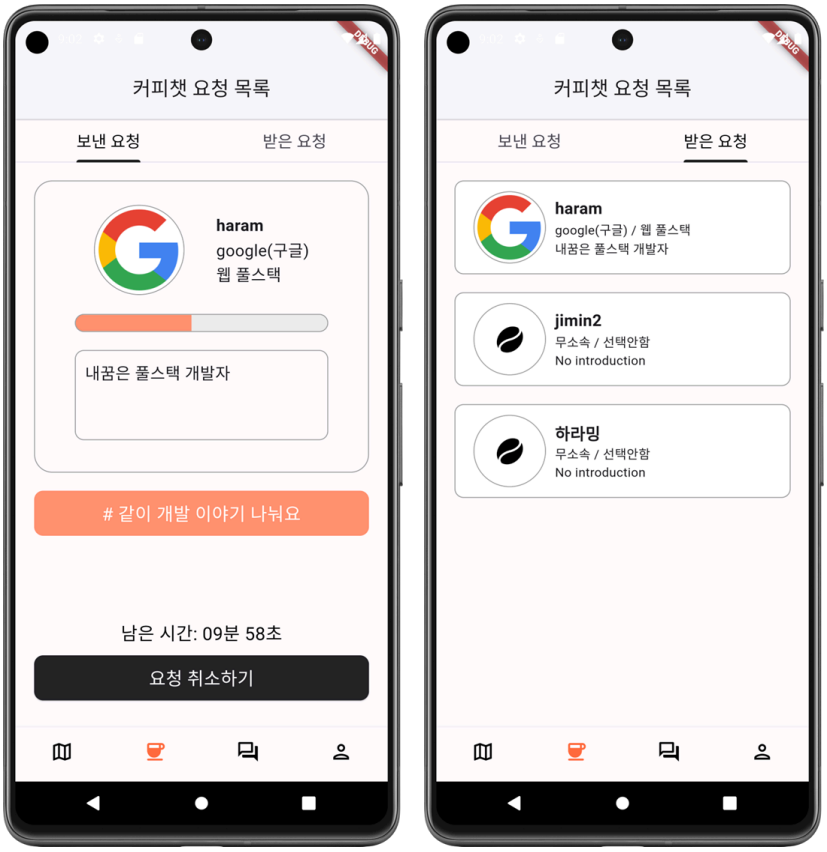
수락 : 요청의 유효성을 확인하고 요청이 만료되지 않았는지 확인한 후에 수락 처리를 하도록 했다. 수락된 요청은 Redis에서 status를 accepted로 업데이트한다. 그리고 수락이 되어 커피챗 매칭이 성사되면 두 사용자 간의 1:1 실시간 채팅방이 생성되도록 했다. 또한, 중복 방지를 위해 요청을 받은 사용자가 이미 다른 사용자에게 보내놓은 요청이 있다면 해당 요청이 삭제되도록 했다.

거절 : 요청의 유효성을 확인하고 요청이 만료되지 않았는지 확인한 후에 거절 처리를 하도록 했다. 거절된 요청은 Redis에서 상태를 status를 declined로 업데이트한다. 요청이 거절되면 요청을 보냈던 사용자는 lock이 풀려 다시 다른 사용자에게 커피챗 요청을 보낼 수 있도록 했다.

취소 : 요청의 유효성을 확인하고 요청이 만료되지 않았는지 확인한 후에 취소 처리를 하도록 했다. 취소된 요청은 Redis에서 상태를 status를 canceled로 업데이트한다. 요청이 취소되면 요청을 보냈던 사용자는 lock이 풀려 다시 다른 사용자에게 커피챗 요청을 보낼 수 있도록 했다.

종료 : 종료 요청을 한 사용자의 유효성을 확인한 후에 종료 처리를 하도록 했다. 종료된 매칭은 Redis에서 상태를 finished로 업데이트하고 매칭 상태를 업데이트한다. 종료 이후에는 커피콩점을 매겨 해당 커피챗의 리뷰를 작성할 수 있다.

커피챗 요청 목록 (보낸/받은)



- **프론트엔드**

UI: 커피챗 요청 목록 페이지에서 보낸 요청과 받은 요청을 탭으로 나누었다. 보낸 요청 탭에서는 사용자가 커피챗 요청을 보낸 상대방의 프로필 정보와 커피챗 요청을 보낼 때 선택한 커피챗 목적을 확인할 수 있으며, 자동으로 취소되기까지 남은 시간 타이머 및 수동 취소 버튼을 넣었다. 받은 요청 탭에는 보낸 요청과 달리 나에게 요청을 보낸 사용자들의 목록을 볼 수 있으며, 각 사용자 선택 시 프로필 상세정보 및 수락/거절 버튼이 있는 모달창을 띄우도록 했다.

기능: 매칭 요청을 하면서 수락, 거절, 종료 등에 데이터를 처리하는 시간이 필요하다. Future와 await을 활용하여 데이터가 로드되지 않는 문제를 해결하였다. api를 호출하는

거의 모든 함수의 자료형은 Future로 선언하였고, await으로 데이터를 기다리도록 처리했다.

보낸 요청 목록과 받은 요청 목록에서 사용자의 상세 정보를 보는 모달이 필요했다. 동일한 모달을 여러개 만들지 않고, 미리 만들어 두었던 UserItem Class를 재활용하여 프로젝트의 응집도를 향상했다. 또한 모달창의 버튼 클릭에 대한 callback 함수를 지정하여 UserItem을 호출하는 곳 마다 다른 event 처리를 할 수 있었기에 low coupling 개념을 적용할 수 있었다. UserItem에서도 ReqDialog, ReceivedReqDialog를 구분하여 독립성을 유지할 수 있도록 했다.

본인이 보낸 요청이나 받은 요청이 없을 때 보낸/받은 요청이 없습니다. 라는 문구를 띄워야 한다. 하지만 empty인 상태에서 빈 리스트를 비교하거나, null로 비교하는 경우 에러가 발생하였다. 따라서 조건문 자체를 아래와 같이 구체화 하여 해결할 수 있었다.

```
snapshot.data == null ||
(snapshot.data!['data'] == null || snapshot.data!['data'].isEmpty))
```

처음에는 보낸 요청의 탭에 접속하는 순간만 생각하여 ReqDialog class 내부에서 타이머를 관리했다. 하지만 탭이 바뀌거나 외부에 나갔다 오면 타이머가 리셋되는 문제가 발생했다. 따라서 보낸 요청의 타이머를 계산할 때 서버에 만료 시간을 요청했다. 만약 앱을 종료 후 다시 접속하는 상황이더라도 타이머가 유지되도록 관리했다.

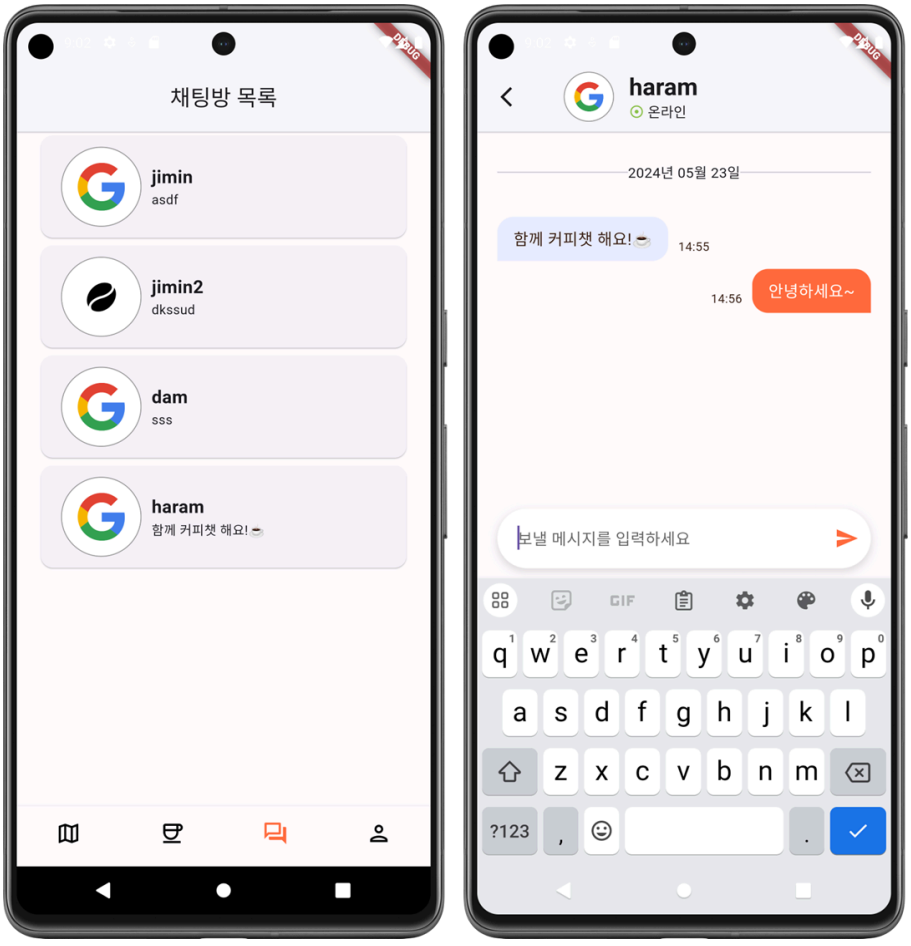
- **백엔드**

보낸 매칭 요청 목록 : 해당 사용자의 ID를 기반으로 Redis에서 정보를 검색하고, 해당 매칭 요청을 수신한 상대방의 정보와 함께 해당 요청의 상태 및 만료 시간 정보를 확인했다. 만약 매칭 요청이 아직 수락 되지 않았고, 만료되지 않은 경우에만 해당 요청이 보여지도록 했다.

받은 매칭 요청 목록 : 해당 사용자의 ID를 기반으로 Redis에서 정보를 검색하고, 해당 매칭 요청을 보낸 상대방의 정보와 함께 해당 요청의 상태 및 만료 시간 정보를 List 구조에 담았다. 만약 매칭 요청이 아직 수락 되지 않았고, 만료되지 않은 경우에만 해당

요청이 보여지도록 했다.

클라이언트에게 Response로 상대방의 프로필 정보(닉네임, 직책, 회사 등), 요청의 상태(대기 중, 수락됨, 거절됨, 취소됨 등), 만료 시간(요청이 유효한 시간 범위 내에 있는지 확인)을 보내서 활용이 용이하도록 했다.

채팅


- 프론트엔드

웹소켓과 STOMP 프로토콜을 통해 실시간 채팅 기능을 제공한다. 채팅 화면에서는 이전 채팅 메시지 목록을 서버로부터 불러와 화면에 표시한다. StompClient 객체를 통해 서버와 웹소켓 통신을 수행한다. 채팅 화면에서는 특정 채팅방의 ID를 얻어와 해당 채팅방을 구독한다. 새로운 메시지가 도착하면 실시간으로 화면에 표시하고, 사용자가 메시지를 전송하면 서버로 전송한다. 새로운 메시지가 도착할 때에는 ScrollController를 사용해 자동으로 맨 아래로 스크롤되도록 구현했다.

유저마다 시차가 다를 수 있다는 점을 고려해서, 서버의 데이터에는 시차를 적용하지 않고 프론트 단에서 각 디바이스의 시차 오프셋을 적용해 화면에 표시하는 것이 좋다고 판단했다. DateTime 클래스의 timeZoneOffset 속성을 이용해 각 디바이스의 시차를 구해올 수 있었다. 서버에서 받아온 시간 스트링을 DateTime 객체로 변환한 후 시차를 적용하고, 다시 약속된 형식의 스트링으로 포맷팅했다.

커피챗 매칭 수락 시 바로 채팅 화면으로 이동하는 경우 채팅방 ID를 알 수 없다는 문제가 있었다. 이를 해결하기 위해, 수락한 match ID를 이용해 채팅방 ID를 얻어오는 방식을 사용했다.

- **백엔드**

  서버는 'pub'으로 시작하는 uri로 들어오는 메시지를 '@MessageMapping' 어노테이션으로 받아 처리하며 처리된 결과 및 메시지는 'sub'으로 시작하는 uri로 들어오는 메시지를 통해 uri를 구독한 사용자들에게 send해주는 방식으로 동작한다.

  메시지 브로커는 InMemory 방식으로, 동시 접속자수가 6만명이 넘는 상황에는 문제가 될 수 있겠으나 프로젝트를 진행하고 시연하는 과정에는 무리가 없을것으로 판단했고, 현재 프로젝트 기간 안에는 InMemory 메시지 브로커를 사용하기로 결정했다.

노쇼 문제 방지를 위한 커피챗 리뷰 기능

- **프론트엔드**

  매칭 진행중인 A와 B 사용자가 존재한다. 두 사용자 중 한 명의 사용자가 커피챗 종료 요청을 보내면 A, B 모두 커피챗 리뷰의 과정으로 넘어간다. 상대 닉네임과 빈 커피콩이 나타나고, 원하는 점수에 해당하는 콩을 선택하면 화면 하단에 '보통이에요.', “만족스러워요.', '완벽해요!' 와 같은 문구가 뜬다. 제출하기 버튼을 클릭하면 상대방의 평점에 반영된다.

