

# 푸드트럭 플랫폼 (Connect-luck)

본 보고서를 2024년도 Project 푸드트럭 플랫폼의  
최종보고서로 제출합니다.

학부(전공)/학과: 컴퓨터소프트웨어공학과

담당(지도)교수: 이종민

팀명: 코드 미라클

제출일: 2024년 06월 01일

팀장: 김남주 20192336

팀원1: 이종민 20193174

팀원2: 이태수 20193164

팀원3: 김민곤 20193172

팀원4: 오승현 20193184

동의대학교 컴퓨터소프트웨어공학과 학과장 귀하

## 〈 요 약 문 〉

프로젝트 목표 (300자내외)	ConnectLuck은 푸드트럭 사업자와 행사 관계자 간의 효율적인 매칭을 목표로 하는 플랫폼이다. 기존의 푸드트럭 업계는 중간 소개인이 푸드트럭을 모집하여 행사에 보내주고 푸드트럭에게서 일정량의 수수료를 받아가는 방식을 이용했다. 하지만 ConnectLuck에서 행사 관계자는 원하는 메뉴와 규모에 맞는 푸드트럭을 쉽게 찾고 모집할 수 있으며, 푸드트럭 사업자는 적합한 행사에 직접 참가 신청을 할 수 있다. 이를 통해 행사는 다양하고 맛있는 음식 옵션을 제공하고, 푸드트럭 사업자는 새로운 고객층에게 자신을 알릴 수 있는 기회를 얻는다. 이렇듯 ConnectLuck은 양측의 요구를 충족시키며 효과적인 비즈니스 연결을 제공하는 것을 목표로 한다.						
내용 (500자내외)	<p>기존 푸드트럭 업계에서는 대부분의 행사 참여 기회가 중개인을 통해 이루어졌다. 이 과정에서 중개인은 상당한 수수료를 취하며, 중개인과 계약하지 않은 푸드트럭 사업자들은 행사 참여 기회조차 얻기 어려웠다. 푸드트럭 사업자들은 행사 참여를 위해 부득이하게 높은 수수료를 지불해야 했고, 행사 관계자 또한 중개인이 제공하는 한정된 선택지 내에서만 푸드트럭을 선택할 수 있었다. 이러한 구조는 푸드트럭 사업자와 행사 관계자 양측 모두에게 제한적인 선택권과 불합리한 조건을 강요했다.</p> <p>ConnectLuck 프로젝트는 이러한 구조를 혁신적으로 변화시키고자 한다. ConnectLuck은 행사 관계자와 푸드트럭 사업자를 직접 연결해주는 플랫폼으로, 행사 관계자가 필요한 메뉴, 규모, 시간 등의 조건을 기준으로 쉽게 원하는 푸드트럭을 모집할 수 있게 해준다. 반대로, 푸드트럭 사업자는 자신의 서비스를 공정하게 행사에 제안하고, 직접 홍보할 수 있는 기회를 얻는다. 이 플랫폼을 통해 행사 관계자는 다양한 푸드트럭 옵션 중에서 최적의 선택을 할 수 있고, 푸드트럭 사업자는 중개인을 거치지 않고 직접 행사에 참여할 수 있게 된다.</p> <p>ConnectLuck은 푸드트럭 업계의 기존 문제점을 해결하고, 양측 모두에게 더욱 공정하고 개방된 비즈니스 환경을 제공하기 위해 계획되었다. 이 플랫폼은 푸드트럭 사업자와 행사 관계자가 서로의 요구사항에 맞춰 직접적으로 연결될 수 있도록 함으로써, 상호 이익을 극대화하고 업계의 새로운 기준을 설정하고자 한다.</p>						
기대효과 (200자내외)	ConnectLuck 프로젝트는 푸드트럭 사업자와 행사 관계자를 직접 연결함으로써 중개인에 의존하지 않는 투명한 거래 환경을 조성하고, 양측에 경제적 이익을 제공한다. 이를 통해 푸드트럭 사업자는 공정한 기회를 얻고, 행사 관계자는 다양하고 맞춤화된 선택을 할 수 있게 된다. 이 과정에서 시장의 활성화가 기대되며, 행사의 질적 향상과 참여자 만족도 증가로 이어질 것이다.						
Keywords	<table border="1" style="width: 100%;"><tr><td style="width: 33%;">푸드트럭</td><td style="width: 33%;">행사</td><td style="width: 33%;">매칭</td></tr><tr><td>연결</td><td>양방향</td><td></td></tr></table>	푸드트럭	행사	매칭	연결	양방향	
푸드트럭	행사	매칭					
연결	양방향						

# 목 차

## I. 서론

1. 프로젝트의 배경 또는 필요성 .....	1
2. 프로젝트의 목표 .....	2
3. 문제 기술 .....	3

## II. 프로젝트 수행 내용

1. 프로젝트의 추진전략 및 방법(체계) .....	4
2. 프로젝트 수행 과정(분석 및 설계 포함) .....	6
3. 프로젝트 수행 결과 .....	111

## III. 결론

1. 프로젝트 달성을 .....	114
2. 프로젝트 결과 논의 .....	115
3. 프로젝트 결과의 활용 방안 .....	115
4. 프로젝트 결과의 기대 성과 .....	116

## IV. 참고문헌 .....

## 116

## [부록] .....

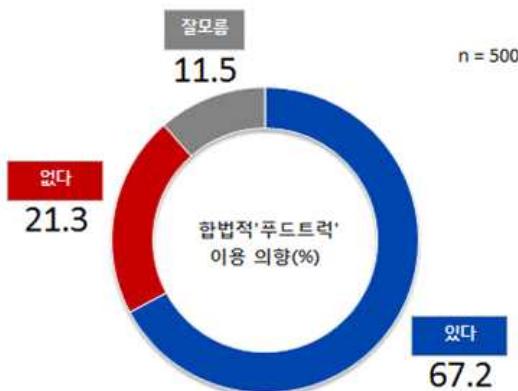
## 116

# I. 서론

## 1. 프로젝트의 배경 또는 필요성

### 1) 현재 사회적 동향 및 전망:

현재 사회에서는 온라인 플랫폼을 통한 서비스 이용이 더욱 확산되고 있다. 이는 특히 푸드트럭 산업에도 영향을 미치고 있으며, 행사나 이벤트에서 푸드트럭을 활용하는 경우가 늘어나고 있다. 또한, 코로나19로 인해 푸드트럭 시장이 침체된 시기를 겪었지만, 현재는 코로나19가 종식됨에 따라 푸드트럭을 활용하는 비중이 재차 증가할 것으로 예상된다.



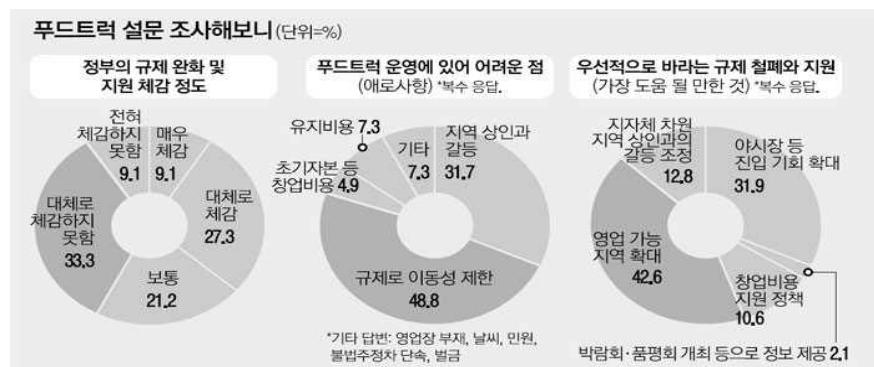
[1] 합법적 푸드트럭 이용 의향 설문조사 그래프



[2] 부산시 푸드트럭 운영현황

### 2) 기존 문제점 분석 및 새로운 요구사항:

기존의 푸드트럭 매칭 방식은 중개인을 통한 번거로운 절차가 있다. 또한, 푸드트럭 사업자와 행사 관계자 간에 소통이 원활하지 않아 효율성이 떨어지는 경우가 많았다. 더불어 푸드트럭 사업자들은 행사 참가 여부를 결정할 때 적절한 정보를 얻기 어려웠고, 행사 관계자들은 원하는 푸드트럭을 쉽게 섭외하기 어려웠다.



[1] 푸드트럭 설문 표

### 3) 프로젝트의 필요성:

위의 문제점을 해결하기 위해 ConnectLuck이 필요로 한다. 이 플랫폼은 푸드트럭 사업자와 행사 관계자를 직접 연결하여 중개인 없이 효율적으로 매칭할 수 있다. 또한, 푸드트럭 사업자는 행사 정보를 쉽게 확인하고 참가를 신청할 수 있으며, 행사 관계자는 원하는 푸드트럭을 직접 섭외할 수 있어 편리하다. 이를 통해 푸드트럭 산업의 발전과 행사 이벤트의 성공을 도모할 수 있다.

## 2. 프로젝트의 목표

### (1) 공학적 목표:

#### 가) 양방향 매칭 시스템 구축:

ConnectLuck 플랫폼은 푸드트럭 사업자와 행사 관계자가 각자 자신의 요구사항을 등록할 수 있도록 한다. 이를 통해 양쪽의 조건을 기반으로 상호 매칭이 이루어진다.

#### 나) 투명하고 신뢰성 있는 플랫폼 운영:

플랫폼 내에서 행사 정보와 푸드트럭 정보를 신뢰성 있게 제공하고, 사용자 간 소통이 원활하게 이뤄지도록 한다. 이를 위해 안정적인 서버 운영과 사용자 인터페이스 개선을 포함한 기술적 개선이 이루어진다.

#### 다) 알림 및 상호 피드백 시스템:

매칭이 이루어지면 ConnectLuck는 푸드트럭 사업자와 행사 관계자에게 알림을 전송하여 상호간의 의사소통을 촉진한다. 또한, 행사 종료 후에는 푸드트럭 사업자와 행사 관계자 간의 평가 및 피드백을 통해 서비스의 품질을 개선한다.

### (2) 비공학적 목표:

#### 가) 푸드트럭 산업의 활성화:

푸드트럭 사업자들이 더 많은 행사 및 이벤트에 참여할 수 있도록 돋는다. 이를 통해 푸드트럭 산업의 활성화와 사업 확장을 지원한다.

#### 나) 행사 및 이벤트의 품질 향상:

행사 주최자가 다양한 종류의 푸드트럭을 쉽게 섭외하여 참여할 수 있도록 한다. 이를 통해 행사 및 이벤트의 참여자들에게 더욱 다채로운 식문화 경험을 제공한다.

#### 다) 사회 경제적 활동 지원:

푸드트럭 사업자들에게 새로운 비즈니스 기회를 제공하여 사회 경제적 활동을 지원하고 지역 사회의 경제적 활동을 촉진한다.

### 3. 문제 기술

#### 1) 매칭 관리:

기존의 푸드트럭 매니저와 행사 관계자가 서로를 찾고 계약하는 과정이 복잡하고 시간과 비용이 많이 발생하여 비효율적이었지만, 푸드트럭과 행사의 요구사항을 기반으로 최적의 매칭을 자동으로 수행하여 시간과 노력을 절감하는 효과를 가져올 수 있다.

#### 2) 수요와 공급:

특정 지역이나 시기에 푸드트럭이 과다하게 집중되거나, 반대로 부족한 경우가 종종 발생하였지만, 시스템을 통해 수요와 공급을 조정하여 과밀현상과 과소현상을 해결할 수 있다.

#### 3) 의사소통 문제:

기존의 행사 관계자와 푸드트럭 매니저 간의 소통이 중개인을 통한 방법이었기에 비효율적이고, 이러한 과정에서 정보가 왜곡되거나 잘못 전달될 가능성이 있지만, 푸드트럭 매니저는 행사의 상세 정보를 확인하며 자신이 직접 행사 참여 여부를 선택할 수 있고, 행사 관계자는 행사에 참여를 희망하는 푸드트럭의 리스트에서 직접 선별할 수 있기에 서로 간의 소통이 효율적으로 진행될 수 있다.

#### 4) 평판 관리:

과거의 성과 데이터를 체계적으로 수집하고 분석하지 않아 향후 의사결정에 활용하기 어려웠기에 리뷰와 평점을 통해 행사 관계자가 신뢰할 수 있는 푸드트럭을 선택할 수 있으며, 푸드트럭 매니저도 자신의 평판을 효과적으로 관리 할 수 있다.

#### 5) 개인 정보 보호:

중개인을 통한 매칭을 하였을 때 개인정보가 유출되거나 이를 악용할 수 있는 위험이 생겨나지만, 개인정보 보호를 위한 JWT 토큰을 발행하여 개인정보 유출 위험을 최소화할 수 있다.