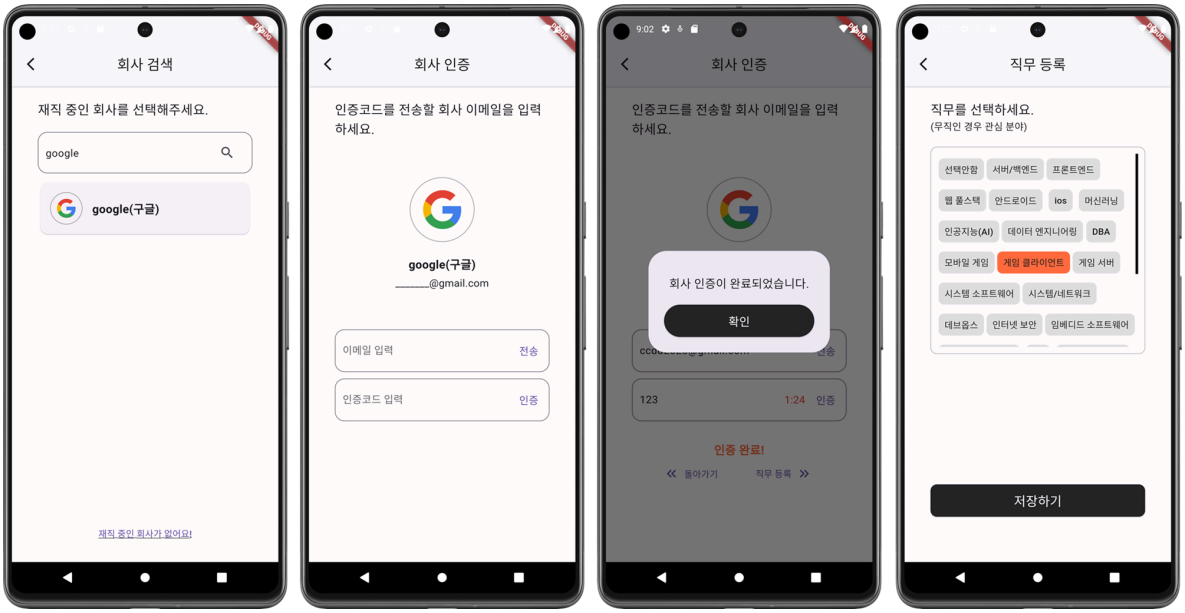
  커피챗 매칭이 성사된 상태에서 장소에 나오지 않거나, 부적절한 이유로 매칭을 종료하는 사례를 방지한다. 매칭을 수락하는 과정에서 사용자의 커피 온도를 확인하고 거절하여 원치않는 상황을 피하는 효과를 기대한다.

- **백엔드**

  동작 과정 : 커피챗 종료 이후 사용자가 리뷰를 남기면 이를 review DB에 저장하고, rating 값 (1~5) 을 평가를 받은 사용자의 커피온도에 반영한다.

개발 내용 : 반영 계산식은 다음과 같이 작성했는데, “기존 평점 + (새로운 평점 - 기준 평점) * 랜덤 반영 비율” 여기서 '기준 평점'을 두어 3점 이하의 평가를 받은 경우 무조건 마이너스로 적용되도록 했고, 4점 이상의 평가를 받은 경우엔 플러스로 반영되도록 했다. 이는 3점 이하의 평점을 준데에는 무조건 불만사항이 있어서라 판단에서 기인한 것이다. 그리고 '비율' 값을 (0.3~1.0) 사이의 랜덤 숫자로 반영하여 반영되는 것에 있어서 규칙성이 없도록 구현했다.

회사 인증



- **프론트**

서버에 저장된 회사 정보를 검색하여 회사 이름, 로고, 도메인 주소를 얻는다. 사용자가 인증 코드 메일을 받기 위해 작성한 이메일 도메인이 서버에 저장된 해당 회사의 도메인과 일치할 때에만 회사 인증 요청이 보내진다.

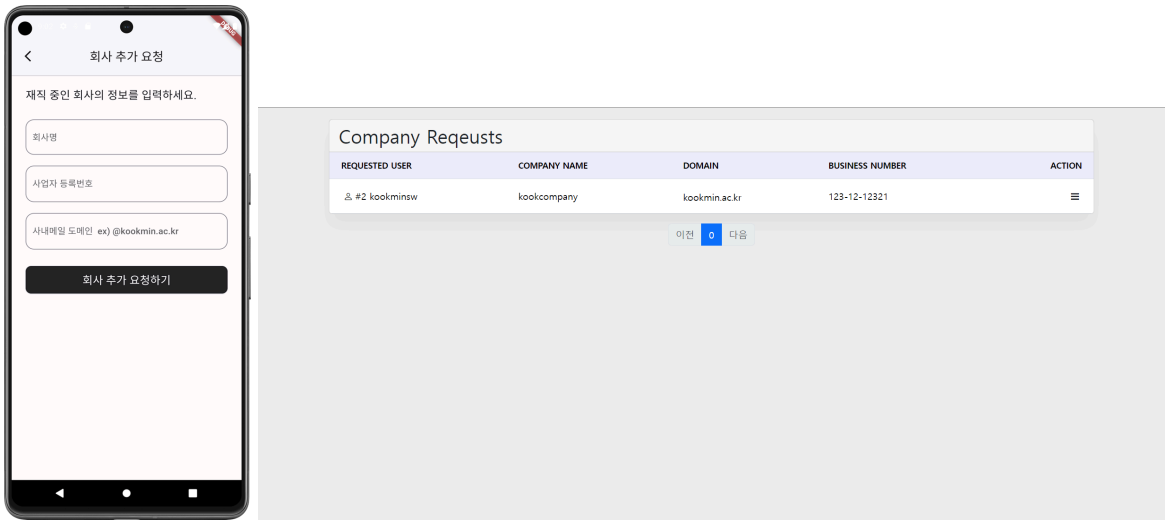
서버에 정의된 인증코드 제한시간인 3분에 맞추어, 텍스트 필드 옆에 3분 타이머를 배치했다. 이 타이머는 이메일 발송 요청이 성공하고 나면 작동한다. 또한 인증 요청이

성공하고 나면 직무 등록 페이지로 바로 이동하는 버튼을 제공하여 사용자 편의성을 높였다.

- **백**

company 테이블에는 회사의 이름과 도메인, 로고 파일이 저장되는데, 이때 파일은 AWS S3에 저장되는것을 StorageService에서 관리한다.
파일의 메타 데이터는 서버 DB에 실제 이름(originName)과, S3에 저장되는 이름(storedName = UUID+originName)을 담고있고 UploadFile 테이블에 저장되지만, 파일의 실제 데이터는 AWS S3에 저장되어있는 형식이다.
이런 방식은 서버의 공간 복잡도를 크게 줄여줄 수 있고, 프론트엔드에 회사 로고 파일을 보내줄 때 AWS S3에 저장되어있는 위치인 String url을 전달해주면 되기때문에 프론트엔드와 백엔드 간 통신을 할 때에도 트래픽을 줄이는 효과를 볼 수 있다.

새 회사 등록 요청 (dashboard)



- **프론트**

사용자가 재직 중인 회사가 서버에 등록되어 있지 않은 경우, 별도의 회사 등록 화면을

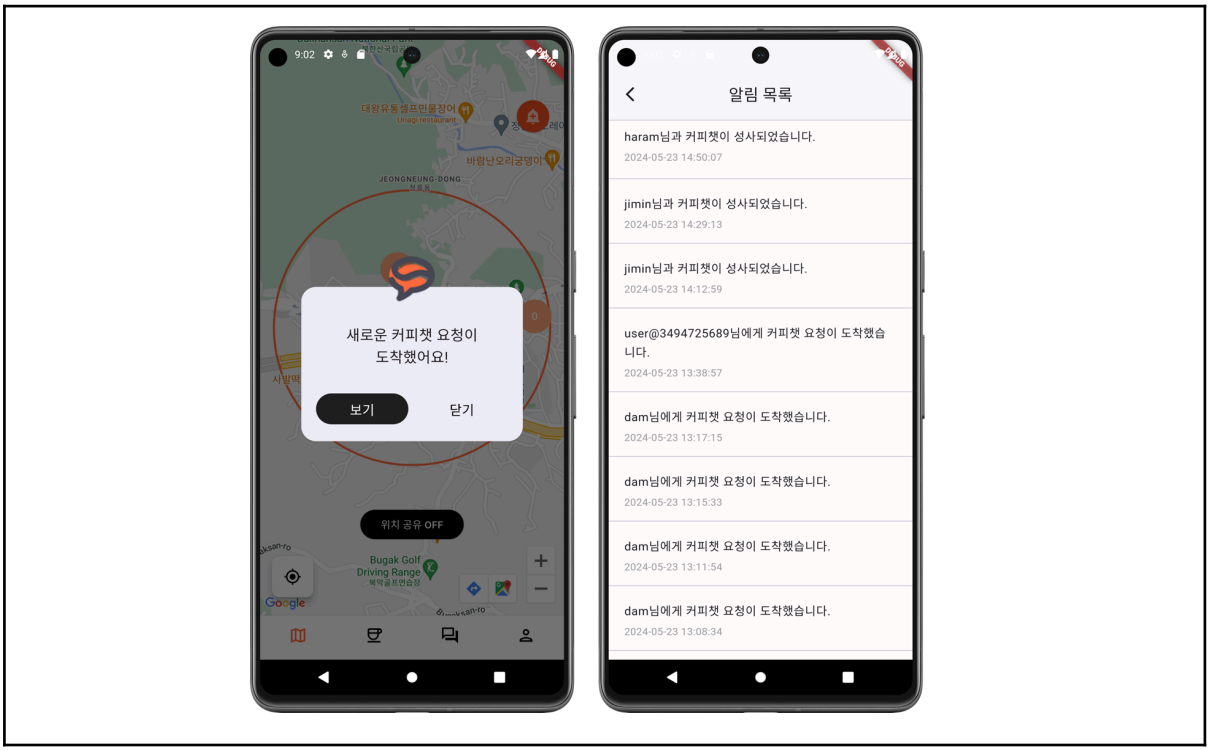
통해 회사 정보를 입력하여 관리자에게 해당 회사 등록 승인을 요청한다.

- **백엔드**

회사 인증을 하기 위해서는 무조건 서버 DB에 저장되어있는 회사의 도메인으로 이메일 인증을 거쳐야하기 때문에, 자체 이메일 인증 시스템에 등록되지 않은 회사인 경우 관리자가 update 해주는 과정이 필요하다. 따라서 회사 등록 승인을 요청하는 api를 만들어 어플리케이션 사용자로부터 회사 등록 요청을 받을 수 있도록 했다.

또 관리자가 회사 등록 요청들을 확인하기 용이하도록 돕기 위해서 관리자 페이지(/dashboard)를 추가했다. 타임리프 템플릿 엔진을 활용한 스프링 MVC 페이지이다.
관리자가 확인한 경우 요청이 적용되었다는 의미에서 데이터를 삭제하는 api까지 적용했다. 관리자는 /dashboard에서 회사 등록 요청을 확인한 후, 필요한 경우 새로운 company 데이터를 DB에 저장하고 이후 필요 없는 회사 등록 요청 데이터는 삭제한다.

푸시 알림 기능



- **프론트엔드**

FCM(Firebase Cloud Messaging)을 이용해 사용자가 다양한 이벤트에 대한 인앱 알림 및 푸시 알림을 실시간으로 받을 수 있도록 구현했다. 이를 위해 먼저 Firebase를 초기화하고 디바이스 토큰을 발급받는다. 이 디바이스 토큰은 로그인 또는 카카오 로그인 시 서버로 함께 전송되어, 해당 계정에 로그인한 기기로 알림이 전달될 수 있도록 했다.

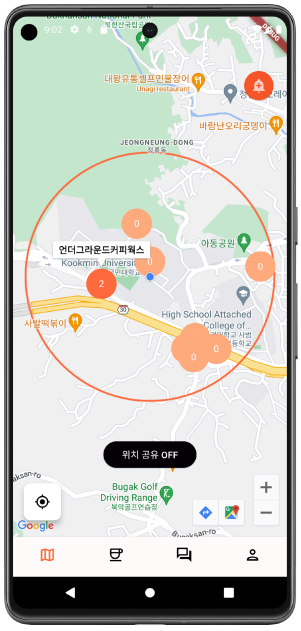
data 부분은 인앱 알림과 알림기록 저장에 사용했다. 그리고 푸시 알림으로 내용을 확인할 수 있도록 notification 부분을 사용했다. 인앱 알림의 경우, 앱이 포그라운드 상태일 때 메시지가 도착하면 data 부분의 title을 확인하여 커피챗 요청, 수락, 거절, 종료 중 어떤 알림인지 구별하고, 이에 맞는 위젯을 반환하도록 했다.

알림 목록 화면을 구현하기 위해서는 앱이 재시작되더라도 알림 로그를 유지할 필요가 있었다. 이를 위해 알림 로그 파일을 getApplicationDocumentsDirectory()를 통해 얻은 앱 전용 저장 공간에 저장하여, 사용자가 언제든지 알림 내역을 확인할 수 있도록 했다.

- 백엔드

Firebase Admin SDK를 활용하여 FCM 서버와 통신하고 인증하는 로직을 구현했다. 푸시 알림을 받을 대상 디바이스 토큰, 제목 및 메시지 내용과 같은 매개변수를 사용하여 푸시 메시지를 구성하고, 필요한 헤더와 인증 토큰을 준비하여 FCM API 엔드포인트에 HTTP POST 요청을 보내 메시지를 전달했다.

또한, 인증에 필요한 액세스 토큰을 얻기 위해 서비스 계정의 개인 키를 사용하여 구글 클라우드 플랫폼에 인증하고 액세스 토큰을 검색했다.

Google Map API (프론트)


휴대폰에 Google Map API 연동 및 500미터 반경 표시 & 카페 표시

Flutter의 geolocator 패키지를 활용하여 사용자의 현재 위치를 받아온다. 이를 바탕으로 500미터 반경 내에 위치한 카페를 Google Map API를 통해 표시하는 기능을

구현하였다. 이를 위해 Android 휴대폰과 에뮬레이터를 사용하여 GPS가 잘 작동하는지 확인하였다.

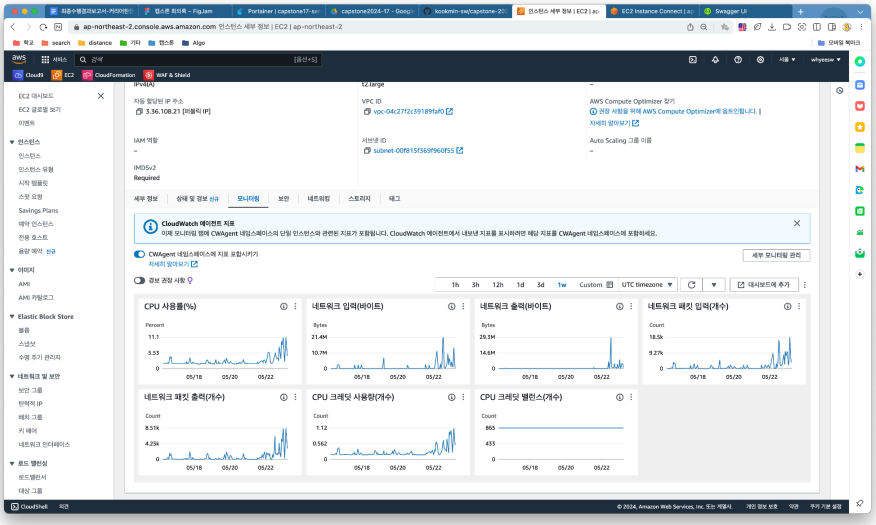
에뮬레이터는 GPS를 지원하지 않기에 에뮬레이터 내부에서 GPS를 고정, Android 휴대폰을 연결하여 테스트하는 두 가지 방법을 통해 GPS가 잘 작동하는지 확인하였다.

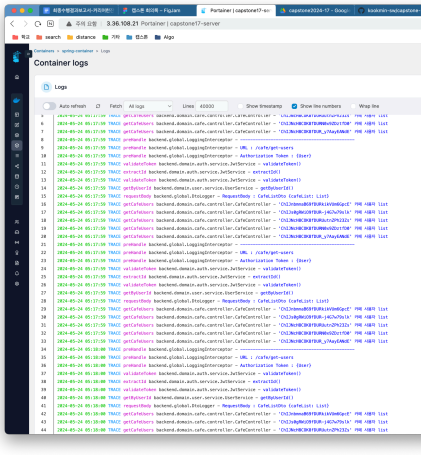
초기에는 사용자가 이동할 때마다 지도를 Reload했지만, API 호출이 많아 서버에 과부하를 줄 것이라 예상했다. 따라서 특정 시간(5분)마다 사용자의 이동 거리를 계산하여 100미터 이상 이동했을 시에만 Reload하도록 수정했다.

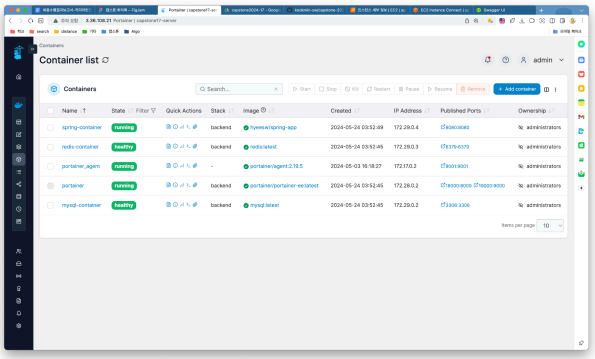
주변 카페를 검색하는 과정에서 Text Search 방식에서 Nearby Search 방식으로 변경하여 성능을 향상시켰다. 데이터가 업데이트되기 이전에 뜨지 않던 정보도 Nearby Search 방식에서는 최신화된 데이터를 제공하였기에 더 많은 정보를 가져올 수 있었다. 예를 들어, 정릉역 기준으로 카페 수가 2개에서 5개 이상으로 증가했다. 또한, Text Search로 검색 범위를 500m로 설정했을 때는 정확성이 떨어져 약 1km 떨어진 카페의 정보까지 인식되는 문제가 있었다. 하지만 Nearby Search에서는 500m 내의 정보 필터링이 정확하여 앞서 처리했던 이중 필터링을 피할 수 있었다.

카페들을 지도에 원형의 마커로 직접 정의하여 표시했고, 각 마커별로 click event 옵션을 주어 detail 화면으로 이동하면서 focus가 해당 카페로 이동하도록 지정했다.

서버 배포 & 자동화 & 로그 (백엔드)







백엔드 CI/CD 동작 과정 : 백엔드 측에서 main 브랜치에 새로운 커밋을 올리면, Github Actions 의 workflow 가 동작한다. 이때 최신 코드로 새 Docker Image 를 생성해 Docker Hub 에 push 하고 Portainer 의 webhook link 로 컨테이너를 새로 생성하라는 트리거를 보낸다. Portainer 에서 해당 컨테이너에 필요한 새 이미지를 다운 받고, EC2에 컨테이너 재빌드를 요청해 서버를 재배포한다.

개발 내용 : AWS의 EC2 t2.micro 는 제공되는 램이 1GB밖에 되지 않아 Spring-Boot 가 동작하다가 멈추는 문제가 있어서 8GB 램을 지원하는 t2.large 인스턴스를 사용해

백엔드 서버를 배포했고, Portainer를 사용해 GUI 환경에서의 더욱 용이한 서버 관리 환경을 구축했다.

또한 Docker-Compose 를 이용하여 Spring-Boot, MySQL, Redis 각각을 컨테이너로 생성해 service layer 로 묶어 각 컨테이너가 동일한 네트워크 안에서 통신할 수 있도록 했다.

Spring-Boot 에서 기본으로 제공하는 로그 대신, LogBack 라이브러리로 로그 환경을 커스텀해서 요청이 온 endpoint, request data, 거치는 Service layer의 메소드를 출력해서 에러 발생시에 더욱 빠른 디버깅이 가능하도록 환경을 셋업했다.

2. .2 시스템 기능 요구사항

☑ **일반 로그인 및 소셜 로그인**

설명 : 회원가입이 완료된 계정으로 로그인, 혹은 카카오를 통한 로그인이 가능해야 한다.

입력 : loginId, password, accessToken(카카오), deviceToken(=FCM 토큰)

출력 : authToken, userUUID

조건 : accessToken은 카카오 서버에서 제공하는 토큰이다. deviceToken은 각 기기의 고유한 값이다.

☑ **마이페이지**

설명 : 유저의 로그인 여부를 판단하고, 로그인한 유저의 정보 열람 및 수정이 가능하다.

입력 : authToken, nickname, company, position, introduction, rating

출력 : isLogined, userId, nickname, company, position, introduction, rating

조건 : 프론트 단에서 유저 정보는 ChangeNotifierProvider로 생성된 user profile 모델로 관리된다.

☑ **카페 선택**

설명 : 사용자 반경 500m 이내의 특정 카페를 자신의 위치로 지정할 수 있어야하고, 지정시 해당 카페 구독자에게 broadcast 한다.

입력 : type(add/delete), userId, cafeId

출력 : type(add/delete), userId, cafeId, 사용자 정보

조건 : type은 add 또는 delete 만 입력되어야한다. 존재하는 userId를 입력해야한다.

☑ **커피챗 매칭 기능 (요청/취소/수락/거절/종료)**

설명 : 사용자가 커피챗 매칭을 요청하거나 취소할 수 있어야 하며, 다른 사용자가 요청을 수락하거나 거절할 수 있어야 한다. 또한, 커피챗이 종료될 때 이를 처리해야 한다.

입력 : matchId, senderId, receiverId, status(pending/accepted/canceled/declined/finished)

출력 : matchId, status(pending/accepted/canceled/declined/finished), senderId, receiverId

조건 : 요청 한 순간부터는 status가 pending/accepted/canceled/declined/finished 중 하나여야 한다. 그리고, senderId와 receiverId는 존재하는 사용자여야 한다. 또한, 요청 상태 변경은 유효한 전환만 허용된다. (요청 상태에서 취소 또는 수락/거절, 수락된 상태에서만 종료 가능)

☑ **커피챗 요청 목록 (보낸/받은)**

설명 : 사용자가 보낸 커피챗 요청 목록과 받은 요청 목록을 조회할 수 있어야 한다.

입력 : userId

출력 : requests: [{matchId, senderId, receiverId, status, expirationTime}]

조건 : userId는 존재하는 사용자여야 하며 expirationTime이 지난 후의 요청은 리스트에서 보이지 않는다.

☑ **채팅**

설명 : 웹소켓과 STOMP 프로토콜을 통해 실시간 채팅 기능을 제공한다.

입력 : nickname, logoUrl, chatroomId, matchId

출력 : chatroomList, chatList

조건 : chatroomId를 알 수 없는 경우 matchId를 통해 서버에 chatroomId를 요청한다.

☑ **커피챗 리뷰 및 커피온도**

설명 : 커피챗 종료 이후 사용자가 리뷰를 남기면 이를 review DB에 저장하고, rating 값 (1~5) 을 평가를 받은 사용자의 커피온도에 반영한다.

입력 : matchId, reviewerId, revieweeId, rating, comment

출력 : matchId, reviewerId, revieweeId, rating, comment

조건 : 평점은 1~5점만 입력해야한다. 존재하는 userId를 입력해야한다.

☑ **회사 인증 기능**

설명 : 사용자가 자신이 재직 중인 회사를 자신의 프로필에 등록하기 위해서는 회사 이메일을 통해 인증 과정을 거쳐야 한다.

입력 : email, authCode

출력 : result

조건 : 입력된 이메일의 도메인이 해당 회사의 도메인과 일치해야 한다.

☑ **회사 등록 요청 (dashboard)**

설명 : 사용자는 별도의 회사 등록 화면을 통해 회사 등록을 요청하고, 관리자는 관리자 페이지에서 회사 등록을 승인한다.

입력 : companyName, bno, domain

출력 : companyName, bno, domain

조건 : 관리자가 직접 승인해야만 서버의 회사 db에 등록된다.

☑ **푸시알림**

설명 : FCM을 이용해 사용자는 다양한 이벤트에 대한 인앱 알림 및 푸시 알림을 실시간으로 받을 수 있다.

입력 : message

출력 : dialog(요청, 수락, 거절, 종료), push

조건 : message에는 data 부분과 notification 부분이 존재한다.

☑ **지도API**

설명 : google map 지도를 나타낸다.

입력 : googleMapKey

출력 : 지도 화면

조건 : 내 위치를 기반으로 반경 500미터의 환경에서만 카페가 검색된다.

☑ **서버 배포 & 자동화 & 로그**

설명 : public ip 를 사용하는 서버를 배포하고, 백엔드 main 브랜치에 변동사항이 생기면, 서버가 재배포되도록 자동화되어야한다. 서버 로그는 백엔드 개발자들에게 가독성 높고, 에러를 빠르게 고칠 수 있도록 커스텀한다.

입력 : main 브랜치 변동사항

출력 : 재배포된 서버

조건 : AWS EC2에 배포된 서버가 있어야한다.

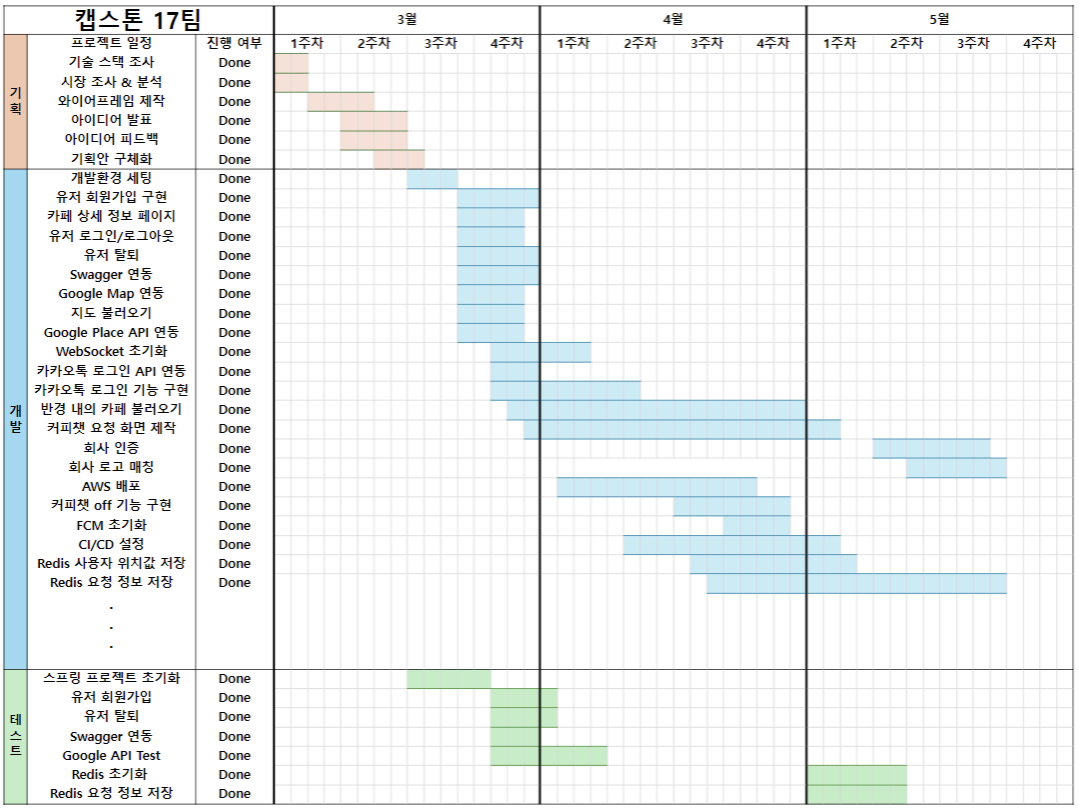
☐ **회사 필터링 (구현 못함)**

설명 : 자신의 반경 500m 의 카페에 속한 사용자 목록에 대해서 사용자들의 회사 및 직무를 필터링해 검색할 수 있다.

입력 : 필터링 값

출력 : 필터링에 해당하는 사용자 목록

조건 : 내 위치가 켜져있어야한다.