## II. 프로젝트 수행내용

### 1. 프로젝트 추진전략 및 방법(체계)

#### 1) 프로젝트 추진 일정 및 방법

[표 1] 프로젝트 추진 일정

No	프로젝트 활동 내용	추진 일정															기간 (주)	추진 방법
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
1	관련 기술 조사																	안드로이드, IOS, 웹 서비스 개발에 사용하기 위한 플랫폼, 개발 도구 등 조사
2	서버 개발																	데이터베이스 구축 및 서버 CRUD API 작성
3	웹 개발																	스프링을 이용한 웹 서비스 개발
4	안드로이드 개발																	안드로이드 스튜디오를 이용해 안드로이드 서비스 개발
5	IOS 개발																	XCODE를 이용해 IOS 서비스 개발
6	프로젝트 피드백																	프로젝트 피드백 및 결점 보완

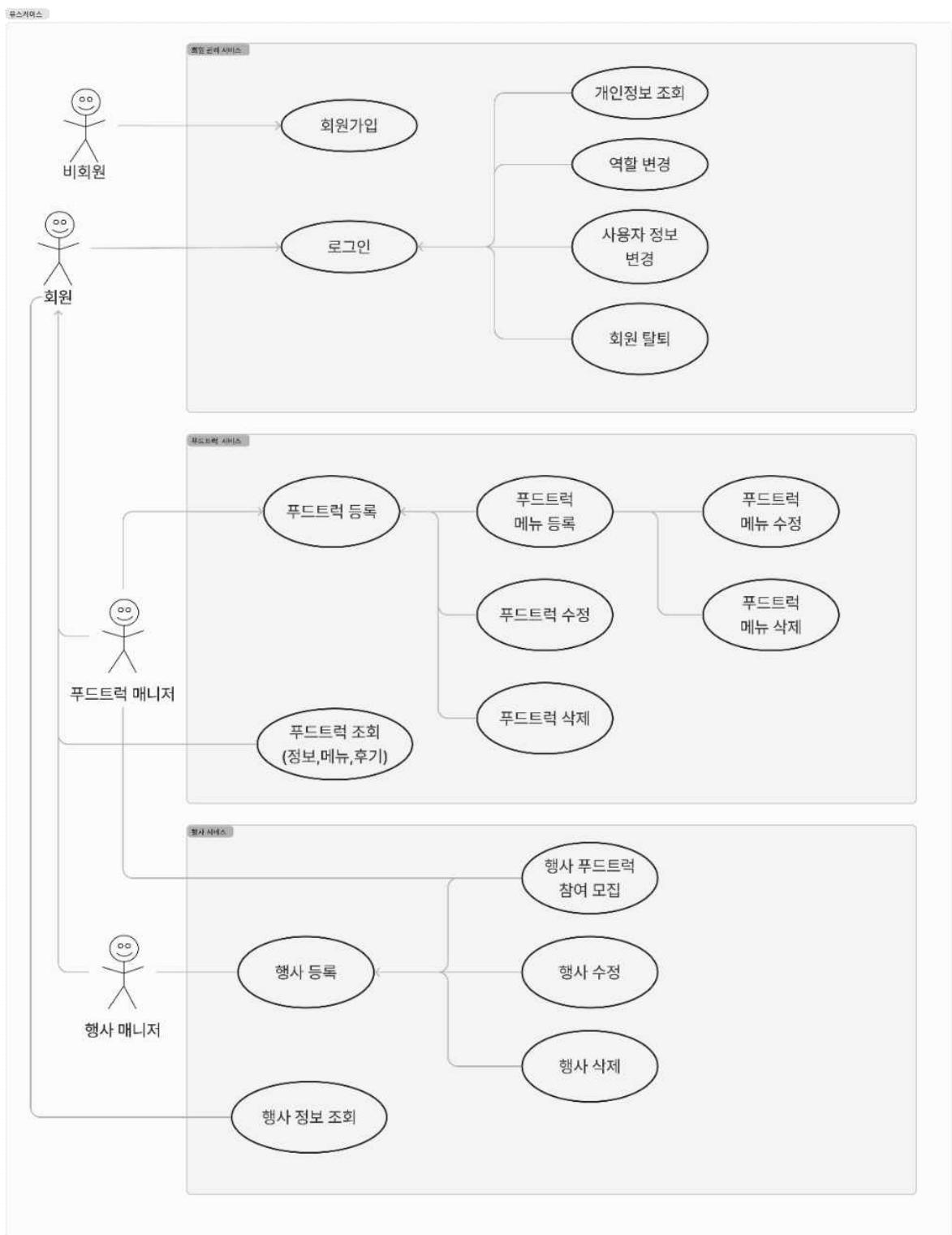
## 2) 프로젝트 추진 체계

[표 2] 프로젝트 추진 체계

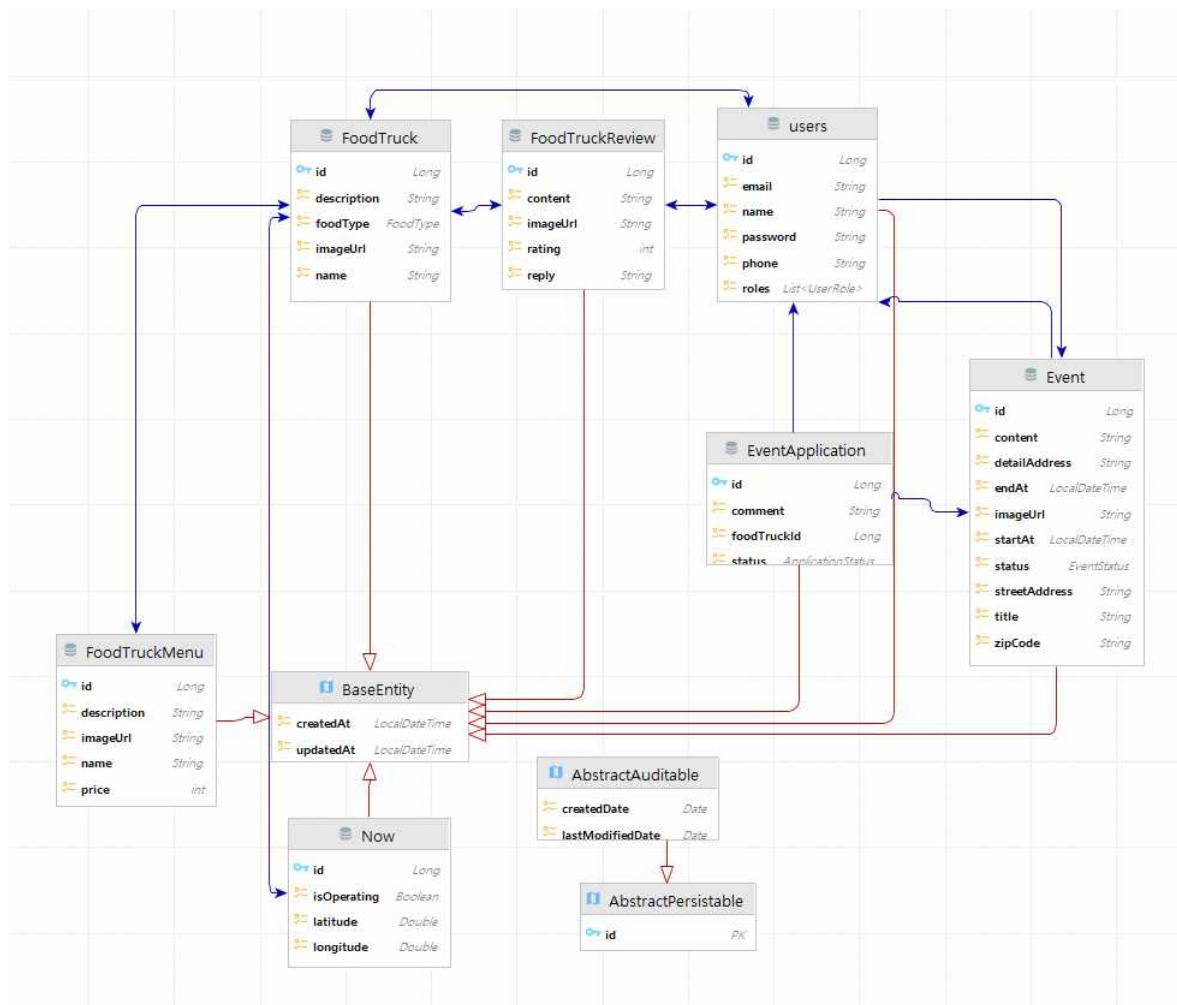
성명 / 학번	수행내용	역할
 김남주  20192336	<ul style="list-style-type: none"> <li>■ 백엔드 아키텍처 설계</li> <li>■ 프로젝트 관리 및 역할 분담</li> </ul>	팀장
 이종민  20193174	<ul style="list-style-type: none"> <li>■ 행사 관리자용 IOS 어플리케이션 개발</li> </ul>	팀원
 이태수  20193164	<ul style="list-style-type: none"> <li>■ 푸드트럭 사업자용 Android 어플리케이션 개발</li> </ul>	팀원
 오승현  20193183	<ul style="list-style-type: none"> <li>■ 푸드트럭 사업자용 기능 개발 및 홈페이지 개발</li> </ul>	팀원
 김민곤  20193172	<ul style="list-style-type: none"> <li>■ 행사 관리자용 기능 개발 및 홈페이지 개발</li> </ul>	팀원

## 2. 프로젝트 수행 과정(분석 및 설계 포함)

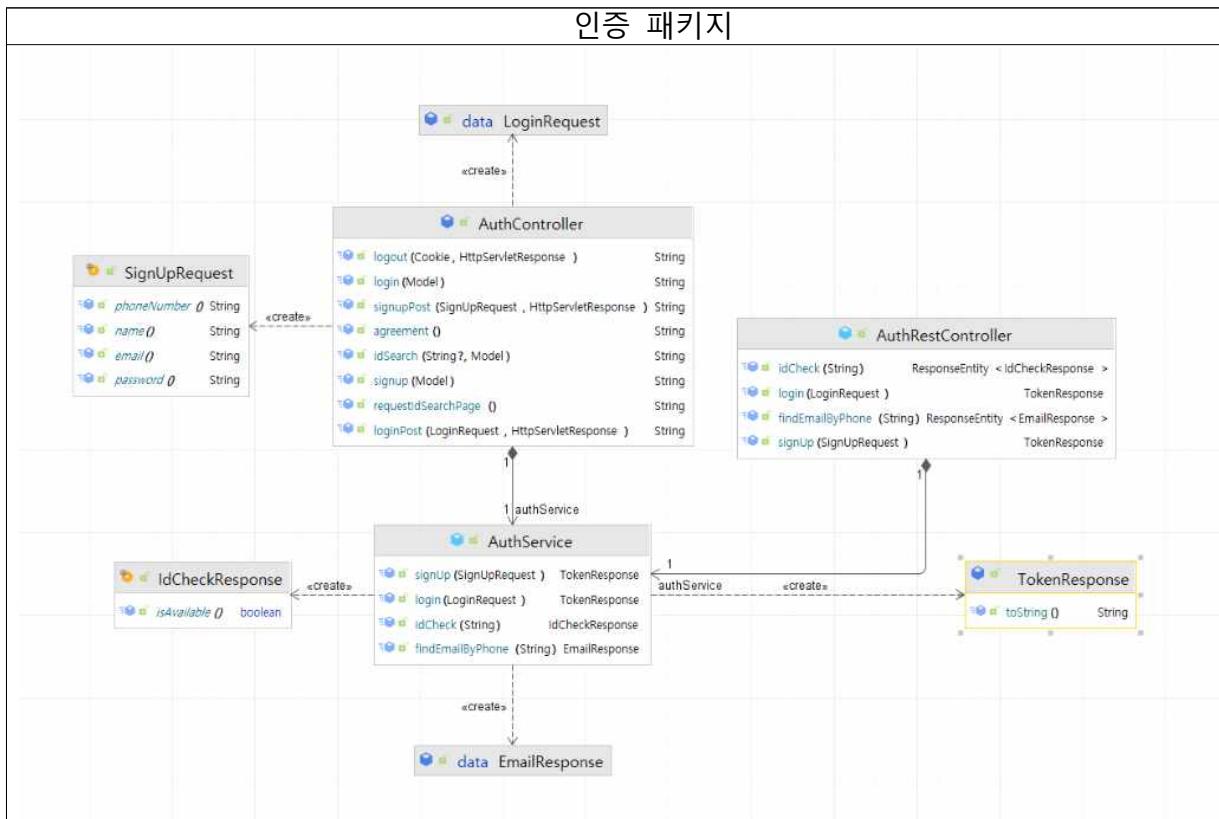
### 1) 유스케이스



## 2) ERD



### 3) 클래스 디어그램



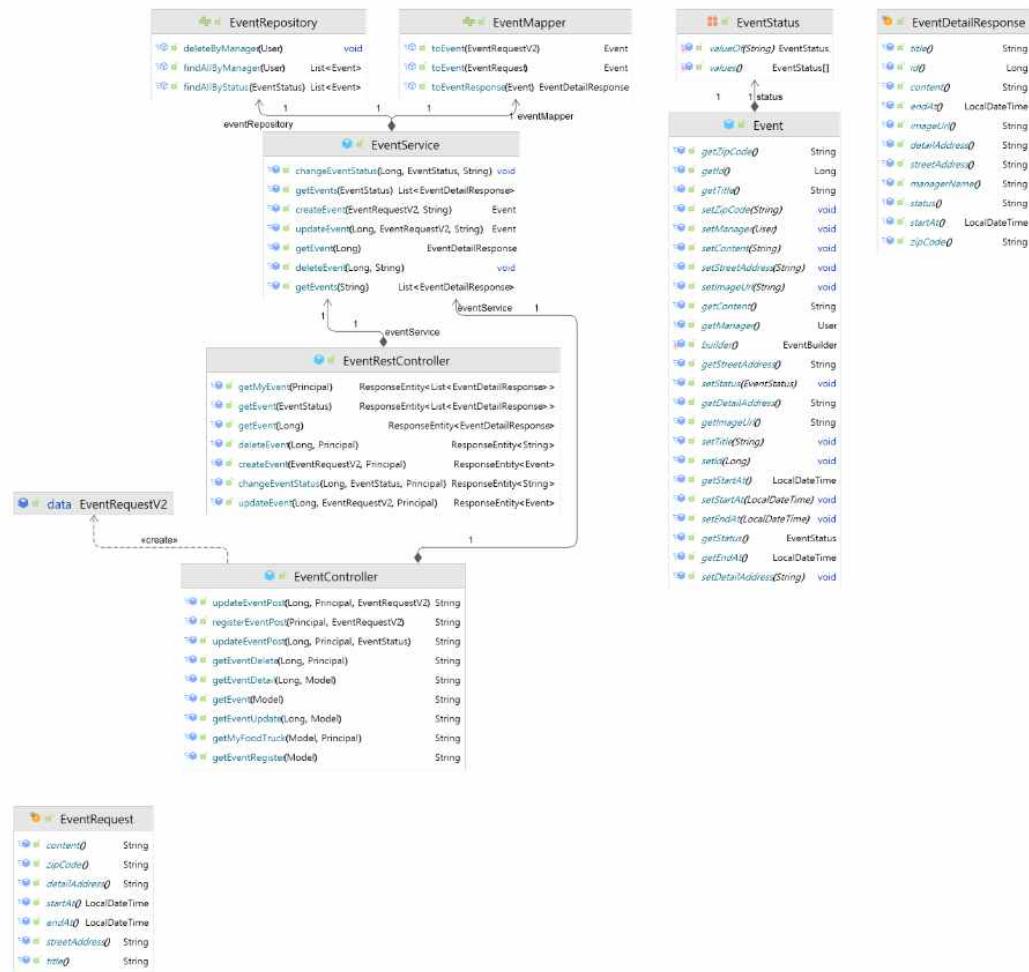
Q

Auth Controller의 경우에는 Spring Security 를 사용하기 위해 필터를 재정의 하는 방향으로 진행했습니다.