2. .3 시스템 비기능(품질) 요구사항

a. 회원관리를 위한 일반 로그인 & 소셜 로그인 기능

- ☑ 보안: 사용자의 로그인 정보와 토큰은 안전하게 암호화되어 저장되고 전송되어야 한다. JWT 토큰의 유효성을 주기적으로 검증하여 만료된 토큰은 즉시 폐기해야 한다.
- ☑ 성능: 로그인 시도에 대한 응답 시간은 2초 이내여야 한다.
- ☐ 확장성: 동시에 1000명 이상의 사용자가 로그인할 수 있어야 한다. -> 서버 자원의 한계로 EC2 인스턴스의 CPU 및 메모리 자원이 동시에 많은 요청을 처리하기에 충분하지 않았을 것이다. 특히, 피크 타임에 많은 사용자가 동시에 로그인하려고 할 때 서버 자원이 부족하여 요청이 지연되거나 실패할 수 있다.

b. 마이페이지 (직무/자기소개/회사 .. 수정)

- ☑ 성능: 마이페이지 정보 로드와 수정 작업은 1초 이내에 완료되어야 한다.
- ☑ 유지보수성: 사용자 정보 수정 시 변경 사항이 실시간으로 반영되어야 한다.
- ☑ 사용성: 사용자 인터페이스는 직관적이고 사용하기 쉬워야 한다.

c. 내 위치 지정(카페 선택), 위치 공유 기능

- ☑ 보안: 사용자의 위치 정보는 안전하게 보호되어야 하며, 사용자의 동의 없이 공유되지 않아야 한다.
- ☑ 정확성: 사용자의 위치 정보는 500미터 반경 내에서 정확하게 반영되어야 한다.
- ☑ 성능: 위치 정보 업데이트는 1초 이내에 이루어져야 한다.

d. 커피챗 매칭 기능 (요청, 취소, 수락, 거절, 종료 등)

- ☑ 신뢰성: 매칭 요청, 수락, 거절 등의 상태 변경은 데이터 손실 없이 정확하게 처리되어야 한다.
- ☑ 성능: 매칭 요청 및 응답 시간은 1초 이내여야 한다.
- ☐ 확장성: 하루 최대 10,000건 이상의 매칭 요청을 처리할 수 있어야 한다. -> 서버 자원의 한계로 EC2 인스턴스의 CPU 및 메모리 자원이 동시에 많은 요청을 처리하기에 충분하지 않았을 것이다. 특히, 피크 타임에 많은 사용자가 동시에 로그인하려고 할 때 서버 자원이 부족하여 요청이 지연되거나 실패할 수 있다.

e. 커피챗 요청 목록 (보낸 / 받은)

- ☑ 사용성: 요청 목록은 직관적이고 쉽게 탐색할 수 있어야 한다.
- ☑ 성능: 요청 목록의 로드 시간은 1초 이내여야 한다.
- ☑ 일관성: 데이터의 일관성을 유지하고, 요청 목록은 항상 최신 상태를 반영해야 한다.

f. 채팅

- ☑ 성능: 메시지 전송 및 수신 시간은 1초 이내여야 한다.
- ☑ 신뢰성: 채팅 메시지는 손실 없이 정확하게 전송되어야 한다.
- ☐ 확장성 : 채팅 시스템은 동시에 10,000명 이상의 사용자가 접속할 때도 안정적으로 동작해야 하며, 초당 1,000개의 메시지를 처리할 수 있어야 합니다. -> 서버 자원의 한계로 EC2 인스턴스의 CPU 및 메모리 자원이 동시에 많은 요청을 처리하기에 충분하지 않았을 것이다. 특히, 피크 타임에 많은 사용자가 동시에 로그인하려고 할 때 서버 자원이 부족하여 요청이 지연되거나 실패할 수 있다.

g. 노쇼 문제 방지를 위한 커피챗 리뷰 기능

- ☑ 신뢰성: 리뷰 데이터는 정확하게 저장되어야 하며, 리뷰 반영 시 오류가 없어야 한다.
- ☑ 사용성: 리뷰 작성 인터페이스는 간단하고 직관적이어야 한다.
- ☑ 성능: 리뷰 제출 후 반영 시간은 1초 이내여야 한다.

h. 회사 인증

- ☑ 보안: 회사 인증 과정에서 사용자의 이메일과 도메인 정보는 안전하게 보호되어야 한다.
- ☑ 정확성: 인증 요청은 정확하게 처리되어야 하며, 잘못된 도메인으로 인증되지 않아야 한다.
- ☑ 성능: 인증 요청에 대한 응답 시간은 2초 이내여야 한다.

i. 새 회사 등록 요청 (dashboard)

- ☑ 유지보수성: 회사 등록 요청 및 관리 기능은 쉽게 유지보수할 수 있어야 한다.
- ☑ 성능: 회사 등록 요청 처리는 1초 이내에 완료되어야 한다.
- ☐ 확장성: 하루 최대 1,000건의 회사 등록 요청을 처리할 수 있어야 한다. -> 로드 밸런서 설정이 최적화되지 않아 특정 서버에 부하가 집중되었을 수 있다. 적절한 부하 분산이 이루어지지 않으면 일부 서버가 과부하 상태가 되어 전체 시스템 성능에 영향을 미칠 수 있다.

j. 푸시알림 기능

- ☑ 성능: 푸시 알림은 이벤트 발생 후 1초 이내에 사용자에게 전달되어야 한다.
- ☑ 신뢰성: 모든 푸시 알림은 정확하게 전송되고 수신되어야 한다.
- ☐ 확장성: 하루 최대 100,000건의 푸시 알림을 처리할 수 있어야 한다. -> 서버 자원의 한계로 EC2 인스턴스의 CPU 및 메모리 자원이 동시에 많은 요청을 처리하기에 충분하지 않았을 것이다. 특히, 피크 타임에 많은 사용자가 동시에 로그인하려고 할 때 서버 자원이 부족하여 요청이 지연되거나 실패할 수 있다.

k. 지도 API

- ☑ 성능: 사용자의 현재 위치와 주변 카페 표시 시간은 2초 이내여야 한다.
- ☑ 정확성: 지도에 표시되는 위치와 카페 정보는 정확해야 한다.
- ☑ 사용성: 지도 인터페이스는 직관적이고 쉽게 사용할 수 있어야 한다.

l. 서버 배포 & 자동화 & 로그

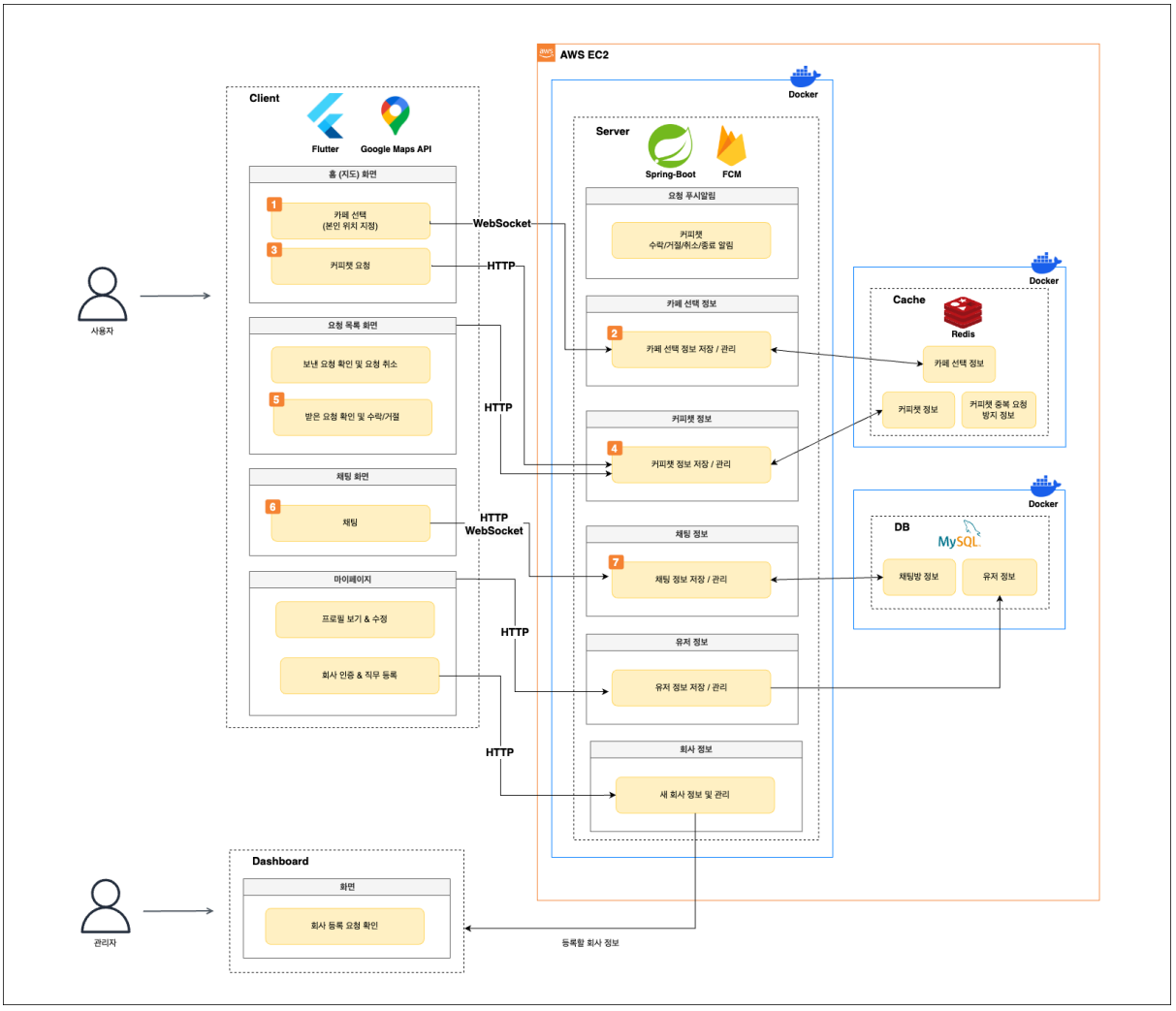
- ☑ 신뢰성: 배포 과정은 오류 없이 안정적으로 수행되어야 한다.
- ☑ 자동화: CI/CD 파이프라인은 완전 자동화되어야 하며, 수동 개입이 필요 없어야 한다.
- ☑ 모니터링: 로그는 실시간으로 모니터링 가능해야 한다.

전반적인 미달성 NFR 원인 분석

- '확장성' 관련 NFR은 부하 테스트를 통해 시스템이 최대 부하를 처리할 수 있는 능력을 평가하지 못했기 때문에, 동시에 많은 사용자가 접속했을 때의 시스템 성능을 보장할 수 없는 상황이다. 또한 현재 어플리케이션이 Monolithic 구조이기 때문에 특정 기능에 병목 현상이 전체 시스템 성능에 영향을 미칠 수 있어 '확장성' 관련해서는 NFR 을 달성하지 못하였다.

2. .4 시스템 구조 및 설계도

2.2.4.1 시스템 아키텍처



클라이언트 (Client)

- **플러터 (Flutter)**와 **구글 맵 API (Google Maps API)**를 사용하여 모바일 애플리케이션을 개발한다.
- 화면 구성 요소 :
    1. **홈 화면 :** 카페 선택, 위치 설정, 카페 요청
    2. **요청 목록 화면 :** 요청 확인 및 취소, 받은 요청 확인 및 수락/거절
    3. **채팅 화면:** 채팅 기능
    4. **마이페이지 :** 프로필 보기 및 수정, 회사 인증 등

서버 (Server)

- **Spring Boot**와 **FCM (Firebase Cloud Messaging)**을 사용하여 다음과 같이 서버 애플리케이션을 구성한다.
    1. **요청 수신 처리 :** 카페 예약/취소/수락/거절 모델 처리
    2. **카페 선택 정보 저장/관리 :** 카페 선택 정보를 Redis 캐시에 저장
    3. **카페 요청 처리 :** 사용자 요청을 받아 처리하고 필요한 정보를 관리
    4. **카페 정보 저장/관리 :** 카페 정보, 요청 정보를 MySQL DB에 저장 및 관리
    5. **요청 확인 및 수락/거절 :** 사용자가 받은 요청을 확인하고 수락/거절하는 기능
    6. **채팅 정보 :** 사용자 간 채팅 정보를 관리

캐시 (Cache)

- Redis를 사용하여 빠른 데이터 접근을 위한 캐시 서버를 구성한다.
- 주요 저장 데이터:
    1. 카페 선택 정보
    2. 카페 요청 정보와 관련된 다양한 상태 정보

데이터베이스 (DB)

- **MySQL**을 사용하여 영구적인 데이터 저장소를 구성한다.
- 주요 저장 데이터:
    1. 카페 정보
    2. 요청 정보
    3. 유저 정보

관리자 대시보드 (Dashboard)

- 관리자가 접근하여 시스템을 관리할 수 있는 대시보드를 구성한다.
- 주요 기능:
    1. 회사 등록 요청 확인
    2. 등록된 회사 정보 관리

통신 방식

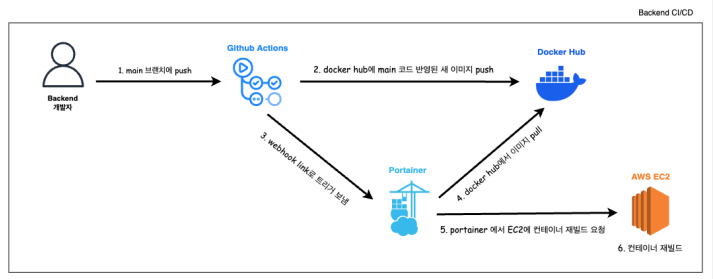
- **HTTP**와 **WebSocket**을 통해 클라이언트와 서버 간의 데이터 통신을 수행한다.
- WebSocket은 실시간 통신을 필요로 하는 기능에서 사용된다.

배포 환경

- **AWS EC2** 인스턴스에서 **Docker 컨테이너**를 이용하여 서버, 캐시, 데이터베이스를 배포한다.

계획서에서 제시했던 구조도와 달리 최종 구조도에는 관리자 대시보드를 추가하여 관리자 기능을 강화했다. 이를 통해 관리자는 회사 등록 요청을 확인하고 등록된 회사 정보를 관리할 수 있게된다. 이 외에도 기능 설명과 플로우 번호 추가를 통해 시스템의 흐름을 파악할 수 있도록 변경했다.

2.2.4.2 CI/CD 파이프라인 다이어그램



1. **개발자가 main 브랜치에 코드 push**
   - 백엔드에서 새로운 코드 변경 사항을 GitHub 저장소의 main 브랜치에 push한다. 이 작업은 새로운 기능 추가, 버그 수정, 성능 개선 등을 포함한다.
2. **GitHub Actions가 새로운 이미지를 Docker Hub에 push**
   - GitHub Actions는 main 브랜치에 푸시된 변경 사항을 감지하고, CI/CD 워크플로우를 트리거한다.
   - 이 워크플로우는 자동으로 새로운 Docker 이미지를 빌드하고, 이 이미지를 Docker Hub에 푸시한다.
3. **Webhook 링크로 트리거 신호 전송**

- Docker Hub에 새로운 이미지가 푸시되면, Docker Hub는 Webhook을 통해 Portainer에 트리거 신호를 보낸다.
- 이 Webhook은 Portainer에 새로운 이미지가 준비되었음을 알리는 역할을 한다.

4. **Portainer가 Docker Hub에서 이미지 pull**
   - Portainer는 Webhook 신호를 받으면 Docker Hub로부터 새로운 이미지를 pull 한다.
5. **Portainer에서 EC2에 컨테이너 재빌드 요청**
   - Portainer는 AWS EC2 인스턴스에 새로운 이미지를 사용하여 컨테이너를 재빌드하도록 요청한다.
   - 이 요청은 기존 컨테이너를 종료하고, 새로운 이미지를 사용하여 새로운 컨테이너를 시작하는 과정을 포함한다.
6. **EC2 인스턴스에서 컨테이너 재빌드**
   - AWS EC2 인스턴스는 Portainer의 요청을 받아 새로운 컨테이너를 재빌드한다.
   - 재빌드된 컨테이너는 최신 코드 변경 사항을 반영한 상태로 실행된다.

이 CI/CD pipeline은 코드 변경 사항이 main 브랜치에 푸시되는 순간부터, 새로운 컨테이너가 AWS EC2 인스턴스에서 재배포될 때까지의 모든 단계를 자동화한다. 이를 통해 코드 변경 후 빠르게 배포하여 새로운 기능이나 수정 사항을 사용자에게 제공할 수 있도록 했으며, 배포 과정에서 발생할 수 있는 오류를 최소화한다.

2.2.4.3 데이터베이스 ERD

1. **Company**
   - 회사에 대한 정보를 저장하는 테이블이다. company 테이블의 정보를 이용하여 회사 인증을 진행하고 사용자가 어떤 회사에 소속되어있는지 파악할 수 있다.
2. **Upload_File**
   - 회사와 관련된 파일 정보를 저장하는 테이블이다. AWS S3와 연결된 링크를 포함하고 있다.
3. **User**
   - 사용자 정보를 저장하는 테이블이다. 사용자 프로필에서 직무를 표시하기 위해 position 테이블과 연결한다. userUUID는 loginID를 유추하여 특정 URL에 접근하는 문제를 방지하는데 사용된다.

4. User_Chatroom
   - 사용자와 채팅방 간의 관계를 나타내는 중간 테이블이다. 어떤 사용자가 어떤 채팅방에 있는지에 대한 정보를 나타낸다.
5. Chatroom
   - 채팅방 정보를 저장하는 테이블이다. 메시징 기능이 구현된 서비스에서 채팅방을 만들고 관리하기 위해 사용된다. 채팅방별 대화 기록을 유지하는 데에도 사용된다.
6. Message
   - 채팅방 메시지를 저장하는 테이블이다. 이전 채팅 기록을 확인하고 싶을 때 사용된다.
7. Position
   - 직무 테이블은 회사 내의 다양한 직무를 나타낸다. 백엔드, 프론트엔드, DevOps 등 직무를 설정할 수 있다. 현직자가 아닌 사람을 위한 학생, 무직 카테고리도 존재한다.
8. Company_Request
   - 회사 요청 테이블은 회사 인증 시 소속된 회사가 없는 경우에 대한 요청 사항을 저장한다. 각 요청은 사용자 ID를 통해 특정 사용자와 연결된다.
9. Review
   - 커피콩 매너온도를 설정하기 위한 정보를 저장하는 테이블이다. 사용자 간의 상호작용과 평가를 기록하여 서비스 개선에 기여할 수 있다.
10. Flyway_Schema_History
    - 데이터베이스 스키마 마이그레이션 히스토리를 관리하는 테이블이다. Flyway를 사용한 마이그레이션 정보를 기록하여 스키마 변경 내역을 추적한다.

2. .5 활용/개발된 기술

1. Flutter
   - ios, 안드로이드 모바일 앱을 제한된 시간 내에 개발하기 위해 크로스 플랫폼 프레임워크인 flutter를 선택했다. 크로스 플랫폼 프레임워크들 중에서도 flutter에는 내장된 기능이 많다는 장점이 있다. 내장된 기능들을 최대한 활용하여 외부 라이브러리

같은 서드 파티 의존성을 줄이고, 보다 안정적으로 서비스를 개발하기 위해 선택했다. 또한 flutter의 튜토리얼 자료나 문서가 체계적으로 작성되어 있어 학습하는 데 용이했고 필요한 내용을 빠르게 찾을 수 있었다.

2. **구글 맵 API**
   - 앱에 지도를 띄워 사용자의 위치를 표시하기 위해 geolocator 패키지를 사용했다.
   - 반경 내 카페들의 상세 정보(영업 시간, 테이크아웃 유무, 내부 식사 유무)와 카페 사진을 불러오기위해 google place Api를 활용하였다.
3. **Spring 프레임워크**
   - 오픈소스 프레임워크이면서 안정적인 개발과 개선이 보장된다. 스프링의 가장 큰 특징인 POJO(Plain Old Java Object) 프로그래밍을 지향한다는 점을 주목했고, 이를 통해 객체지향 설계가 용의하고 코드가 단순하기 때문에 테스트와 디버깅이 쉬워지는 등의 장점, 서비스를 개발할 때 기술보다 비즈니스 로직을 구현하는데에 집중할 수 있다는 장점 때문에 해당 프레임워크를 선택했다.
4. **Redis (인메모리데이터베이스)**
   - 매칭 요청, 온라인인 사용자의 상태 정보와 같이 DB에 영속적으로 남길 필요가 없는 휘발성 데이터를 서버 메모리에 캐싱해두는 목적으로 Redis 서버를 사용한다.
   - 매칭 요청은 제한시간(10분)이 주어져있는데, Redis의 expire 옵션을 사용해 제한시간(10분)이 지났을 때 별도의 서버 로직을 추가하지 않아도 자동으로 key를 메모리 해제할 수 있다. 이를 통해 메모리 효율성을 높이고자 한다.
5. **MySQL**
   - MySQL은 오픈 소스 데이터베이스 관리 시스템으로, 무료로 사용할 수 있기 때문에 초기 비용이나 라이선스 비용 없이 소프트웨어를 사용할 수 있다. 또한 오픈 소스이므로 커뮤니티 지원이 활발하며, 개발 및 유지보수가 용이하므로 프로젝트에 적합하다고 판단했다.
6. **WebSocket Stomp**
   - HTTP, WebSocket Stomp 등 다양한 통신 프로토콜 중 '실시간 채팅' 기능과 '카페 지정 및 위치 공유' 기능에는 Stomp 프로토콜이 적합하다고 판단하였다. HTTP는 단방향 통신이라 지속적인 채팅 연결에는 적합하지 않고, WebSocket은 메세지를 주고 받는 형식이 정해져있지 않아서 메세지 형식을 따로 지정해줘서, 프론트/백 측에서 각자 파싱을 해줘야하는 불편함이 있다. 반면, Stomp는 Websocket 기반이지만

Websocket과 달리 pub/sub 구조로 되어있으며, 클라이언트 간에 직접적인 메시지를 주고 받지 않고, 메세지 브로커가 발행된 메세지를 구독자에게 대신 전달해주는 구조이다. 이 덕분에 복잡한 네트워크 환경에서도 유연한 통신을 할 수 있고, 또한 메세지 브로커가 Websocket 연결 및 세션 관리와 같은 저수준의 세부사항을 대신 처리해주는 등 개발 측면에서 편리함이 있어 Stomp를 선택했다.

7. **EC2, Docker, Github Actions**

- 서버 배포를 위해서 AWS의 EC2를 선택했다. OS 호환 문제 때문에 Docker로 Spring-boot Application, Mysql, Redis 각각을 Docker 이미지로 말아서, EC2 인스턴스에 컨테이너로 띄워 서로 통신하도록 구현할 예정이다. 또한 서버 배포를 할 때마다 복잡하고 반복되는 위 과정을 자동화하기 위해 Github Actions 을 사용한다. Git Flow 브랜치 전략에 따라 master, develop, feature 브랜치로 나뉘어져 있는데, master 브랜치에 변동사항이 있을 때, Github Actions을 통해 서버가 배포되도록 할 예정이다.

2. .6 현실적 제한 요소 및 그 해결 방안

1. **Google Map 관련**

- 화면에 지도를 띄우는 과정에서 화면이 하얗게 나타나고 지도가 나타나지 않는 현상이 발생했다. android와 IOS의 API Key가 분리가 되지 않음을 파악했고 Key를 동일하게 수정하여 해결했다.
- 다음은 Google Map 현재 위치 가져오는 과정에서의 문제이다. Flutter의 emulator에서는 GPS 기능을 지원하지 않기 때문에 자신의 위치를 자동으로 읽을 수 없다. 따라서 현재 위치를 불러오는 기능을 수행할 때 엉뚱한 위치로 이동한다. 해당 기능을 테스트하기 위해 애뮬레이터의 위치를 국민대학교로 지정한 후 현재 위치로 이동 테스트를 진행했으며 성공했다. 실제 폰에서도 잘 작동된다.

2. **서버 배포 관련**

- AWS 프리티어에서 무료로 제공하는 EC2 t2.micro 인스턴스를 사용해서 서버를 배포하려고 시도 했었다. 근데 t2.micro 는 램이 1GB 밖에 되지 않아서 인스턴스에 spring-boot, mysql, redis 모두 띄웠을 때 중간에 동작이 멈추는 상황이 발생했다. 램 크기의 문제라고 판단하여 유료인 t2.large 인스턴스로 바꾸어서 해결했다.

- 서버 배포를 하는 과정에서 가장 많이 본 에러는 spring-boot 와 mysql 사이에 connection link failure 였다. 다양한 원인으로 연결이 안 됐었는데, mysql 서버가 셋업되기 전에 spring이 실행되는 상황이 있었어서, mysql이 정상적으로 실행됐는지 헬스체크를 하고 정상 실행된 이후에 spring을 실행하는 방법으로 해결했다.

2. .7 결과물 목록

번호	결과물	기술문서 유/무	링크
1	소스코드	무	https://github.com/kookmin-sw/capstone-2024-17
2	PR 및 issue	무	issues : https://github.com/kookmin-sw/capstone-2024-17/issues PR : https://github.com/kookmin-sw/capstone-2024-17/pulls
3	API 명세서	유	기술문서
4	ERD 설계도	유	기술문서
5	UseCase 다이어그램	유	기술문서
6	flowchart	유	기술문서
7	회의록	무	회의록 보고서 : https://drive.google.com/drive/u/0/folders/18dLZ3_y3KcgOVZjgMyAkG4NrXtwiNGJ- figma : https://www.figma.com/board/p6N3qOMLMxJEV7tCLRj0N9/%EC%BA%A1%EC%8A%A4%ED%86%A4-%ED%9A%8C%EC%9D%98%EB%A1%9D?t=OR16F6FFHqykMGGS

			-0
8	사용자 매뉴얼	유	기술문서
9	운영자 매뉴얼	유	기술문서

□ 기대효과 및 활용방안

1. 실시간 커피챗 매칭

다른 사용자에게 직접 연락을 취해 약속 날짜와 장소를 잡는 대신, 현재 가까운 곳에 위치한 사용자에게 커피챗 요청을 보내 바로 커피챗이 이루어질 수 있도록 한다. 이러한 실시간 매칭 방식은 사용자들의 편의성을 증가시키고 현재 우리나라의 딱딱하고 재미없는 커피챗의 이미지를 보다 가볍고 캐주얼하게 만들어 커피챗 문화 활성화에 기여할 수 있으리라 기대된다.

- 경쟁 서비스와의 비교

vs. 링크드인(LinkedIn): 직접적인 커피챗 매칭보다는 다른 사용자의 정보를 습득하고 연락을 취하는 용도로 사용된다. 커피챗 요청 시 적당히 예의와 양식을 갖춰 메시지를 보내야 하고, 상대로부터 답변이 오지 않거나, 답변이 오더라도 서로 가능한 일정을 맞춰야 하므로 커피챗이 성사되기까지 여러 절차와 번거로움을 요한다.

2. 대면 만남

대면 만남을 통한 커피챗은 온라인 커피챗에 비해 친밀감과 유대감을 형성하기 좋고, 깊은 대화를 이어나가기에 더욱 적합하다. 따라서 휴지처럼 뽑아 쓰고 버리는 단순 일회성 관계가 아닌, 지속적인 관계 및 비즈니스 인맥으로의 발전을 기대해볼 수 있다.

커피챗 목적을 기준으로 구분해서 보자면, 가볍게 회사에서 점심식사 후 남는 시간을 의미있게 쓰고자 하는 개발자는 대면 커피챗을 통해 산책 메이트이자 소중한 개발자 인맥을 얻을 수 있다. 또한 다른 직무로의 이직을 준비하는 이들에게 대면 커피챗은 즉각적인 소통으로 실무의 생생한 이야기를 들을 수 있어 더욱 실질적인 도움을 줄 수 있다. 주니어 개발자로부터 참신하고 기발한 젊은 세대의 시선을 배우고자 하는 시니어 개발자의 경우라면, 편안한 분위기의 일대일 만남을 통해 상대를 더 면밀히 살펴 회사에 필요한 인재를 찾을 수도 있을 것이다.