해당 기능을 사용하기 위해 UserDetails 와 UserDetailService를 상속받은 유저 클래스를 생성후 Spring Security FilterChain 을 상속받은 설정 클래스에서 해당 필터등을 빈으로 등록하게 됩니다.

저희 프로젝트가 WebMVC와 RESTful API를 사용하기 위해서 세션 또는 토큰 적용 등 하나를 선택해야 하는 상황이었는데, 핸드폰에서는 세션사용이 힘들기 때문에, JWT 방식으로 채택하였습니다. 따라서 웹 환경에서는 토큰을 보관해야 했는데, 이를 해결하기 위해 쿠키에 토큰을 보관하는 방식으로 개발을 진행했습니다.

## 행사 관련 패키지



DTO (요청, 반환) - 컨트롤러(WEB, RESTful) - 서비스 - 레포지터리(저장소) 구조로 이루어져 있으며 Spring Data JPA를 활용하여 코드를 간소화 시켜 서비스와 컨트롤러 등 어플리케이션 동작에 필요한 로직 작성에 집중하였다.

EventRepository와 EventMapper를 이용하여 데이터를 처리하는 메서드를 가진 EventService 클래스를 작성하고 해당 클래스를 이용해 EventController와 EventRestController를 작성하고 실질적인 기능 수행을 담당한다.

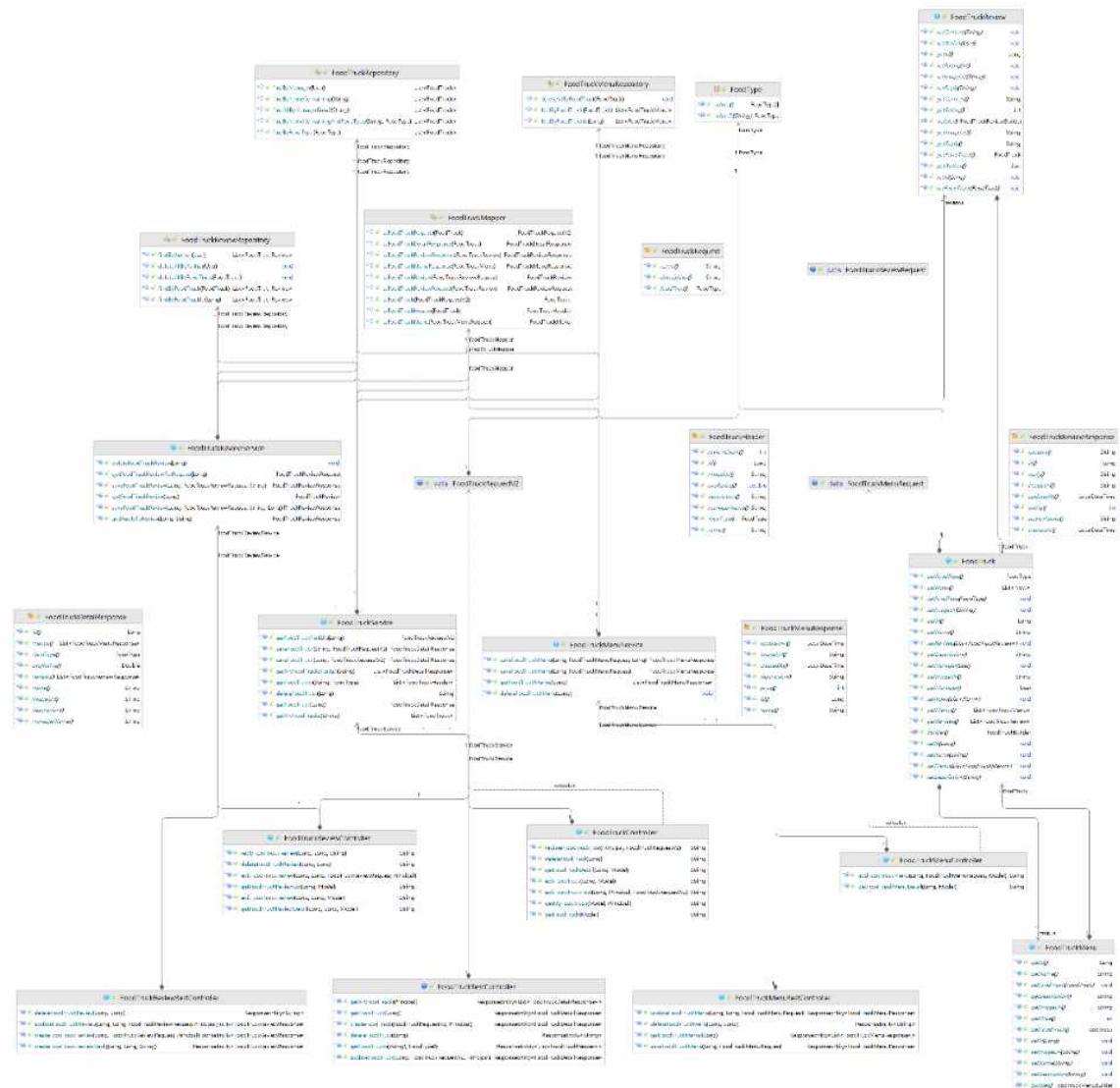
## 행사 신청 관련 패키지



DTO (요청, 반환) - 컨트롤러(WEB, RESTful) - 서비스 - 레포지터리(저장소) 구조로 이루어져 있으며 Spring Data JPA를 활용하여 코드를 간소화 시켜 서비스와 컨트롤러 등 어플리케이션 동작에 필요한 로직 작성에 집중하였다.

EventApplicationRepository와 EventApplicationMapper를 이용하여 데이터를 처리하는 메서드를 가진 EventApplicationService 클래스를 작성하고 해당 클래스를 이용해 EventApplicationController와 EventApplicationRestController를 작성하고 실질적인 기능 수행을 담당한다.

## 푸드트럭 관련 패키지



푸드트럭의 경우 푸드트럭 정보, 푸드트럭 메뉴, 푸드트럭 후기의 기능이 들어가기 때문에, 상당히 복잡한 패키지 구조를 가지고 있다.

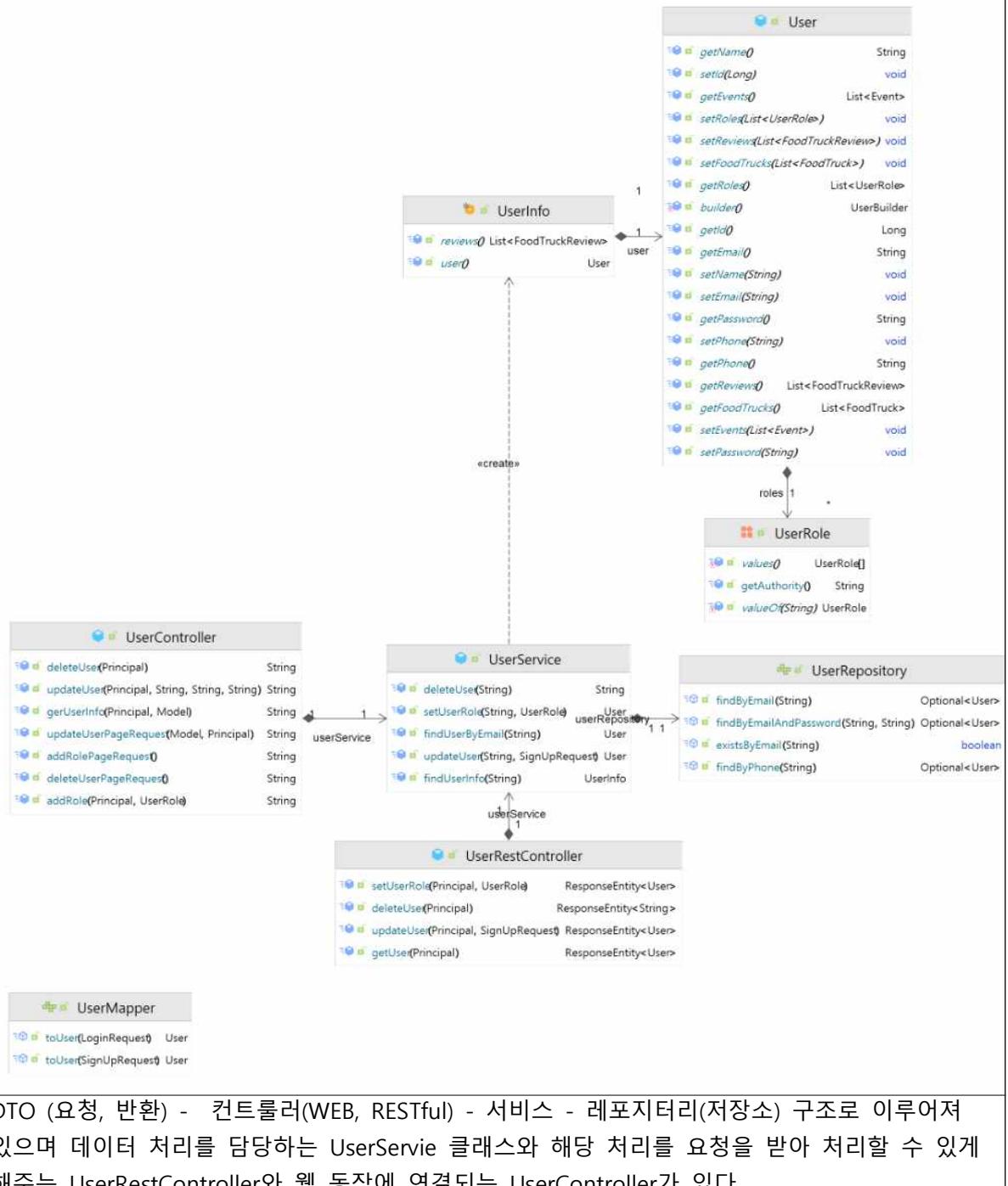
하지만 대부분 비슷한 구조로 이루어져 있는데

DTO (요청, 반환) - 컨트롤러(WEB, RESTful) - 서비스 - 레포지터리(저장소) 구조로 이루어져 있다.

DTO의 잣은 코드 수정을 대비해서 모델 매핑 라이브러리인 MapStruct를 적용하여 모델 변환을 원활하게 진행하였다. 덕분에 코드가 수정되더라도 매핑 메소드를 수정하여 바로바로 사용이 가능하다.

Spring Data JPA를 활용하여 코드를 많이 간소화 시켰고, 덕분에 서비스 로직에 집중할 수 있다.

## 유저 관련 패키지



## 1) 회원가입 기능

### (1) 분석 및 설계

서비스를 사용하기 위해서는 회원가입이 선행 되어야 합니다. 따라서 비회원 대상으로 진행하는 유스케이스이며, 회원가입 진행 시 아이디 중복체크, 비밀번호 암호화 등 여러 처리가 들어가야 합니다.

### (2) 구현

Server
AuthRestController.java
<pre>/*  * 회원가입  *  * @param signUpRequest 회원가입 요청 정보  * @return String jwt 토큰  */ public TokenResponse signUp(SignUpRequest signUpRequest) {     if (userRepository.existsByEmail(signUpRequest.getEmail())) throw new CustomException(CustomErrorCode.ALREADY_EXIST_USER_ID);      User user = userMapper.toUser(signUpRequest);     List&lt;UserRole&gt; roles = new ArrayList&lt;&gt;();     roles.add(UserRole.USER);     user.setRoles(roles);     user.setPassword(passwordEncoder.encode(user.getPassword()));     User savedUser = userRepository.save(user);      return new TokenResponse(jwtTokenProvider.createToken(savedUser.getEmail(),     savedUser.getRoles())); }</pre>
AuthController.java
<pre>/*  * 회원가입 처리  *  * @param signUpRequest 회원가입 정보  * @param response HttpServletRespose  * @return 흠 화면  */ @PostMapping("/signup") fun signupPost(     signUpRequest: SignUpRequest,     response: HttpServletRespose ): String {     // 회원가입 후 자동 로그인     // 쿠키 저장 시 토큰을 URL 인코딩(URLEncoder.encode)하여 저장     val token = authService.signUp(signUpRequest).token     val encodedToken = URLEncoder.encode(TOKEN_PREFIX + token, StandardCharsets.UTF_8)     val cookie = Cookie(AUTHORIZATION_HEADER, encodedToken)     cookie.maxAge = 60 * 30 // 30분     cookie.path = "/"     response.addCookie(cookie)      return "redirect:/" }</pre>
AuthRestController.java
<pre>@PostMapping("/signup")</pre>

```

@Operation(summary = "회원가입", description = "입력된 품을 바탕으로 로그인을  
진행합니다.<br>사용 전 email-check를 통해 이메일이 사용가능 여부를 확인바랍니다.")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "회원가입 성공", content =
@Content(schema = @Schema(implementation = TokenResponse.class))),
    @ApiResponse(responseCode = "400", description = "회원가입 실패", content =
@Content(schema = @Schema(implementation = CustomErrorResponse.class)))
})
public TokenResponse signUp(
    @Parameter(name = "로그인 객체", description = "회원가입 요청 정보") @RequestBody
SignUpRequest signUpRequest) {
    return authService.signUp(signUpRequest);
}

@Schema(description = "로그인 요청 DTO")
data class LoginRequest(
    @field:Schema(description = "이메일", example = "test1@test.com")
    val email: String,
    @field:Schema(description = "비밀번호", example = "test1")
    val password: String
)

```

Web
profile_edit.html
<pre> &lt;!-- 회원가입 폼 추가 --&gt; &lt;form method="post" th:action="@{/auth/signup}" th:object="\${signUpRequest}"&gt;     &lt;div class="form-group"&gt;         &lt;label for="email"&gt;이메일:&lt;/label&gt;         &lt;input class="form-control" id="email" name="email" required type="email"&gt;     &lt;/div&gt;     &lt;div class="form-group"&gt;         &lt;label for="password"&gt;비밀번호:&lt;/label&gt;         &lt;input class="form-control" id="password" name="password" required type="password"&gt;     &lt;/div&gt;     &lt;div class="form-group"&gt;         &lt;label for="name"&gt;이름:&lt;/label&gt;         &lt;input class="form-control" id="name" name="name" required type="text"&gt;     &lt;/div&gt;     &lt;div class="form-group"&gt;         &lt;label for="phone"&gt;전화번호:&lt;/label&gt;         &lt;input class="form-control" id="phone" name="phone" required type="text"&gt;     &lt;/div&gt;     &lt;button class="btn btn-primary btn-block" type="submit"&gt;회원가입&lt;/button&gt; &lt;/form&gt; </pre>

Copyright © Code Miracle Corp. All Rights Reserved.