- **경쟁 서비스와의 비교**

vs. 커피챗(Coffeechat): 음성 커피챗 플랫폼. 커피챗 매칭을 해주는 플랫폼이지만, 음성 대화로 진행하며 멘티가 멘토에게 비용을 지불하는 멘토링 성격의 커피챗이다. 서로 도움을 주고받는 게 아니라 한 쪽이 일방적으로 도움을 준다는 점에서 커피챗의 본래 취지와는 동떨어진다. 또한 지속적인 관계 형성이 아닌 취업이나 이직 준비를 위한 일회성 전문가 컨설팅에 가깝다.

2. .1 자기평가

- **공채연**

이전에 경험했던 flutter 프로젝트와는 달리 클라이언트-서버간 통신이 중요한 주제의 프로젝트를 수행하게 되었다. google map api를 사용해본 경험이 처음인데 환경변수나 사소한 설정에 따라 까다롭게 반응해서 어려움을 겪었다.

google map의 플러그인 중 하나인 google places api를 사용하여 주변 카페를 검색했다.

검색할 때 Text Search 방식에서 Nearby Search 방식으로 변경하여 성능을 향상시켰다.

데이터가 업데이트되기 이전에 뜨지 않던 정보도 Nearby Search 방식에서는 최신화된 데이터를 제공하였기에 더 많은 정보를 가져올 수 있었다. 또한 Nearby Search에서는 500m 내의 정보 필터링이 정확하여 앞서 처리했던 이중 필터링을 피할 수 있었다.

해당 내용을 수정하기 전에는 좋지 않은 데이터로 캡스톤 프로젝트를 수행해야 한다는 막막함이 있었지만 최신화된 데이터를 가져왔을 때 큰 성취감을 느낄 수 있었다.

GitHub flow를 학습하면서 학부 프로젝트 수준에서 큰 협업을 진행했다고 생각한다.

feature, bug 등의 Issue template을 활용해서 branch 분기에 있어서 큰 효율을 얻었다.

매회의마다 todo list를 작성해서 issue로 만들었고, 화요일에 중간 점검을 하는 식으로 개발을 진행했다. slack과 github를 연동해두어 팀원들의 진행상황을 파악할 수 있어 개발이 활발하게 이루어졌다고 생각한다. 진행상황을 한 눈에 파악할 수 있는 milestone을 알게되어 앞으로도 잘 활용할 것 같다.

- **권해담**

플러터와 모바일 앱 모두 거의 처음 접해본 기술 스택이었기 때문에 이번 프로젝트는 쉽지 않은 도전이었다. 초반에는 개발에 들이는 시간보다 스터디에 들이는 시간이 더 많았고, 코드를 짜는 데 익숙하지 않아 좀처럼 속도가 붙지 않았다. 특히나 Provider나 context와 같은 전역 상태 관리와 관련된 개념들이 리액트 등의 웹 프론트엔드에서 다루는 개념과 약간 유사하면서도 달라 어렵게 느껴져 초반에는 잘 쓰지 않게 되었다. 그래서 state만을 사용해 상태를 관리했으나 점차 효율적이지 못하고 복잡하게 꼬이는 코드를 보며 한계를 느끼게 된 이후로 이제는 Provider와 ChangeNotifierProvider가 없이는 코드를 어떻게 짜나 싶을만큼 꽤 능숙하게 다룰 수 있는 수준이 되었다.

다음으로 어려웠던 부분은 동기와 비동기, 주요 업무의 처리 흐름과 관련된 부분이었다. 앱을 실행하면 주변 카페 목록을 가져오고 카페 목록의 유저 목록을 가져오는 http post 요청을 보내고, 웹소켓을 연결하여 카페 목록에 대한 sub 요청을 보내는 등의 주요한 업무들이 서로 얽혀있어 어떤 순서로 어떻게 처리할 지 결정하는 것이 쉽지 않았다. 운영체제 등 전공 수업시간에 배운 것 중 기억에 남아있는 아주 일부 내용만으로 코드를 짜다보니 부족한 점이 분명 있을 거라고 생각한다. 이 부분은 다음에 다른 프로젝트를 할 때 조금 더 보완해볼 계획이다.
마지막으로 인가를 위한 유저 토큰에 대해 깊이 있게 공부했던 것이 기억에 남는다. 회원 관리 부분을 직접 맡지는 않았지만, 인가가 필요한 api들에 토큰을 넘겨줄 때나, 토큰 만료 시 자동 로그아웃 처리 등에서 해당 부분에 대한 완벽한 이해를 필요로 했다. 특히 로그아웃 시 위치공유가 꺼져야 하는데, 유저가 직접 로그아웃하는 경우가 아닌 토큰 만료로 인해 로그아웃될 때 그것을 어떻게 알아차리며 어떻게 위치공유 오프라인으로 처리할 것인가를 정하면서 이해가 잡혀갔다.
개발 외적으로도 기획이나 디자인, 남들에게 우리의 서비스를 설득시키는 것, 협업 등에 있어서 이 프로젝트를 하기 전보다 훨씬 많이 배우고 성장했음을 느낀다.

- **김승언**

이번 프로젝트를 통해 한 번도 경험해보지 못했던 기술을 공부하고 직접 적용해볼 수 있었다. 특히, Redis Cache를 처음 사용해보면서 MySQL과의 차이를 체감할 수 있었는데, Redis를 활용해 매칭 요청을 실시간으로 처리하면서 데이터의 빠른 읽기/쓰기가 필요한 경우에는 Redis가 얼마나 유용한지 깨달을 수 있었다. 또한, 매칭 요청에 대한 락을 설정하여 중복 요청을 방지하고, 10분 동안 요청을 유효하게 유지하는 로직을 구현하면서 Redis의 다양한 기능을 익히게 되었다.
또한, Custom Exception을 통해 예외처리를 꼼꼼히 해보면서 클라이언트와의 디버깅 시간을 훨씬 단축시킬 수 있었다. 예외 상황에 대해 구체적인 메시지를 제공함으로써 문제 발생 시 정확한 원인을 빠르게 파악할 수 있었고, 이를 통해 예외처리의 중요성을 크게 깨달았다. 특히, 매칭 요청, 수락, 거절, 취소, 종료 등의 다양한 상황에 대해 세심한 예외처리를 구현하면서 시스템의 안정성과 신뢰성을 높일 수 있었는데 어떤 로직에서 오류가 나는지 바로 찾을 때마다

너무 기뻤다. 그리고, FCM을 통해 매칭 요청, 수락, 거절 등의 이벤트에 대해 실시간으로 푸시 알림을 전송하는 로직을 구현해보면서 새로운 기술에 대한 재미를 다시금 느끼는 경험이 되었다. 마지막으로, 이번 프로젝트는 개발 인원이 많아 여러 의견이 나왔기에 최종 기획안을 정하는 데 시간이 오래 걸렸다. 그러나 이렇게 오랫동안 시간을 투자해서 기획을 탄탄히 잡았기에 개발할 때 더욱 수월했고, 프로젝트의 방향성과 목표를 명확하게 설정할 수 있었다. 이처럼 주 2회 팀 회의를 하면서 다양한 의견을 조율하는 방법을 배웠고, 개발 프로세스 중 Agile 개발 프로세스도 익힐 수 있었다. 이러한 경험을 통해 새로운 기술을 두려워하지 않고 도전할 수 있는 용기를 얻었고, 앞으로도 지속적으로 학습하고 성장하는 개발자가 되겠다는 다짐하게 되었다. 이번 프로젝트는 나에게 있어 매우 값진 경험이었고, 앞으로의 개발자 생활에 큰 밑거름이 될 것이라고 자부한다.

- **김혜은**

이번 프로젝트에서 가장 기억에 남는 작업은 서버를 AWS EC2에 배포하고, github actions, portainer 로 자동화 시키는 것이였다. 서버 배포하기 위해 부단히 노력했지만, spring/mysql/redis 등 다양한 인프라가 엮여있었고, 초반에 docker 컨테이너 간에 어떻게 통신하는지에 대한 이해가 부족했어서 배포하는데 계속 문제가 있었는데, 구글링하면서 공부하고, 지인에게 도움도 요청 하면서 결국 해결해내서 그 과정을 notion에 기록헤서 다시 회고할 수 있었다. 이후 백엔드 배포 자동화 시스템을 Github Actions 와 Portainer를 이용해서 구축했다. 또한 빠른 에러 디버깅을 위해 LogBack 라이브러리를 이용해 서버 로깅 커스텀 작업을 했었다. 서버 배포하는 작업이 어렵고 힘들었지만, 인프라 구축에 대해 공부할 수 있는 좋은 기회였고, 안정적인 배포 시스템을 구축한 뒤 문제 없이 프론트 팀원들이 최신 코드를 바로바로 사용하는 모습이 가장 뿌듯했었다.

또한 STOMP pub/sub과 Redis 를 활용한 카페선택 기능을 담당했었는데, 프론트 및 백엔드 친구들과 어떻게 하면 앱에 부하를 줄이고, 통신량도 줄이면서 우리가 원하는 실시간 반영을 구현할 수 있을까에 대해 많은 논의를 거쳐서 구현한 기능이였다. 데이터 일관성을 위해 프론트에서 웹소켓 연결이 끊기는 것을 서버측에서 감지해 해당 유저를 오프라인 시키는 작업도 기억에 남는다.

이렇게 약 3개월 간 6명의 팀원들과 열심히 기획하고, 어떻게 하면 최선의 선택을 할 수 있을지 서로 도움주고, 공식적인 회의 외에 비공식적인 회의에도 적극적으로 참여하며 팀원들의 로직 고민 관련해서 열심히 피드백 주었던 과정들이 뿌듯하다. 그리고 많은 팀원들과 이렇게 큰

프로젝트를 해본게 처음이여서 Github Issue 및 PR 을 왜 사용해야하는지, 애자일하게 프로젝트를 진행하는 것들이 왜 중요한지 제대로 깨달을 수 있었던 경험이었고, 앞으로 기본이 잡힌, 협업을 잘하는 개발자가 되기 위한 좋은 토대가 될 것이다.

- 이하람

Spring 프레임워크를 처음 경험한 만큼 부딪히고 깨지면서 단기간에 많은 것을 배울 수 있었다. 그런 과정에서 Spring 이전에 경험했던 Django 프레임워크가 제공했던 User authentication, APIView, ViewSet 등 많은 기능들이 얼마나 강력하면서도 편리한 기능이었는지 비교할 수 있었다. Spring을 사용하면서 더 깊은 영역을 고려하고 뜯어보는 경험이 새로웠다. 처음엔 api 엔드포인트 하나를 완성하는데에도 시간이 오래걸리고 버벅거렸는데, 프로젝트의 구조와 Spring 프레임워크가 제공하는 DI나 IoC, 자바 언어와 어노테이션 등에 대한 이해가 높아지면서 이후 AWS Storage service나 smtp Mail service를 개발할 땐 이미 능숙하게 작업할 수 있게되었다.
이전에 경험했던 프로젝트에 비해 설계를 위해 많은 시간을 들였다. 코드를 짜기 전에 어떤 방법으로 기능을 구현할 것인지, 왜 그렇게 해야하는지 토론하는 과정이 험난했지만, 그 가운데에서 효과적으로 내 생각을 전달하는 방법과 팀원들과 부드럽게 소통하는 법을 많이 배웠다. 기술적인 부분에서도 로직을 설계할 때 웹소켓과 http request/response 사이에서 어느쪽이 서버와 프론트에 덜 부담이 갈지, 운영까지 고려하면서 DB와 redis 메모리 사이에 어떤것을 선택해야 할지 고민했었던 것이 기억에 남는다.
한편, DB를 설계하고 수정하는 과정을 반복하면서 배포 서버를 자주 재부팅, 초기화하는 작업이 필요했다. 이때 운영 서버는 hibernate에 의존하지 않고 git 처럼 DB 형상을 관리해야한다는 것을 처음 알게 되었다. flyway를 적용시키면서 seed data나, 필드의 default value를 언제 어떻게 설정하는것이 좋은지 등에 대해서도 고민하면서 DB에 대한 이해가 높아졌다.
아쉬웠던 것은, stomp 프로토콜을 사용하면서 추가로 외부 메시지 브로커를 도입해 모니터링과 더불어 안정성을 높이고싶었지만, 시간부족으로 In Memory 브로커만 적용한 점이 아쉽다. 또 fcm 서비스가 종종 작동하지 않는 경우가 있었는데 그 원인을 자세하게 파악하지 못한 것이 아쉬웠다.
협업 면에서는 branch 전략, Issue template과 Pr, review request 등 협업을 위한 규칙을 세우고 지키는 것이 프로젝트 진행에 많은 도움이 되는것을 경험할 수 있었다. 이슈를 올리고 pr을 올리기 위해서 한 번 더 생각하게되는 점, 팀원들에게 작업 결과와 내가 고민한 부분을 효과적으로 공유할 수 있는 점에서 그렇다. 6명이라는 작지 않은 규모의 팀 프로젝트가 원만히

마무리되어 뿌듯하고 기쁘다. 팀원 모두가 밤잠 줄여가며 서로를 배려하는 마음이 없었다면 이룰 수 없었을 만족스러운 마무리라고 생각한다.

- **홍지민**

팀 인원이 많다 보니 주제를 정하는 과정이 많은 시간을 소모했고, 몇 차례의 기획 변경을 거치면서 앞으로 잘 해낼 수 있을까 걱정하기도 했다. 하지만 그런 고민의 시간이 있었기에 팀원들이 모두 만족할 만한 주제를 선정할 수 있었고, 결과적으로 더 좋은 팀워크를 이끌어낼 수 있었다고 생각한다. 주제가 주제이다보니 개발자 네트워킹에 대한 의지가 더욱 확고해진 것 같기도 하다.