IOS
UserService.swift

```

import Foundation
import Alamofire

struct UserService {
    static let shared = UserService()

    func checkEmailDuplication(_ email: String, completion: @escaping (Bool?, Error?) -> Void) {
        guard let encodedEmail = email.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed) else {
            print("이메일 인코딩 실패")
            completion(nil, NetworkError.responseError)
            return
        }

        NetworkService.shared.performRequest("/auth/email-check?email=\(encodedEmail)", method: .post) { (response: EmailCheckResponse?, error) in
            completion(response?.isAvailable, error)
        }
    }

    func login(loginRequest: LoginRequestDTO, completion: @escaping (String?, Error?) -> Void) {
        NetworkService.shared.performRequest("/auth/login", method: .post, parameters: loginRequest) { (response: LoginResponse?, error) in
            completion(response?.token, error)
        }
    }

    func signUp(user: SignUpRequestDTO, completion: @escaping (String?, Error?) -> Void) {
        NetworkService.shared.performRequest("/auth/signup", method: .post, parameters: user) { (response: SignUpResponseDTO?, error) in
            if let response = response {
                completion(response.token, nil)
            } else {
                completion(nil, error)
            }
        }
    }
}

```

```

        }
    }

UserRegistrationViewModel.swift
import SwiftUI
import Combine

class UserRegistrationViewModel: ObservableObject {
    @Published var email: String = ""
    @Published var isEmailAvailable: Bool = false
    @Published var emailCheckResult: String? = nil

    @Published var password: String = ""
    @Published var phoneNumber: String = ""
    @Published var name: String = ""

    @Published var showAlert: Bool = false
    @Published var alertMessage: String = ""
    @Published var isLoading: Bool = false

    private var cancellables: Set = []

    func setEmail(_ email: String) {
        self.email = email
    }

    func setPassword(_ password: String) {
        self.password = password
    }

    func setPhoneNumber(_ phoneNumber: String) {
        self.phoneNumber = phoneNumber
    }

    func setName(_ name: String) {
        self.name = name
    }

    func checkEmailDuplication(_ email: String, completion: @escaping (Bool?) -> Void) {
        UserService.shared.checkEmailDuplication(email) { isAvailable, error in
            if let error = error {
                // 에러 처리
                print(error.localizedDescription)
                self.alertMessage = "이메일 중복 확인 실패(서버 에러)"
                completion(nil) // 에러 발생 시 nil 반환
            } else if let isAvailable = isAvailable {
                // 결과 반환
                if isAvailable {
                    self.alertMessage = "사용 가능한 이메일입니다."
                } else {
                    self.alertMessage = "이미 사용중인 이메일입니다."
                }
                completion(isAvailable)
            } else {
                // 예상치 못한 상황 처리
            }
        }
    }
}

```

```

        completion(nil)
    }
}

func signUp(completion: @escaping (Bool) -> Void) {
    isLoading = true
    let requestDto = SignUpRequestDTO(name: name, email: email, phone: phoneNumber, password: password)

    UserService.shared.signUp(user: requestDto) { [weak self] token, error in
        DispatchQueue.main.async {
            self?.isLoading = false
            if let token = token {
                self?.alertMessage = "회원가입이 완료되었습니다."
                completion(true)
            } else {
                self?.alertMessage = error?.localizedDescription ?? "회원가입 중 오류가 발생했습니다."
                completion(false)
            }
            self?.showAlert = true
        }
    }
}

```

UserReponse.swift

```

import Foundation

// MARK: - User
struct UserResponse: Codable {
    let createdAt, updatedAt: String?
    let roles: [String]?
    let id: Int?
    let email, password, name, phone: String?
    let reviews: [ReviewResponse]?
}

// MARK: - Review
struct ReviewResponse: Codable {
    let createdAt, updatedAt: String?
    let id: Int?
    let content: String?
    let rating: Int?
    let imageURL: String?
    let reply: String?

    enum CodingKeys: String, CodingKey {
        case createdAt, updatedAt, id, content, rating
        case imageURL = "imageUrl"
        case reply
    }
}

struct LoginResponse: Codable {
    let token: String?
}

struct EmailCheckResponse: Codable {
    let isAvailable: Bool?
}

```

```
}

struct SignUpResponse: Decodable {
    let success: Bool
}

struct SignUpRequestDTO: Encodable {
    let name: String
    let email: String
    let phone: String
    let password: String
}

struct SignUpResponseDTO: Decodable {
    let token: String
}
```

SignUpRequestDTO.swift

The image consists of two side-by-side screenshots of a mobile application interface. Both screenshots show the time as 4:46 on the left and 4:47 on the right. The top portion of both screenshots shows a navigation bar with a back arrow icon.

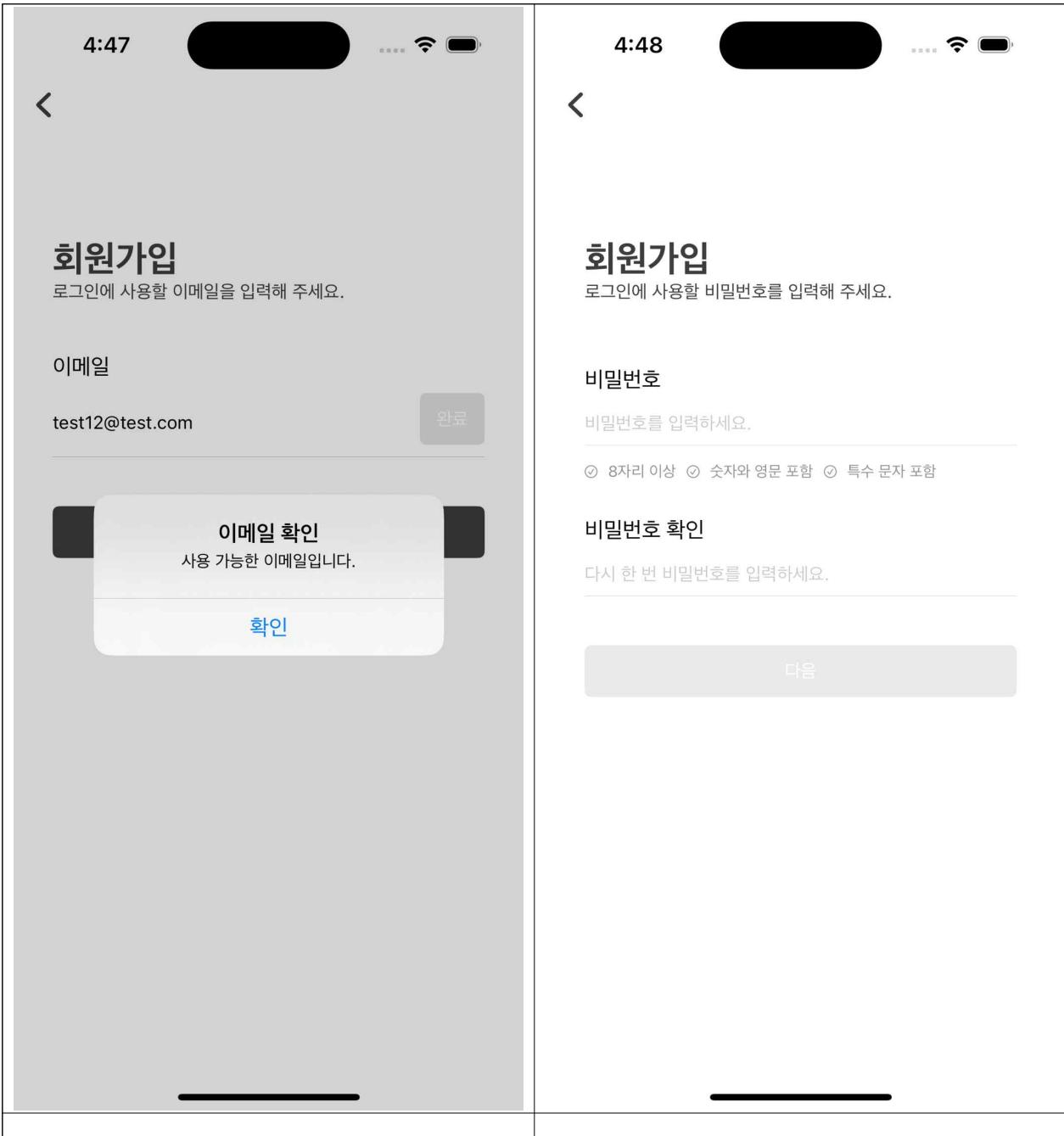
**Screenshot 1 (Left):**

- Header:** 환영합니다.
- Text:** 여러분이 원하시는 축제와 푸드트럭을 찾아보세요!
- Buttons:**  and

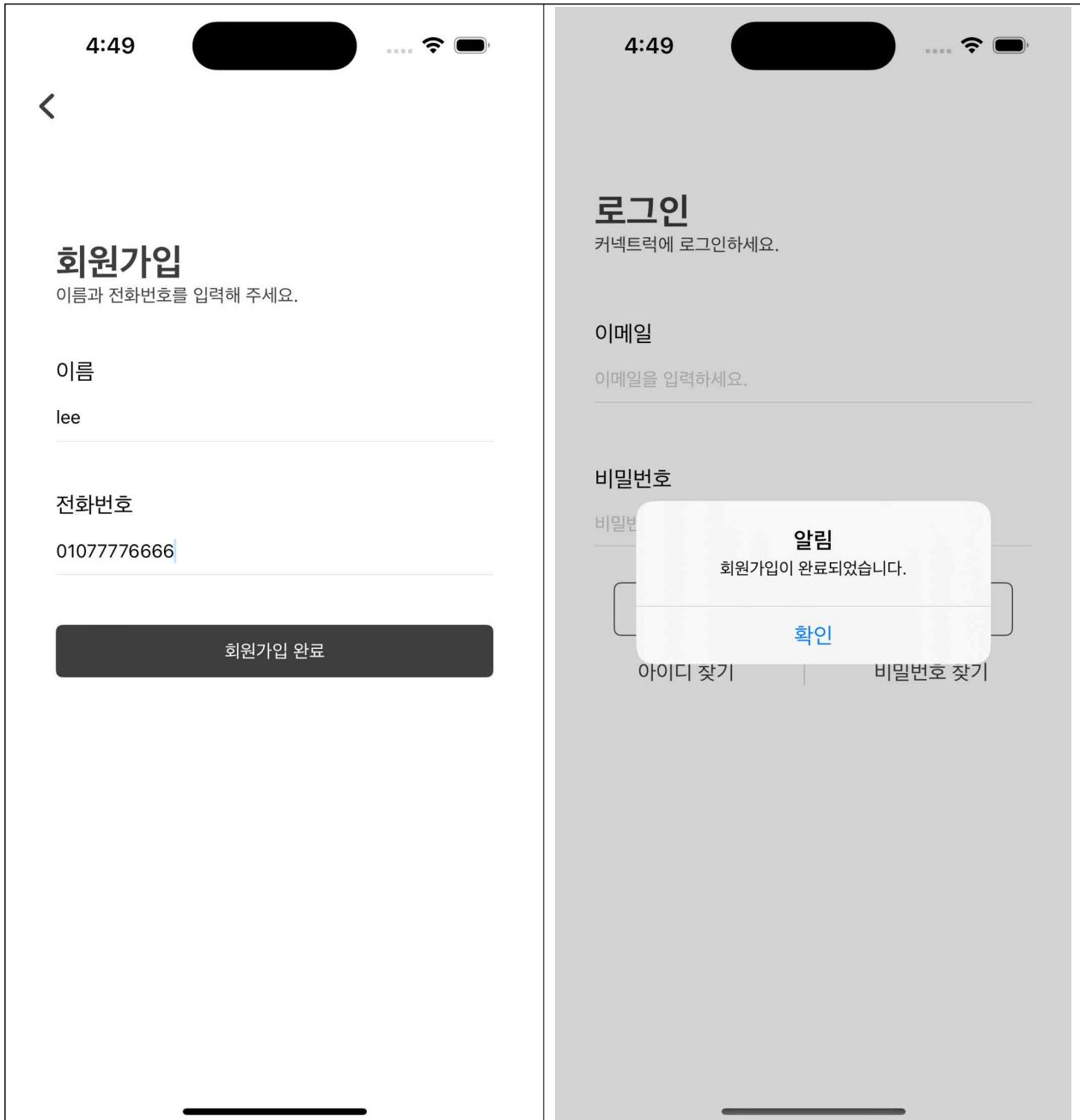
**Screenshot 2 (Right):**

- Title:** 회원가입
- Text:** 로그인에 사용할 이메일을 입력해 주세요.
- Form:** An input field labeled "아이디(이메일)을 입력하세요." with a "확인" button to its right.
- Buttons:**

<p>4:47</p> <p>&lt;</p> <h2>회원가입</h2> <p>로그인에 사용할 이메일을 입력해 주세요.</p> <p>이메일</p> <p>test@test.com</p> <p>확인</p> <p>다음</p>	<p>4:47</p> <p>&lt;</p> <h2>회원가입</h2> <p>로그인에 사용할 이메일을 입력해 주세요.</p> <p>이메일</p> <p>test@test.com</p> <p>완료</p> <div data-bbox="853 735 1329 921"><p>이메일 확인 이미 사용중인 이메일입니다.</p><p>확인</p></div>
---	--



<p>4:48</p> <p>&lt;</p> <h2>회원가입</h2> <p>로그인에 사용할 비밀번호를 입력해 주세요.</p> <p>비밀번호</p> <p>••••••••••</p> <p>⑤ 8자리 이상 ⑥ 숫자와 영문 포함 ⑦ 특수 문자 포함</p> <p>비밀번호 확인</p> <p>•••••••••• </p> <p>⑤ 비밀번호가 일치합니다</p> <p>다음</p>	<p>4:48</p> <p>&lt;</p> <h2>회원가입</h2> <p>이름과 전화번호를 입력해 주세요.</p> <p>이름</p> <p>이름을 입력하세요.</p> <p>전화번호</p> <p>전화번호를 입력하세요.</p> <p>회원가입 완료</p>
--	--



Android
object MySharedPreferences.kt
/** *SharedPreferences를 사용하여 사용자 정보를 저장하는 싱글톤 객체 */
object MySharedPreferences {
private const val USER_INFO ="user_info"
private const val USER_EMAIL ="user_email"
private const val USER_PASSWORD ="user_password"
private const val JWT_TOKEN ="jwt_token"
object MySharedPreferences.kt
/** *사용자 정보를 저장하는 함수 */
@param context Context

```

    *@param email 사용자 이메일
    *@param password 사용자 비밀번호
    */
    fun setUserInfo(context:Context,email:String,password:String){
        val sharedpreferences:SharedPreferences =
            context.getSharedPreferences(USER_INFO,Context.MODE_PRIVATE)
        val editor:SharedPreferences.Editor =sharedpreferences.edit()
        editor.putString(USER_EMAIL,email)
        editor.putString(USER_PASSWORD,password)
        editor.apply()
    }
}

object MySharedPreferences.kt

/***
 *JWT 토큰을 저장하는 함수
 *@param context Context
 *@param token JWT 토큰
 */
fun setToken(context:Context,token:String){
    val sharedpreferences:SharedPreferences =
        context.getSharedPreferences(USER_INFO,Context.MODE_PRIVATE)
    val editor:SharedPreferences.Editor =sharedpreferences.edit()
    editor.putString(JWT_TOKEN,token)
    editor.apply()
}
}

object MySharedPreferences.kt

/***
 *JWT 토큰을 반환하는 함수
 *@param context Context
 *@return JWT 토큰
 */
fun getToken(context:Context):String {
    val sharedpreferences:SharedPreferences =
        context.getSharedPreferences(USER_INFO,Context.MODE_PRIVATE)
    return sharedpreferences.getString(JWT_TOKEN,"")!!
}

object MySharedPreferences.kt

/***
 *사용자 이메일을 반환하는 함수
 *@param context Context
 *@return 사용자 이메일
 */
fun getUserEmail(context:Context):String {
    val sharedpreferences:SharedPreferences =
        context.getSharedPreferences(USER_INFO,Context.MODE_PRIVATE)
    return sharedpreferences.getString(USER_EMAIL,"")!!
}

object MySharedPreferences.kt

/***
 *사용자 비밀번호를 반환하는 함수
 *@param context Context
 *@return 사용자 비밀번호
 */
fun getPassword(context:Context):String {
    val sharedpreferences:SharedPreferences =
        context.getSharedPreferences(USER_INFO,Context.MODE_PRIVATE)
    return sharedpreferences.getString(USER_PASSWORD,"")!!
}

```

```

object MySharedPreferences.kt

/**
 *사용자 정보를 삭제(로그아웃)하는 함수
 *@param context Context
 */
fun clearUserInfo(context:Context){
    val sharedpreferences:SharedPreferences =
        context.getSharedPreferences(USER_INFO,Context.MODE_PRIVATE)
    val editor:SharedPreferences.Editor =sharedpreferences.edit()
    editor.clear()
    editor.apply()
}

interface AuthApi.kt

@POST("/api/auth/signup")
suspend fun signUp(
    @Body signUpRequest:SignUpRequest
):Response<TokenResponse>
interface AuthApi.kt

@POST("/api/auth/findEmailByPhone")
suspend fun findEmailByPhone(
    @Query("phone")phone:String
):Response<EmailResponse>
interface AuthApi.kt

@POST("/api/auth/email-check")
suspend fun emailDuplicateCheck(
    @Query("email")email:String
):Response<IdCheckResponse>
class LoginFragment.kt

/**
 *현재이메일과비밀번호값을사용하여LoginRequest객체를생성합니다.
 *@return이메일과비밀번호가포함된LoginRequest객체를반환합니다.
 */
fun getLoginRequest():LoginRequest{
    returnLoginRequest(
        email=email.value!!,
        password=password.value!!
    )
}

class AuthViewModel.kt

/**
 *제공된사용자정보를사용하여회원가입을시도합니다.
 *authApi를사용하여회원가입요청을보내고,가입성공여부를업데이트합니다.
 */
fun signUp(){
    val signUpRequest=SignUpRequest(
        email=email.value!!,
        password=password.value!!,
        name=name.value!!,
        phoneNumber=phoneNumber.value!!
    )

    viewModelScope.launch{
        val response=authApi.signUp(signUpRequest)
        _signUpSuccess.postValue(response.isSuccessful)
    }
}

```

### class AuthViewModel.kt

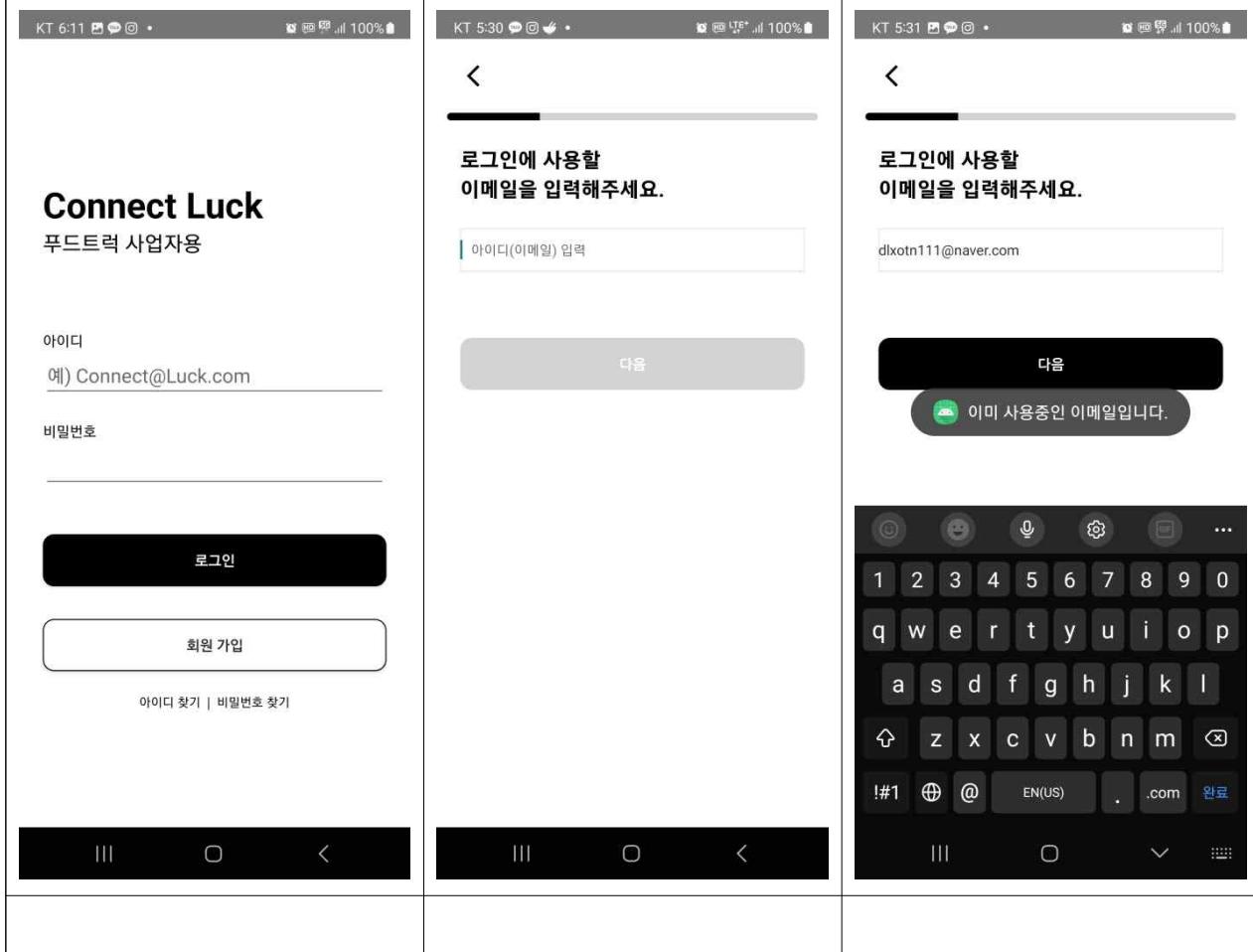
```
@HiltViewModel
class AuthViewModel @Inject constructor(
    private val authApi: AuthApi
) : ViewModel() {
    val email = MutableLiveData<String>("")
    val password = MutableLiveData<String>("")
    val name = MutableLiveData<String>("")
    val phoneNumber = MutableLiveData<String>("")
    val profileImageUrl = MutableLiveData<String>("")

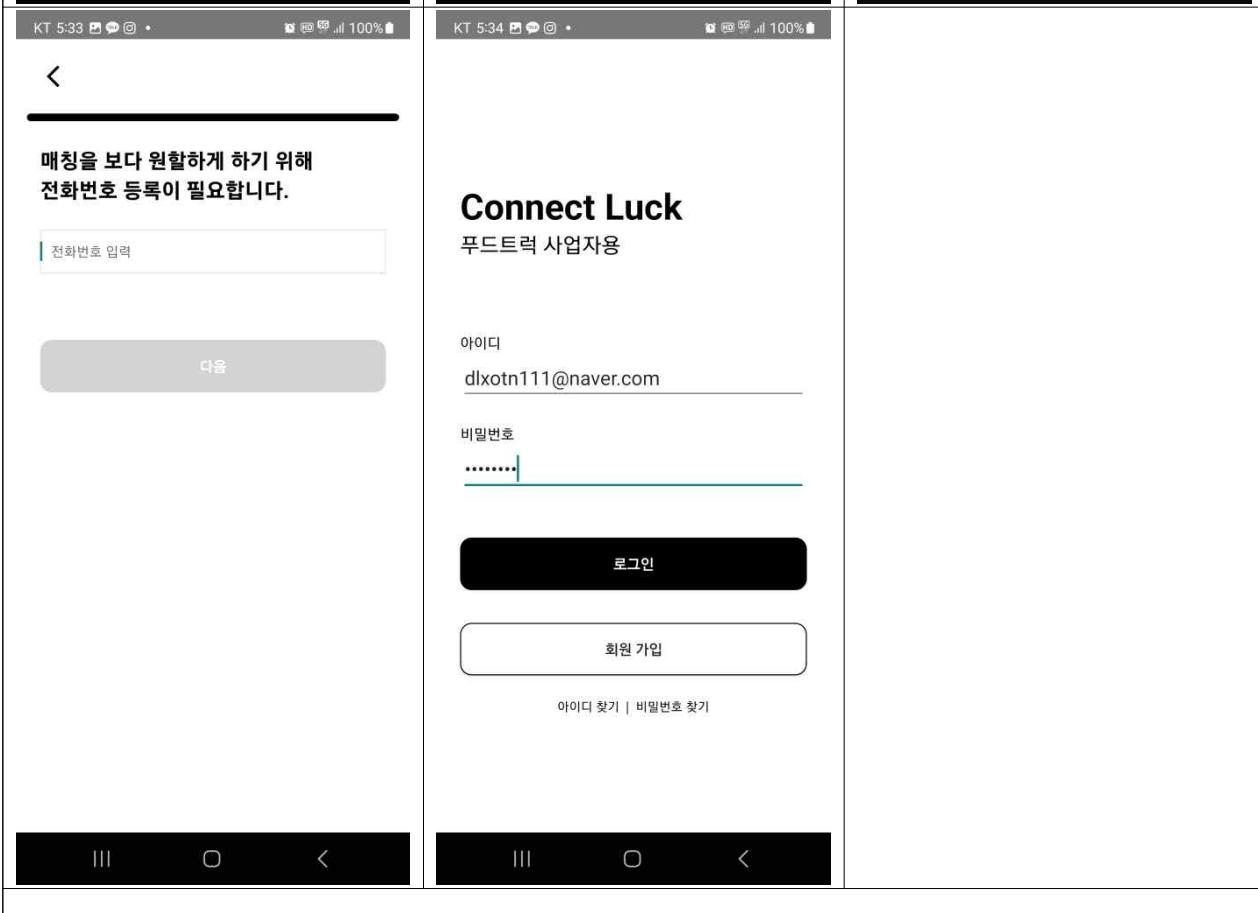
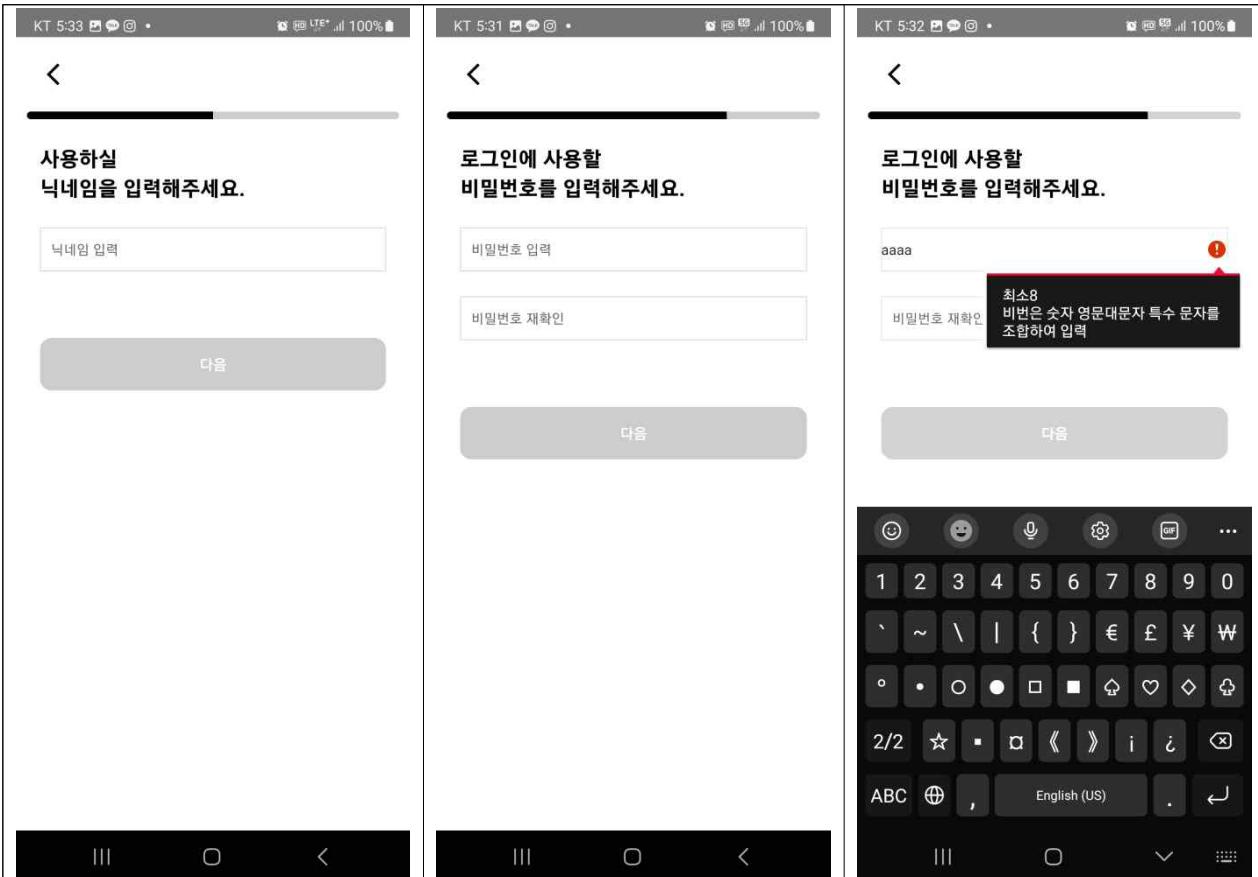
    private val _signUpSuccess = MutableLiveData<Boolean>()
    val signUpSuccess: LiveData<Boolean> = _signUpSuccess
```