이번 프로젝트에서 프론트엔드를 맡으며 Flutter를 사용해 처음으로 앱 개발을 시도하게 되었는데, 처음이었던 만큼 여러 가지 기술적인 어려움이 있었다. 특히 위젯의 빌드와 상태 관리가 매우 까다로웠다. 그래도 사용자에게 편리하고 직관적인 인터페이스를 제공하려고 노력하다보니 많은 공부가 되었다. 개인적으로는 FCM을 사용해서 첫 푸시 알림을 받았을 때가 기억에 남는다. 그 순간 우리 팀이 만들고 있는 앱이 실제로 사용자와 소통하고 있다는 것을 실감했다. 사용자 경험을 개선하기 위해 필요한 도구들을 습득하는 것이 얼마나 중요한지 깨닫게 되었다.

협업 과정에서는 일주일에 두 번씩 정기적으로 회의를 진행하고 회의록을 작성하면서 팀원들의 의견을 더욱 귀담아 들을 수 있었고, 내가 놓쳤던 부분을 되돌아볼 수 있었다. 여러모로 앞으로의 개발자 커리어에 큰 도움이 될 프로젝트라고 생각한다.

3 참고 문헌

번호	종류	제목	출처	발행년도	저자	기타
1	기술문서	flutter documentation	https://docs.flutter.dev/			

2	기술문서	카카오 로그인: flutter	https://developers.kakao.com/docs/latest/ko/kakaologin/flutter			
3	웹페이지	Redis Pub/Sub을 활용한 쿠폰 발급 비동기 처리	https://oliveyoung.tech/blog/2023-08-07/async-process-of-coupon-issuance-using-redis/	2023	어푸	
4	웹페이지	spring RedisTemplate.hasKey()값이 null이 나올 경우	https://velog.io/@greentea/spring-RedisTemplate.hasKey%EA%B0%92%EC%9D%B4-null%EC%9D%B4-%EB%82%98%EC%98%AC-%EA%B2%BD%EC%9A%B0	2023	GreenTea	
5	웹페이지	[스프링부트] Spring Boot의 Validation : @Valid 어노테이션으로 유효성 검증하기	https://velog.io/@minji/%EC%8A%A4%ED%94%84%EB%A7%81%EB%B6%80%ED%8A%B8-Spring-Boot%EC%9D%98-Validation-Valid-%EC%96%B4%EB%85%B8%ED%85%8C%EC%9D%B4%EC%85%98%EC%9C%BC%EB%A1%9C-%EC%9C%A0%ED%9A%A8%EC%84%B1-%EA%B2%80%EC%A6%9D%ED%95%98%EA%B8%B0	2022	minji	

4 부록

☐ 사용자 매뉴얼

사용법 보기**[Link] -** Youtube

☐ 운영자 매뉴얼

[frontend]

flutter --version

- Flutter 3.19.6 • channel stable
- Dart 3.3.4 • DevTools 2.31.1

Android SDK Manager - SDK Tools 이동

Android SDK Command-line Tools 설치

안드로이드 스튜디오 **-> device manager -> create virtual device ->** 디바이스 생성 **->** 실행

```
git clone https://github.com/kookmin-sw/capstone-2024-17.git
cd frontend
flutter pub get
flutter run
```

[backend]

Backend 설치 가이드

□ 테스트 케이스

대분류	소분류	기능	테스트 방법	기대 결과	테스트 결과
인증	*로그인 기능*	일반 로그인 테스트	1) signIn 메서드를 호출하여 기존 사용자가 로그인한다. 2) 로그인에 성공하면 AuthDto가 반환되는지 확인한다.	로그인이 성공적으로 이루어지고, AuthDto가 반환되며, 토큰이 Redis에 저장된다.	성공
인증	*로그인 기능*	잘못된 비밀번호 로그인 테스트	1) signIn 메서드를 호출하여 잘못된 비밀번호로 로그인 시도. 2) CustomException이 발생하는지 확인.	'CustomException이 발생하고, 에러 코드가 ErrorCode.LOGIN_FAILED이다.	성공
커피챗	*매칭 기능*	매칭 요청	1) sendMatchRequest 메서드를 호출하여 매칭 요청을 보낸다. 2) 매칭 요청이 Redis에 저장되었는지 확인한다. 3) 수신자에게 푸시 알림이 전송되었는지 확인한다.	매칭 요청이 성공적으로 저장되고, 수신자에게 푸시 알림이 전송된다	성공
커피챗	*매칭 기능*	매칭 수락	1) acceptMatchRequest 메서드를 호출하여 매칭 요청을 수락한다.	매칭 요청이 성공적으로 수락되고,	성공

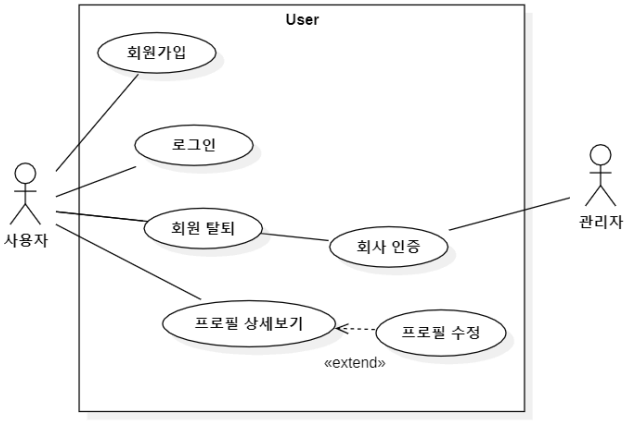
			2) 매칭 요청 상태가 "accepted"로 변경되었는지 확인한다. 3) 채팅방이 생성되었는지 확인한다. 4) 송신자에게 푸시 알림이 전송되었는지 확인한다.	송신자에게 푸시 알림이 전송된다	
커피챗	*매칭 기능*	매칭 거절	1) declineMatchRequest 메서드를 호출하여 매칭 요청을 거절한다. 2) 매칭 요청 상태가 "declined"로 변경되었는지 확인한다. 3) 송신자에게 푸시 알림이 전송되었는지 확인한다.	매칭 요청이 성공적으로 거절되고, 송신자에게 푸시 알림이 전송된다	성공
커피챗	*매칭 기능*	매칭 취소	1) cancelMatchRequest 메서드를 호출하여 매칭 요청을 취소한다. 2) 매칭 요청 상태가 "canceled"로 변경되었는지 확인한다. 3) 수신자에게 푸시 알림이 전송되었는지 확인한다.	매칭 요청이 성공적으로 취소되고, 수신자에게 푸시 알림이 전송된다	성공
커피챗	*매칭 기능*	매칭 종료	1) finishMatch 메서드를 호출하여 매칭을 종료한다. 2) 매칭 요청 상태가 "finished"로 변경되었는지 확인한다.	매칭이 성공적으로 종료되고, 상대방에게 푸시 알림이 전송되며,	성공

			3) 상대방에게 푸시 알림이 전송되었는지 확인한다. 4) 매칭중 상태가 "no"로 변경되었는지 확인한다.	매칭 중 상태가 "no"로 변경된다.	
커피챗	*매칭 기능*	보낸 요청 목록 조회	1) getMatchRequestInfo 메서드를 호출하여 보낸 매칭 요청 목록을 조회한다. 2) 반환된 목록이 예상과 일치하는지 확인한다.	보낸 매칭 요청 목록이 올바르게 반환된다.	성공
커피챗	*매칭 기능*	받은 요청 목록 조회	1) getMatchReceivedInfo 메서드를 호출하여 받은 매칭 요청 목록을 조회한다. 2) 반환된 목록이 예상과 일치하는지 확인한다.	받은 매칭 요청 목록이 올바르게 반환된다.	성공

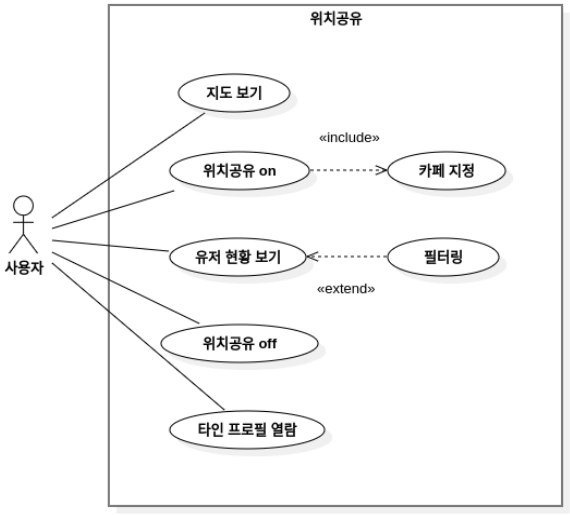
□ “커리어 한잔”에 대한 기술 문서

4.4.1 UseCase Diagram

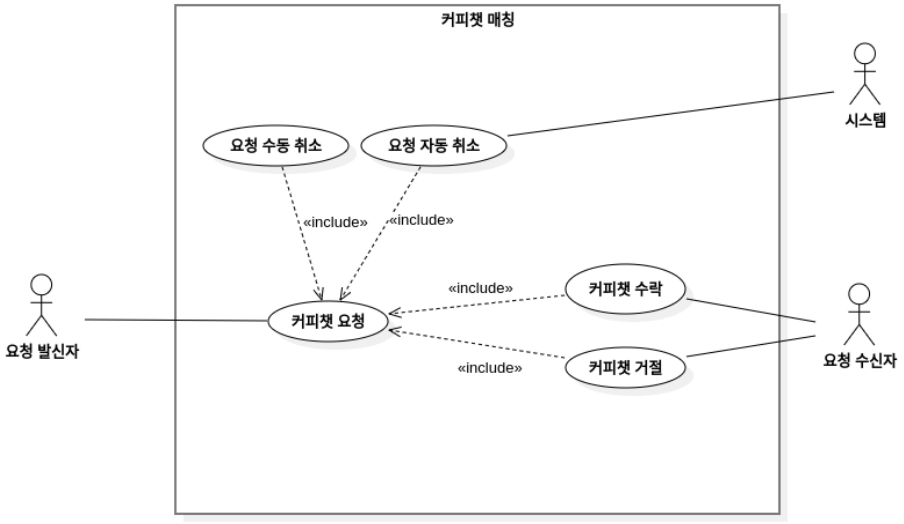
1. 유저 UseCase Diagram



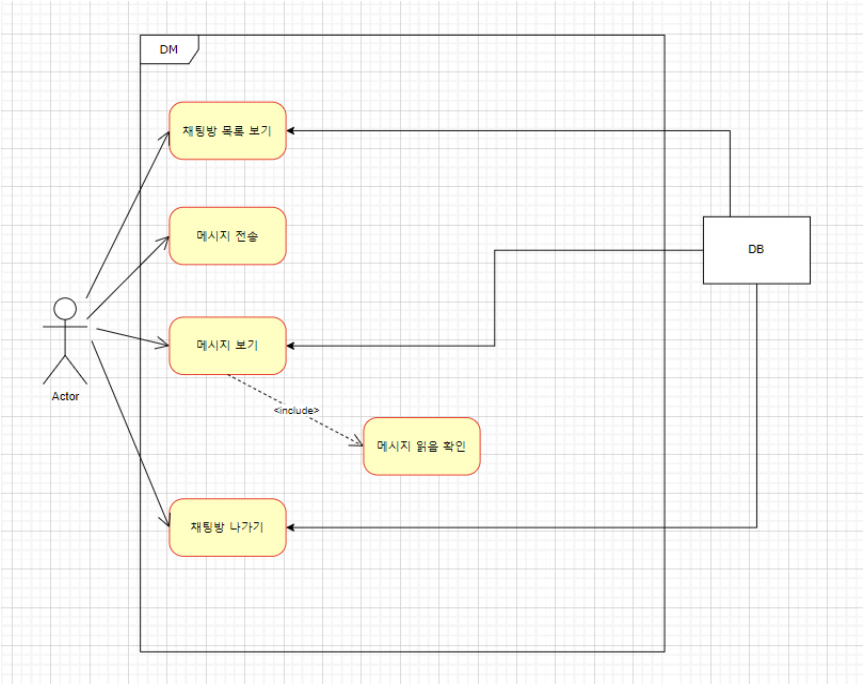
2. 위치 공유 UseCase Diagram



3. 커피챗 매칭 UseCase Diagram

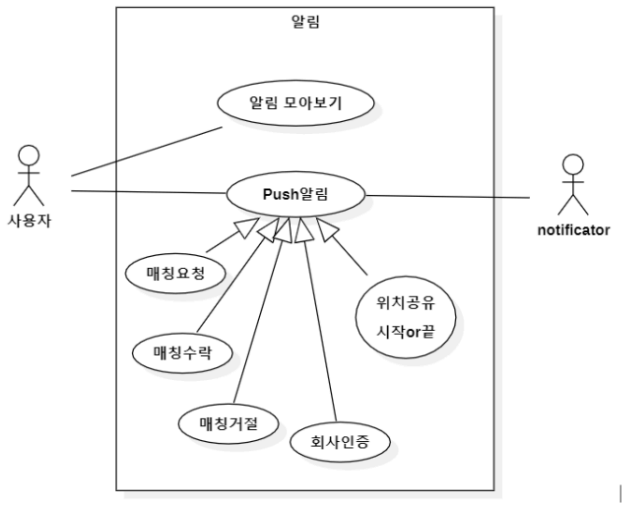


4. 채팅 UseCase Diagram



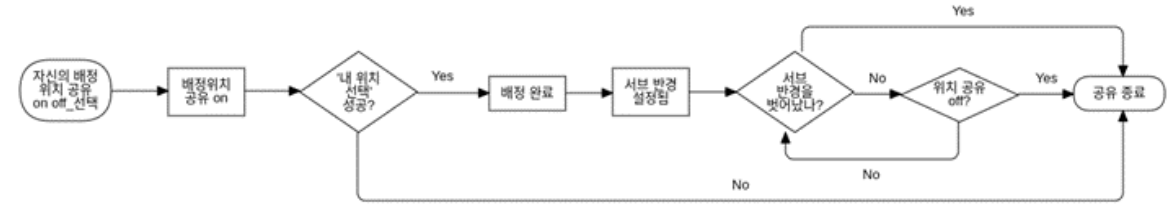
5. 알림 UseCase Diagram

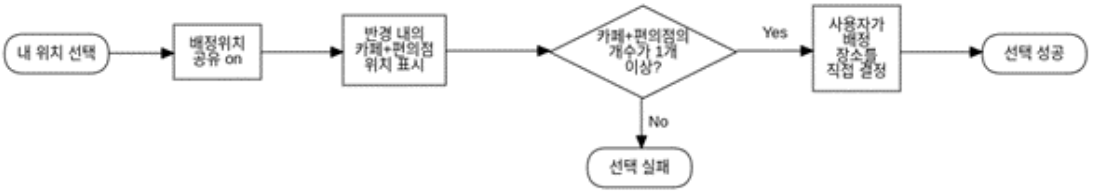
[Notification]