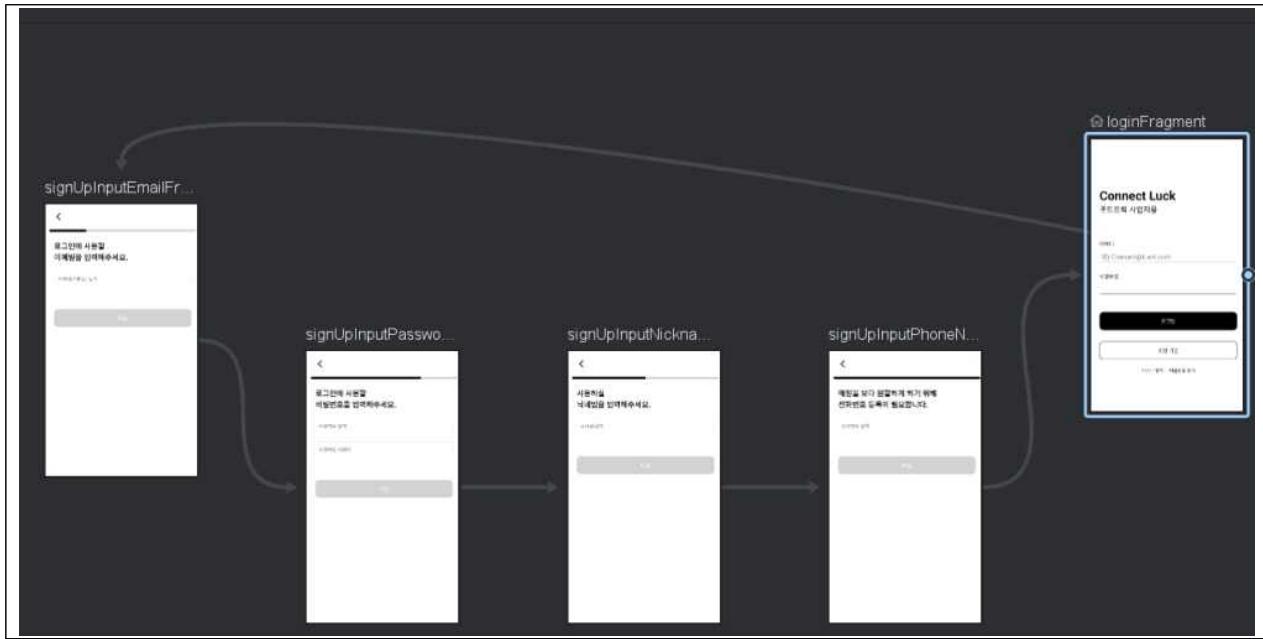
### class SignUpInputNicknameFragment.kt

```
// Saripaar` 라이브러리를 사용하여 유효성 검사
@Length(min = 2,max = 100,message ="최소2")//길이
@NotEmpty(message ="입력해주세요")//필수입력
EditText nickname;
```

회원가입 절차를 거치면서 나온 데이터들을 **class AuthViewModel.kt** 변수에 값 저장 후 마지막 회원 가입 버튼을 누르면 authApi.signUp(signUpRequest)를 호출







## 2) 로그인 기능

### (1) 분석 및 설계

로그인을 진행 할 때에는 ID, PW를 사용하여 로그인을 진행합니다. 로그인을 성공하면 JWT를 발급 받고, RESTful 의 경우 헤더에 토큰을 담아서, Web의 경우 쿠키에 URL 인코딩을 진행하여 쿠키에 보관하게 됩니다. 토큰의 유효시간은 30분이며, 쿠키 수명도 30분으로 설정합니다

### (2) 구현

Server
TokenResponse.java
<pre>class TokenResponse{     val token: String } {     override fun toString(): String {         return "TokenResponse(token='\$token')"     } }</pre>
AuthService.java
<pre>/**  * 로그인  *  * @param loginRequest 로그인 요청 정보  * @return User 엔티티  */ public TokenResponse login(LoginRequest loginRequest) {     Optional&lt;User&gt; user = userRepository.findByEmail(loginRequest.getEmail());      if (user.isEmpty()) throw new CustomException(CustomErrorCode.EMAIL_NOT_FOUND);     if (!passwordEncoder.matches(loginRequest.getPassword(), user.get().getPassword()))         throw new CustomException(CustomErrorCode.PASSWORD_NOT_MATCH);      return new TokenResponse(jwtTokenProvider.createToken(user.get().getEmail(),     user.get().getRoles())); }</pre>
AuthController.java
<pre>@PostMapping("/login") fun loginPost(     loginRequest: LoginRequest,     response: HttpServletResponse ): String {     // 로그인     val token = authService.login(loginRequest).token     val encodedToken = URLEncoder.encode(TOKEN_PREFIX + token, StandardCharsets.UTF_8)      // 쿠키에 토큰 저장     val cookie = Cookie(AUTHORIZATION_HEADER, encodedToken)     cookie.path = "/"     response.addCookie(cookie)      // 홈 화면으로 리다이렉트     return "redirect:/" }</pre>
AuthRestController.java
<pre>@PostMapping("/login") @Operation(summary = "로그인", description = "입력된 이메일과 비밀번호를 사용하여 로그인을 진행합니다.&lt;br&gt;로그인 성공 시 JWT 토큰을 반환합니다.&lt;br&gt;인증이 필요한 요청 시 헤더에 Authorization</pre>

```

Bear {JWT}를 포함하여 요청을 보내야 합니다.<br><b>주의: Auth 관련 API 호출 시 Header 부분이
비어 있어야 합니다.</b>")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "로그인 성공", content =
@Content(schema = @Schema(implementation = TokenResponse.class))),
    @ApiResponse(responseCode = "400", description = "로그인 실패", content =
@Content(schema = @Schema(implementation = CustomErrorResponse.class)))
})
public TokenResponse login(
    @Parameter(description = "로그인 요청 정보") @RequestBody LoginRequest loginRequest)
{
    return authService.login(loginRequest);
}

```

Web
<pre> &lt;!-- 로그인 폼 추가 --&gt; &lt;form method="post" th:action="@{/auth/login}" th:object="\${loginRequest}"&gt;     &lt;div class="form-group"&gt;         &lt;label for="email"&gt;이메일:&lt;/label&gt;         &lt;input class="form-control" id="email" name="email" required type="text"&gt;     &lt;/div&gt;     &lt;div class="form-group"&gt;         &lt;label for="password"&gt;비밀번호:&lt;/label&gt;         &lt;input class="form-control" id="password" name="password" required type="password"&gt;     &lt;/div&gt;     &lt;button class="btn btn-primary btn-block" type="submit"&gt;로그인&lt;/button&gt; &lt;/form&gt; </pre>

IOS
<pre> import Foundation import Alamofire  struct UserService {     static let shared = UserService()      func checkEmailDuplication(_ email: String, completion: @escaping (Bool?, Error?) -&gt; Void)     {         guard let encodedEmail = email.addingPercentEncoding(withAllowedCharacters: .urlQueryAllowed) else {             print("이메일 인코딩 실패")             completion(nil, NetworkError.responseError)             return         }     } } </pre>

```

    }

    NetworkService.shared.performRequest("/auth/email-check?email=\(encodedEmail)", method: .post) { (response: EmailCheckResponse?, error) in
        completion(response?.isAvailable, error)
    }
}

func login(loginRequest: LoginRequestDTO, completion: @escaping (String?, Error?) -> Void) {
    NetworkService.shared.performRequest("/auth/login", method: .post, parameters: loginRequest) { (response: LoginResponse?, error) in
        completion(response?.token, error)
    }
}

func signUp(user: SignUpRequestDTO, completion: @escaping (String?, Error?) -> Void) {
    NetworkService.shared.performRequest("/auth/signup", method: .post, parameters: user) { (response: SignUpResponseDTO?, error) in
        if let response = response {
            completion(response.token, nil)
        } else {
            completion(nil, error)
        }
    }
}
}

```

UserResponse.swift

```

import Foundation

// MARK: - User
struct UserResponse: Codable {
    let createdAt, updatedAt: String?
    let roles: [String]?
    let id: Int?
    let email, password, name, phone: String?
    let reviews: [ReviewResponse]?
}

// MARK: - Review
struct ReviewResponse: Codable {
    let createdAt, updatedAt: String?
    let id: Int?
    let content: String?
    let rating: Int?
    let imageURL: String?
    let reply: String?

    enum CodingKeys: String, CodingKey {
        case createdAt, updatedAt, id, content, rating
        case imageURL = "imageUrl"
        case reply
    }
}

struct LoginResponse: Codable {
    let token: String?
}

```

```

struct EmailCheckResponse: Codable {
    let isAvailable: Bool?
}

struct SignUpResponse: Decodable {
    let success: Bool
}

import Foundation

struct LoginRequestDTO: Codable {
    var email: String
    var password: String
}

import SwiftUI
import Combine

class LoginViewModel: ObservableObject {
    static let shared = LoginViewModel()

    @Published var email: String = ""
    @Published var password: String = ""
    @Published var alertMessage: String = ""
    @Published var showAlert: Bool = false
    @Published var isLoading: Bool = false
    @Published var isLoggedIn: Bool = false
    @Published var accessToken: String? = nil

    private var cancellables: Set = []

    private init() {}

    func login(completion: @escaping (Bool) -> Void) {
        guard !email.isEmpty, !password.isEmpty else {
            self.alertMessage = "이메일과 비밀번호를 입력해 주세요."
            self.showAlert = true
            completion(false)
            return
        }

        self.isLoading = true

        let loginRequestDTO = LoginRequestDTO(email: email, password: password)

        UserService.shared.login(loginRequest: loginRequestDTO) { [weak self] token, error in
            DispatchQueue.main.async {
                self?.isLoading = false
                if let token = token {
                    self?.alertMessage = "로그인 성공."
                    self?.accessToken = token
                    self?.isLoggedIn = true
                    print("로그인성공: \(self?.accessToken)")

                    completion(true)
                } else {
                    self?.alertMessage = error?.localizedDescription ?? "로그인 중 오류가 발생했습니다."
                    self?.showAlert = true
                    completion(false)
                }
            }
        }
    }
}

```

```
        }
    }
}

func logout() {
    self.isLoggedIn = false
    self.accessToken = nil
}
}
```

4:32

..... ⚡ 🔋

4:32

..... ⚡ 🔋

## 로그인

커넥트럭에 로그인하세요.