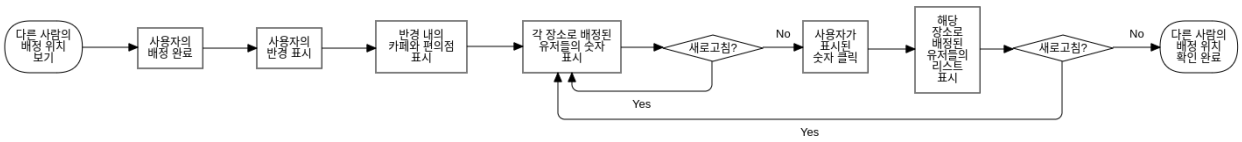


4.4.2 FlowChart

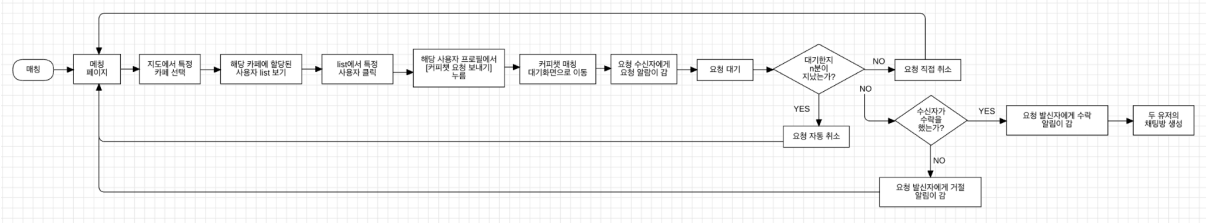
[위치공유]



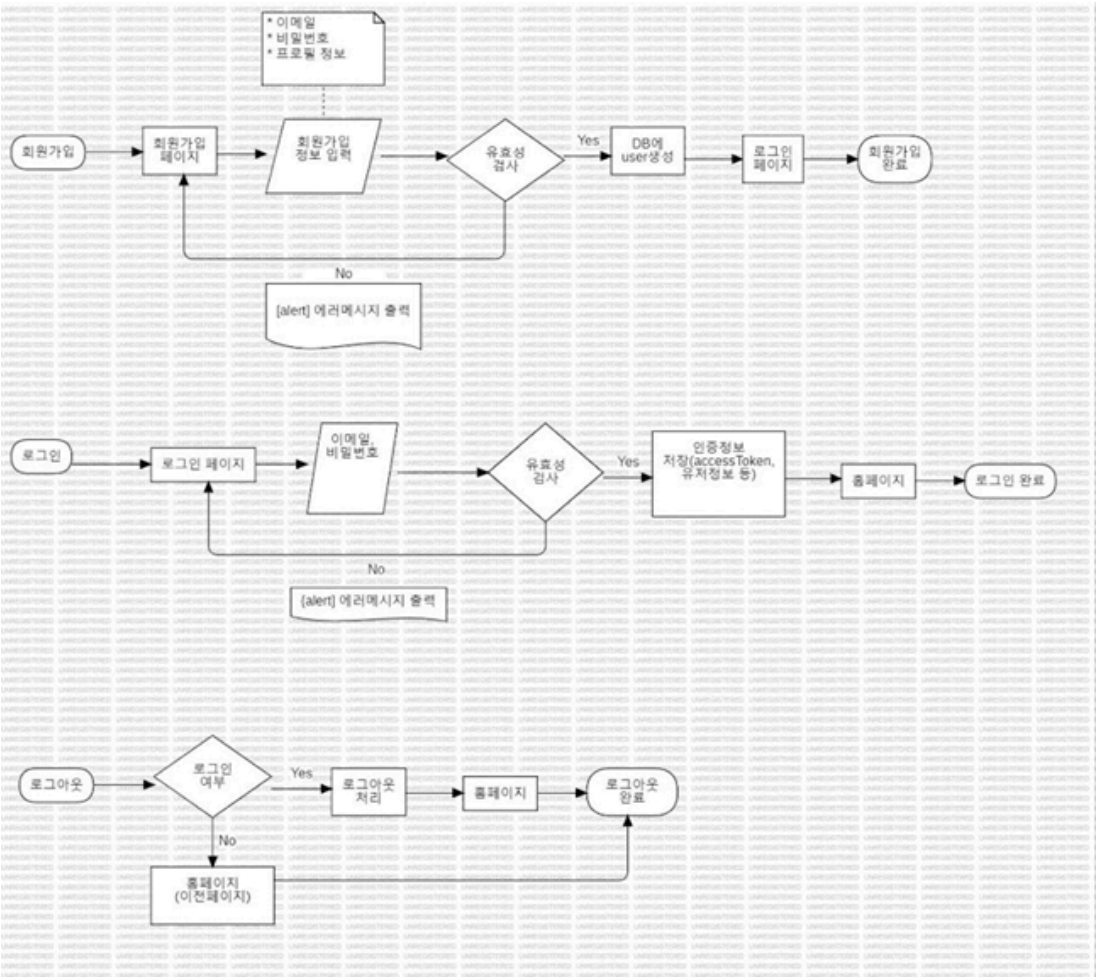


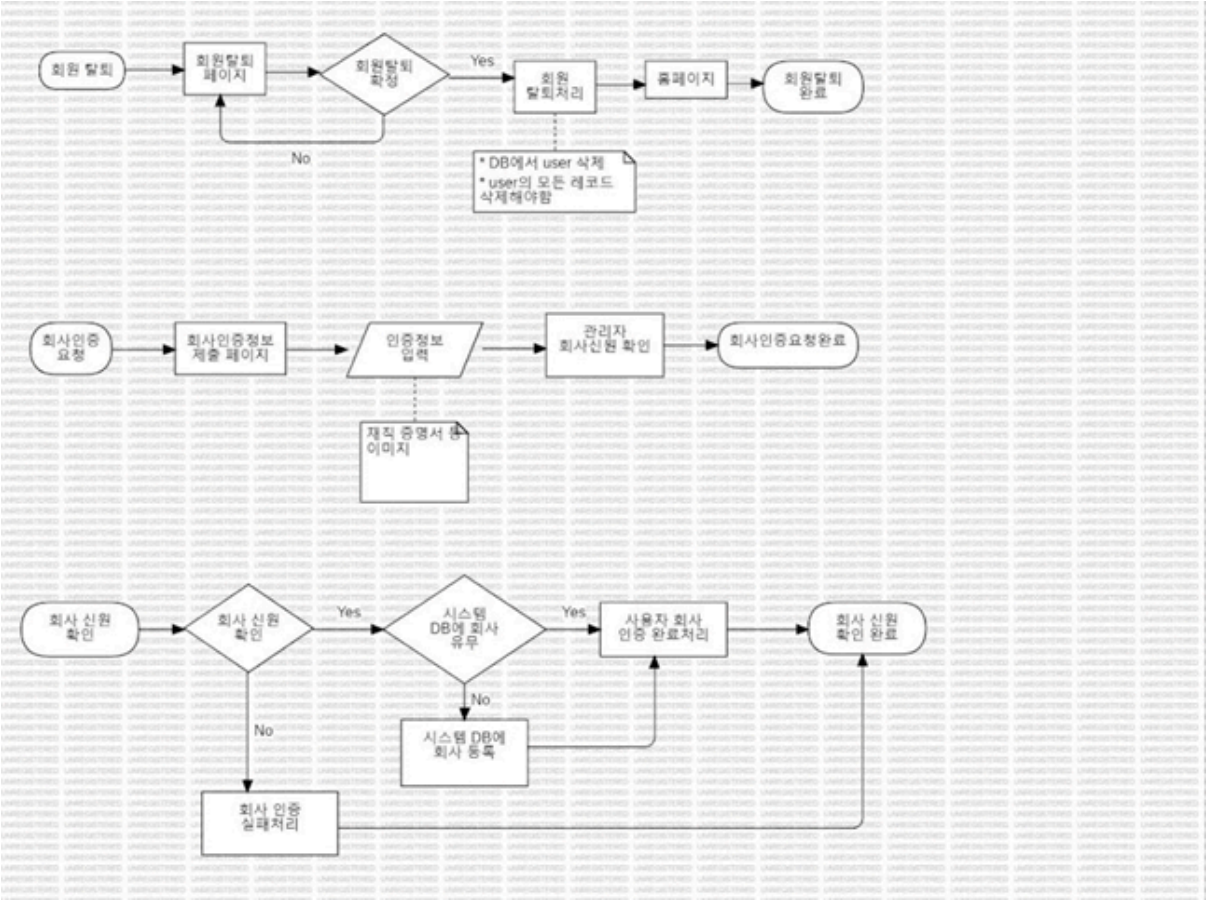


[커피챗 매칭]

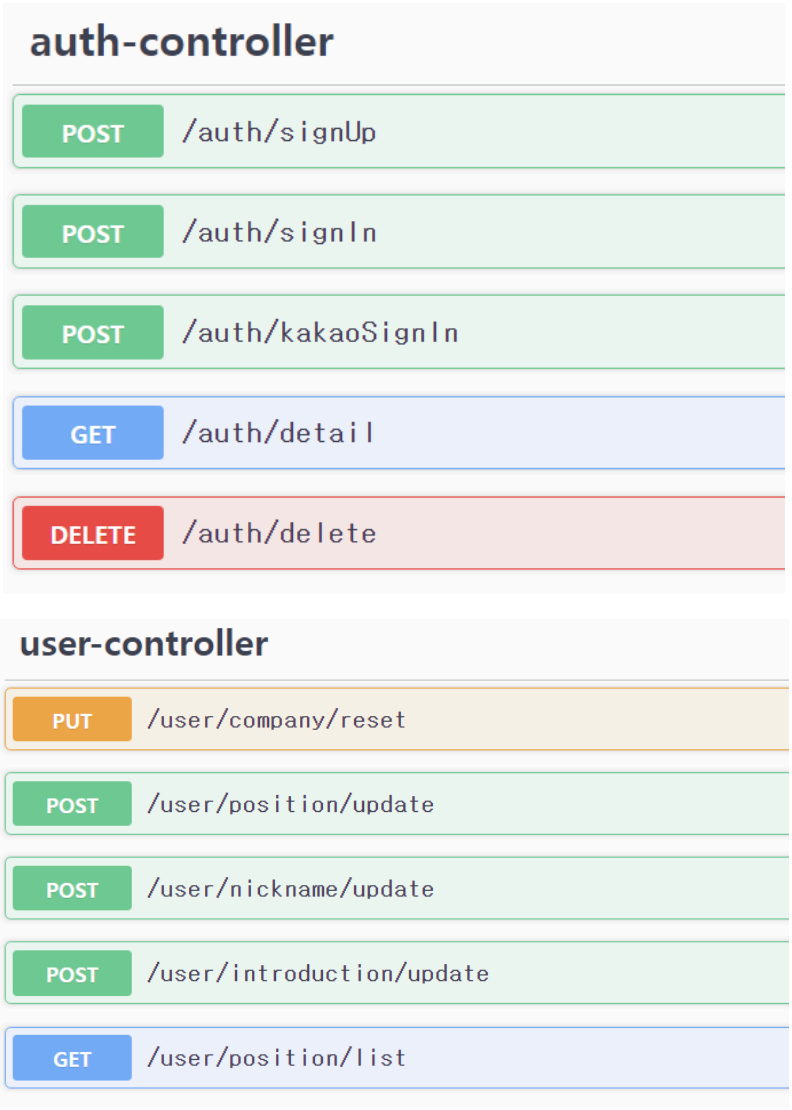


[유저]





4.4.3 API 명세서



match-controller

PUT	/match/finish
PUT	/match/decline
PUT	/match/cancel
PUT	/match/accept
POST	/match/review
POST	/match/request
GET	/match/request/info
GET	/match/received/info
GET	/match/isMatching
GET	/match/check/reviewed

company-controller

POST	/email/verification
POST	/email/verification-request
GET	/company/search

company-request-controller

POST	/company/request

chatroom-controller

POST /chatroom/create

GET /chatroom/list

message-controller

GET /message/list

cafe-controller

POST /cafe/get-users

4.4.4 협업 규칙

Commit Message Convention

[기능에 영향]

- [feat] 새로운 기능, 페이지 추가
- [fix] 버그 수정
- [chore] 자잘한 수정 (디펜던시, gitignore, prettier, build script)
- [perf] 성능 개선

[기능 영향X]

- [refactor] 리팩토링(결과의 변경 없이 코드의 구조를 재조정 / 가독성 및 유지보수)
- [style] UI를 추가/수정하거나, 스타일 관련 작업의 경우
- [test] 테스트 관련
- [docs] 문서 관련 수정 (리드미, 다이어그램,, 등 코드말고 문서)
- [release] 배포관련 작업
- [remove] 파일 또는 코드를 삭제만 한 경우 (이미지파일 등)
- [add] 새로운 파일 추가
- [move] 코드나 파일을 이동하는 경우
- [rename] 파일 혹은 폴더명을 수정 한 경우

4.4.5 Agile을 적용한 프로젝트 진행