### 이메일

이메일을 입력하세요.

## 환영합니다.

여러분이 원하시는 축제와 푸드트럭을 찾아보세요!

### 비밀번호

비밀번호를 입력하세요.

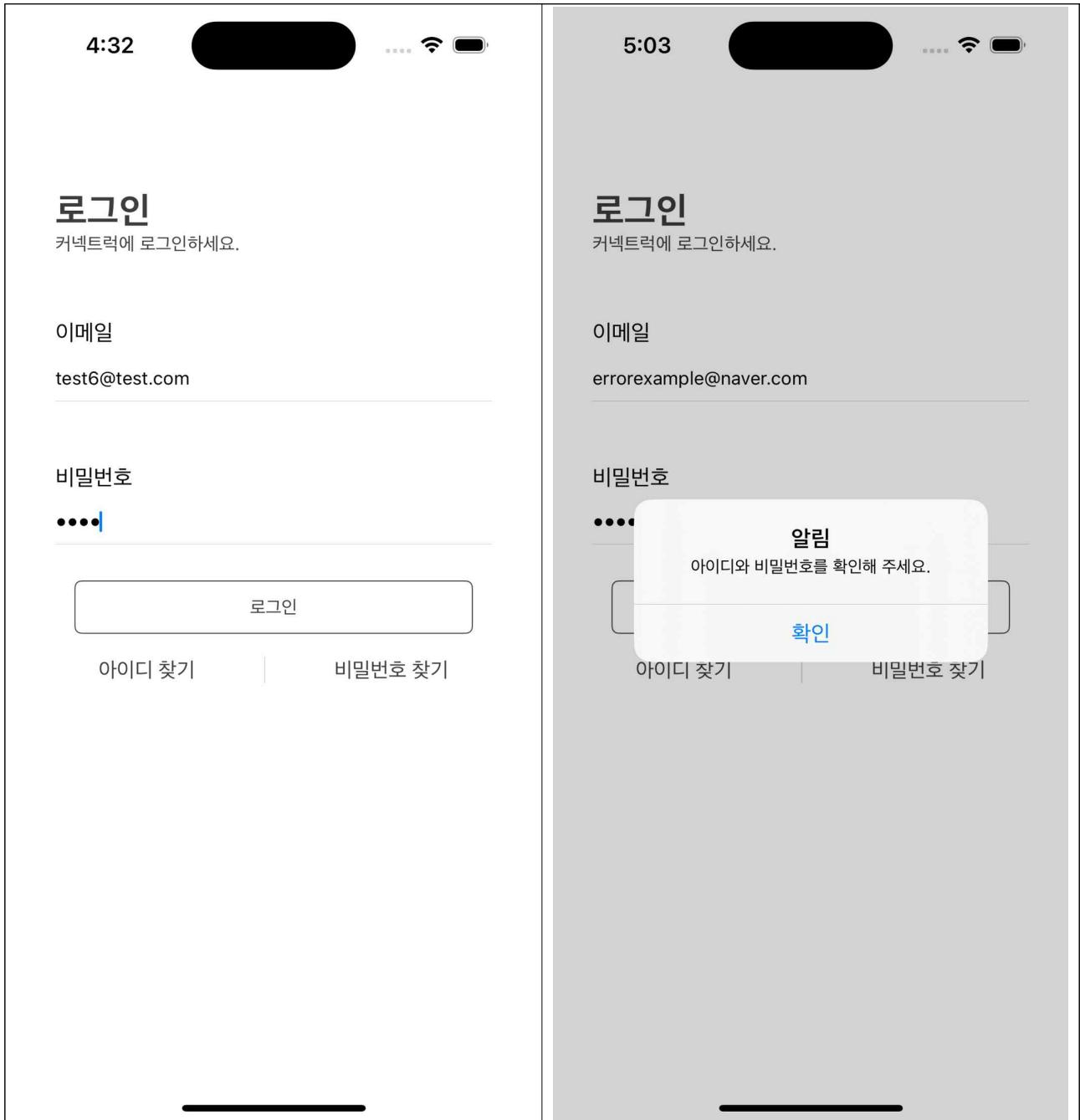
로그인

아이디 찾기

비밀번호 찾기

로그인

회원가입



Android
<b>class</b> LoginFragment.kt
<pre>@AndroidEntryPoint class LoginFragment : Fragment(){      @Inject     lateinit var authApi:AuthApi      private lateinit var binding:FragmentLoginBinding     private lateinit var authViewModel:AuthViewModel      override fun onCreateView(         inflater:LayoutInflater,         container:ViewGroup?,         savedInstanceState:Bundle?</pre>

```

):View {
    binding = FragmentLoginBinding.inflate(inflater, container, false)
    authViewModel = ViewModelProvider(requireActivity())[AuthViewModel::class.java]
    binding.viewModel = authViewModel

    binding.loginButton.setOnClickListener {
        lifecycleScope.launch {
            val loginRequest: LoginRequest = authViewModel.getLoginRequest()
            val response: Response<TokenResponse> = authApi.login(loginRequest)
            if (response.isSuccessful) {
                val token: String = response.body()!!.token!!
                Toast.makeText(context, "로그인 성공", Toast.LENGTH_SHORT).show()
                MySharedPreferences.setUserinfo(
                    requireContext(),
                    loginRequest.email,
                    loginRequest.password
                )
                MySharedPreferences.setToken(requireContext(), token)

                //로그인 성공 시 EventActivity를 시작
                val intent = Intent(requireContext(), HomeActivity::class.java)
                startActivity(intent)
                requireActivity().finish()
            } else {
                Toast.makeText(context, "로그인 실패", Toast.LENGTH_SHORT).show()
            }
        }
    }

    binding.signup.setOnClickListener {
        val action = LoginFragmentDirections.actionLoginFragment2ToSignUpInputEmailFragment2()
        Navigation.findNavController(binding.root).navigate(action)
    }

    binding.emailFind.setOnClickListener {
        val action = LoginFragmentDirections.actionLoginFragment2ToEmailFindFragment()
        Navigation.findNavController(binding.root).navigate(action)
    }

    return binding.root
}

```

### class LoginFragment.kt

Dagger Hilt 사용하여 의존성 주입 - 코드 모듈화하여 재사용성을 높이고, 클래스 간의 결합도를 낮춰 유지보수성 향상

lifecycleScope.launch 사용하여 코루틴 시작하여 비동기 작업 수행 - UI 업데이트와 같은 작업을 수행하면서도 앱의 응답성 유지

### class StartActivity.kt

```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    binding = ActivityStartBinding.inflate(layoutInflater)
    setContentView(binding.root)

    val email = MySharedPreferences.getUserEmail(this)
    val password = MySharedPreferences.getUserPassword(this)
}

```

```

//기기에 저장된 아이디와 비밀번호로 자동 로그인 시도 후 화면 전환
lifecycleScope.launch {
    if (email.isNotBlank()&&password.isNotBlank()){
        val response =authApi.login(LoginRequest(email,password))
        if (response.isSuccessful){
            Toast.makeText(this@StartActivity,"자동 로그인 성공")
            MySharedPreferences.setToken(this@StartActivity,response.body()!!.token!!)
        }else {
            Toast.makeText(this@StartActivity,"로그인 정보가 만료되었습니다.")
            MySharedPreferences.clearUserInfo(this@StartActivity)
        }
    }
    changeActivity()
}
}

/**
 *로그인 정보에 따라 화면 전환
 */
private fun changeActivity(){
    intent =if (MySharedPreferences.getUserEmail(this).isBlank()){
        Intent(this,AuthActivity::class.java)
    }else {
        Intent(this,HomeActivity::class.java)
    }
    startActivity(intent)
    this.finish()
}

```

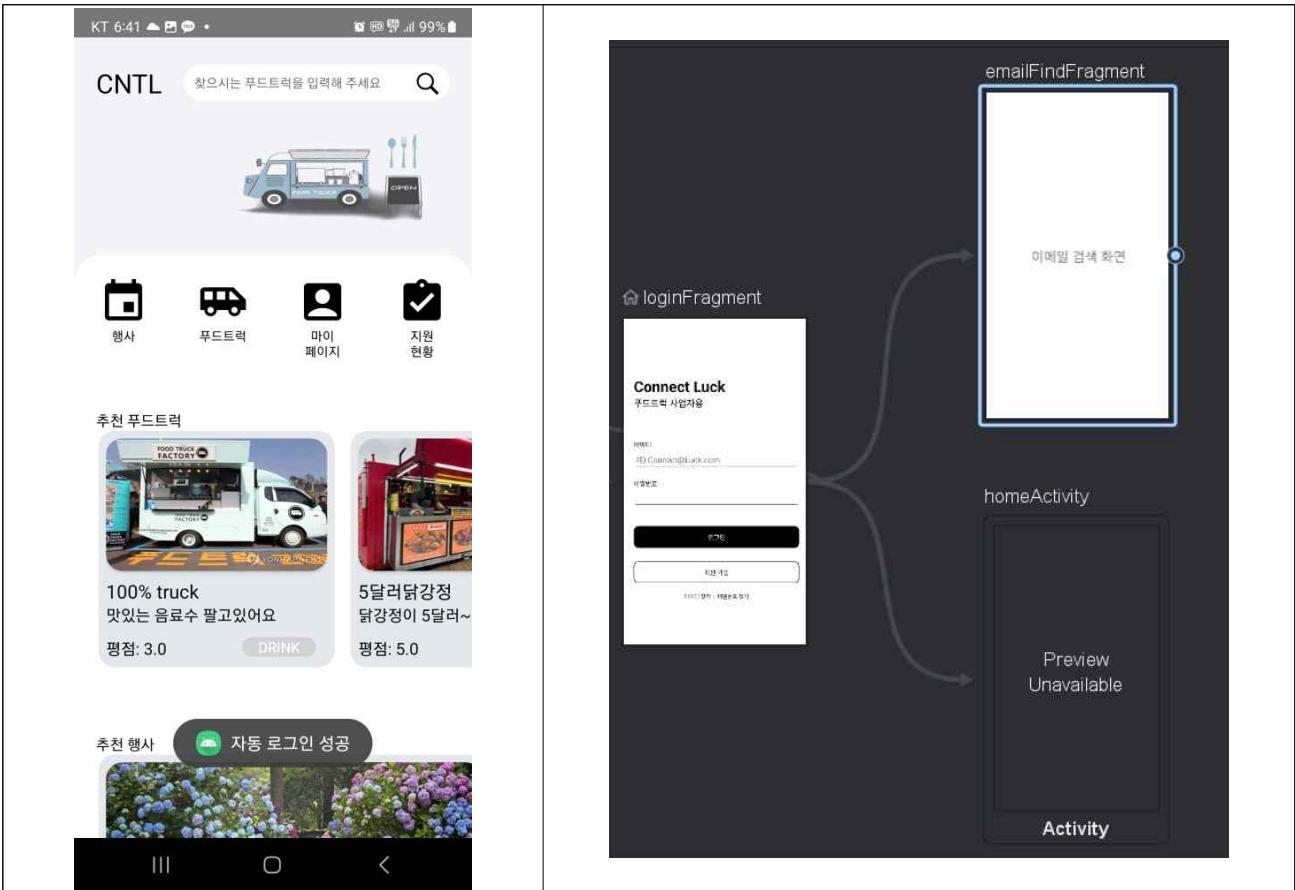
interface AuthApi.kt

```

/**
 * 서버의 로그인 API 엔드포인트
 *
 * @param loginRequest 로그인 요청 데이터를 전송
 * @return 서버로부터 토큰 응답을 받음
 */
@POST("/api/auth/login")
suspend fun login(
    @Body loginRequest: LoginRequest
): Response<TokenResponse>

```

사용자가 입력한 이메일과 비밀번호를 가져와서 서버에 로그인 요청은 authApi인터페이스의 login메서드를 통해 요청, 성공하면 서버에서 반환된 토큰을 받아와서 저장하고, 로그인이 성공했다는 메시지를 표시한 후, 홈 화면으로 이동



### 3) 회원정보 수정

#### (1) 분석 및 설계

기존 사용자들에게 개인정보를 수정하는 기능을 제공합니다. 수정될 수 있는 정보로는 이름, 전화번호, 비밀번호 등 개인정보들을 수정할 수 있습니다, 다만 이메일 변경의 경우 메인 고유키로 사용되고 있기 때문에, 조심해서 제공해야 하는 기능입니다. 따라서 이메일까지 변경을 염두에 두고 있다면 개인이 1:N으로 참조중인 여러 데이터들도 고려를 해서 제공해야 합니다.

또한, 유저 권한을 변경할 수 있어야 합니다. 첫 회원가입 시 User라는 역할만 부여받게 되는데, 서비스를 사용하기 위해서는 푸드트럭\_매니저, 행사\_매니저의 역할을 부여 받아야 합니다.

#### (2) 구현

Server
SignUpRequest.java
<pre>@Schema(description = "회원가입 요청 정보 객체") public record SignUpRequest(     @Schema(description = "이메일", example = "test@test.com")     String email,     @Schema(description = "비밀번호", example = "test")     String password,     @Schema(description = "이름", example = "홍길동")     String name,     @Schema(description = "휴대폰 번호", example = "010-1234-5678")     String phoneNumber ){}</pre>
UserService.java
<pre>/**  * 유저 정보 수정  *  * @param userEmail      수정할 유저 이메일  * @param signUpRequest 수정할 정보  * @return 수정된 유저 정보  */ public User updateUser(String userEmail, SignUpRequest signUpRequest){     User findUser = userRepository.findByEmail(userEmail).orElseThrow(() -&gt; new CustomException(CustomErrorCode.USER_ID_NOT_MATCH));     if (signUpRequest.getEmail() != null) findUser.setEmail(signUpRequest.getEmail());     if (signUpRequest.getPassword() != null) findUser.setPassword(passwordEncoder.encode(signUpRequest. password()));     if (signUpRequest.getName() != null) findUser.setName(signUpRequest.getName());     if (signUpRequest.getPhoneNumber() != null) findUser.setPhone(signUpRequest.getPhoneNumber());     return userRepository.save(findUser); }</pre>
UserController.java
<pre>/**  * 유저 정보 수정  *  * @param principal    로그인한 유저 정보  * @param password     비밀번호  * @param name          이름  * @param phoneNumber  전화번호  * @return 유저 정보 수정 후 프로필 페이지로 이동  */ @PostMapping("/update") public String updateUser(</pre>

```

    Principal principal,
    @RequestParam("password")String password,
    @RequestParam("name")String name,
    @RequestParam("phoneNumber")String phoneNumber
){
    SignUpRequest signUpRequest =new
    SignUpRequest(principal.getName(),password,name,phoneNumber);
    userService.updateUser(principal.getName(),signUpRequest);
    return "redirect:/user";
}

```

UserRestController.java

```

@PatchMapping
@Operation(summary ="사용자 정보 수정",description ="<h1>사용자정보를
수정합니다.</h1>변경하지 않을 정보는 null로 입력하세요.<br><br><h3>주의: 사용자 정보 변경 시
JWT 토큰을 다시 발급 받으세요.</h3>이전 토큰을 사용하면 요청이 정상적으로 처리되지
않습니다.")
public ResponseEntity<User>updateUser(
    Principal principal,
    @RequestBody SignUpRequest user
){
    return ResponseEntity.ok(userService.updateUser(principal.getName(),user));
}

```

### Web

#### profile\_edit.html

```

<form method="post" th:action="@{/user/update}">
    <div class="form-group">
        <label for="email">이메일:</label>
        <input class="form-control" disabled
id="email" name="email" th:value="${userInfo.user.email}" type="email">
    </div>
    <div class="form-group">
        <label for="password">비밀번호:</label>
        <input class="form-control" id="password" name="password" required type="password">
    </div>
    <div class="form-group">
        <label for="name">이름:</label>
        <input class="form-control" id="name" name="name" required
th:value="${userInfo.user.name}" type="text">
    </div>
    <div class="form-group">
        <label for="phoneNumber">휴대폰 번호:</label>
        <input class="form-control" id="phoneNumber" name="phoneNumber" required
th:value="${userInfo.user.phone}" type="text">
    </div>
    <button class="btn btn-primary" type="submit">저장</button>
</form>

```

## 회원 정보 수정

이메일:

비밀번호:

이름:

휴대폰 번호:

#### 4) 푸드트럭 등록 기능

##### (1) 분석 및 설계

후드트럭 매니저들을 위한 기능입니다. 본인이 운영하는 푸드트럭을 등록할 수 있는 기능을 제공해야 합니다. 해당 서비스에서는 푸드트럭의 기본적인 정보 - 상호명, 대표 이미지, 음식 종류등의 정보들을 저장 할 수 있어야 합니다.

##### (2) 구현

Server
FoodTruckRequestV2.kt
<pre>@Schema(description ="푸드트럭 등록 요청 DTO") data class FoodTruckRequestV2(     @field:Schema(description ="상호명",example ="맛있는 푸드트럭")     var name:String,     @field:Schema(description ="설명",example ="맛있는 음식을 판매하는 푸드트럭입니다.")     var description:String,     @field:Schema(description ="이미지",nullable =true)     var image:MultipartFile?,     @field:Schema(description ="음식 종류",example ="KOREAN")     var foodType:FoodType, ){     override fun toString():String {         return "FoodTruckRequestV2(name='\$name', description='\$description', image=\$image, foodType=\$foodType)"     } }</pre>
FoodTruckService.java
<pre>/**  * 푸드트럭을 추가합니다.  *  * @param userEmail 푸드트럭 매니저의 ID  * @param foodTruckRequest 추가할 푸드트럭 정보  * @return 저장된 푸드트럭 정보  */ public FoodTruckDetailResponse saveFoodTruck(String userEmail,FoodTruckRequestV2 foodTruckRequest){     FoodTruck foodTruck =foodTruckMapper.toFoodTruck(foodTruckRequest);     foodTruck.setManager(userRepository.findByEmail(userEmail).orElseThrow());      // 이미지가 있는 경우 이미지 업로드     if (foodTruckRequest.getImage() !=null){  foodTruck.setImageUrl(imageUploader.uploadImage(foodTruckRequest.getImage()).getData().getUr l());     }      return foodTruckMapper.toFoodTruckDetailResponse(foodTruckRepository.save(foodTruck)); }</pre>
FoodTruckController.java
<pre>@PreAuthorize("hasRole('FOOD_TRUCK_MANAGER')") @PostMapping("/register") public String registerFoodTruckPost(     Principal principal,     FoodTruckRequestV2 foodTruckRequest ){     foodTruckService.saveFoodTruck(principal.getName(),foodTruckRequest);     return "redirect:/food-truck/my"; }</pre>
FoodTruckRestController.kt

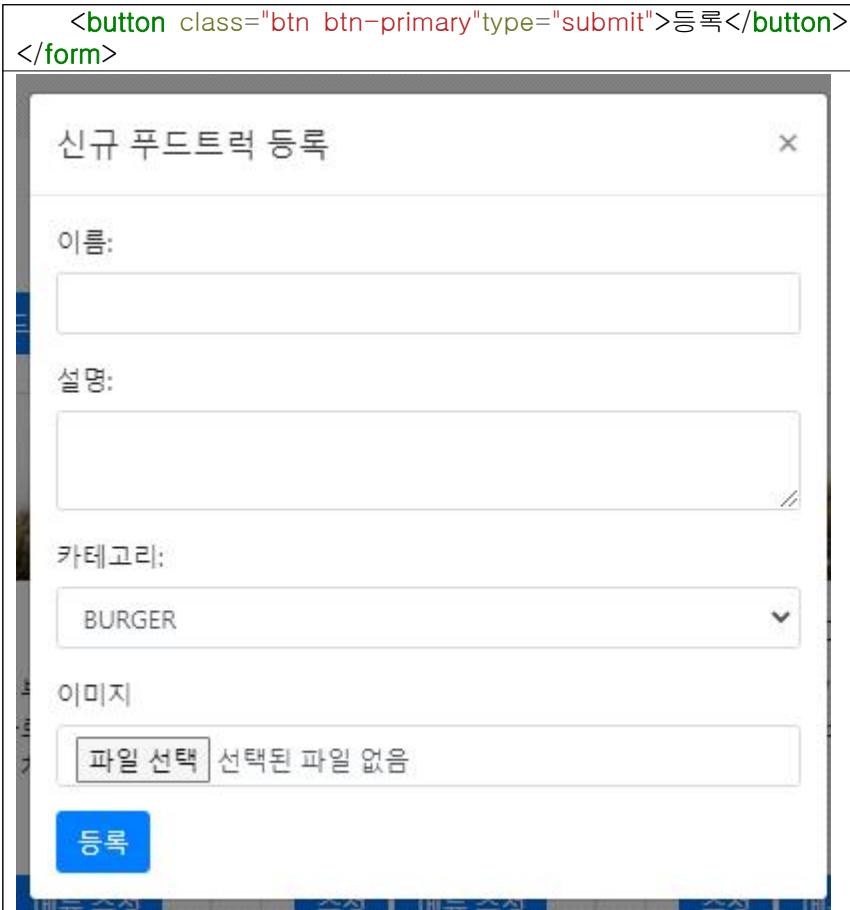
```

@Operation(
    summary ="푸드트럭 등록",
    description ="<h1>신규 푸드트럭을 등록합니다.</h1><b>푸드트럭 관리자만 가능합니다.</b><h3>주의 사항</h3><ul><li>name, description, foodType은 필수 항목입니다.</li><li>이미지를 입력하지 않을 경우 기본 이미지가 등록됩니다.</li><li>이미지를 비워둘 경우 Send Empty value를 체크 해주세요<br>빈 값 전송시 에러가 발생합니다.</li></ul><br>모두 Body의 Form Data로 전달해야 합니다."
)
@ApiResponses(
    value =[

        ApiResponse(
            responseCode ="201",
            description ="푸드트럭 등록 성공",
            content =[Content(mediaType =MediaType.APPLICATION_JSON_VALUE,schema
=Schema(implementation =FoodTruckDetailResponse::class))]
        ),
        ApiResponse(
            responseCode ="400",
            description ="푸드트럭 등록 실패",
            content =[Content(mediaType =MediaType.APPLICATION_JSON_VALUE,schema
=Schema(implementation =CustomErrorResponse::class))]
        ),
    ]
)
@PostMapping(consumes =[MediaType.MULTIPART_FORM_DATA_VALUE])
@PreAuthorize("hasRole('ROLE_FOOD_TRUCK_MANAGER')")
fun createFoodTruck(
    @ModelAttribute foodTruckRequestV2:FoodTruckRequestV2,principal:Principal
):ResponseEntity<FoodTruckDetailResponse>{
    val result:FoodTruckDetailResponse
    =foodTruckService.saveFoodTruck(principal.name,foodTruckRequestV2)
    return ResponseEntity.created(URI.create("/api/food-truck/${result.id}")).body(result)
}

```

Web
<pre> food_truck_edit.html &lt;form enctype="multipart/form-data" method="post" th:action="/food-truck/register"       th:object="\${foodTrucksRegisterForm}"&gt;     &lt;div class="form-group"&gt;       &lt;label for="name"&gt;이름:&lt;/label&gt;       &lt;input class="form-control" id="name" name="name" required type="text"&gt;     &lt;/div&gt;     &lt;div class="form-group"&gt;       &lt;label for="description"&gt;설명:&lt;/label&gt;       &lt;textarea class="form-control" id="description" name="description" required&gt;&lt;/textarea&gt;     &lt;/div&gt;     &lt;div class="form-group"&gt;       &lt;label for="foodType"&gt;카테고리:&lt;/label&gt;       &lt;select class="form-control" id="foodType" name="foodType" required&gt;         &lt;option th:each="foodType : \${T(ac.kr.deu.connect.luck.food_truck.entity.FoodType).values()}"                th:text="\${foodType}" th:value="\${foodType}"&gt;&lt;/option&gt;       &lt;/select&gt;     &lt;/div&gt;     &lt;div class="form-group"&gt;       &lt;label for="image"&gt;이미지&lt;/label&gt;       &lt;input class="form-control" id="image" name="image" type="file"&gt;     &lt;/div&gt; </pre>



```

Android
class FoodTruckCreateFragment.kt

/*
 *선택된 URI에서파일을생성
 *
 *@param uri선택된이미지의 URI
 *@return생성된파일객체
 */
private fun getFileFromUri(uri:Uri):File {
    val filePathColumn =arrayOf(MediaStore.Images.Media.DATA)
    val cursor =requireContext().contentResolver.query(uri,filePathColumn,null,null,null)
    cursor?.moveToFirst()
    val columnIndex =cursor?.getColumnIndex(filePathColumn[0])
    val filePath =columnIndex?.let {cursor.getString(it)}?:""
    cursor?.close()
    Log.d(TAG,"Real Path: $filePath")
    return File(filePath)
}

class FoodTruckCreateFragment.kt

/*
 *미디어 권한을 확인하고 갤러리를 엽니다.
 *필요할 경우 권한을 요청합니다.
 */
@RequiresApi(Build.VERSION_CODES.TIRAMISU)
private fun checkPermissionsAndOpenGallery(){
    val permissionsToRequest =mutableListOf<String>()
```

```

(ContextCompat.checkSelfPermission(requireContext(), Manifest.permission.READ_MEDIA_IMAGES)
    != PackageManager.PERMISSION_GRANTED
){
    permissionsToRequest.add(Manifest.permission.READ_MEDIA_IMAGES)
}
|
(ContextCompat.checkSelfPermission(requireContext(), Manifest.permission.READ_MEDIA_AUDIO)
    != PackageManager.PERMISSION_GRANTED
){
    permissionsToRequest.add(Manifest.permission.READ_MEDIA_AUDIO)
}
|
(ContextCompat.checkSelfPermission(requireContext(), Manifest.permission.READ_MEDIA_VIDEO)
    != PackageManager.PERMISSION_GRANTED
){
    permissionsToRequest.add(Manifest.permission.READ_MEDIA_VIDEO)
}

if (permissionsToRequest.isNotEmpty()){
    requestPermissionsLauncher.launch(permissionsToRequest.toTypedArray())
}else {
    openGallery()
}
}

```

#### class FoodTruckCreateFragment.kt

```

/**
 * 갤러리를 엽니다.
 */
private fun openGallery(){
    val intent = Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI)
    pickImageLauncher.launch(intent)
}

companion object {
    private const val TAG = "FoodTruckCreateFragment"
}

```

#### class FoodTruckCreateFragment.kt

```

/**
 * 이미지를 선택하고 결과를 처리하는 런처
 * 사용자가 이미지를 성공적으로 선택한 경우, 선택한 이미지의 URI를 가져와서 이미지 뷰에 표시합니다.
 */
private val pickImageLauncher = registerForActivityResult(
    ActivityResultContracts.StartActivityForResult()
){result->
    if(result.resultCode==Activity.RESULT_OK){
        result.data?.data?.let{uri->
            selectedImageUri=uri
            val inputStream:InputStream?=requireContext().contentResolver.openInputStream(uri)
            val bitmap=BitmapFactory.decodeStream(inputStream)
            binding.profile.setImageBitmap(bitmap)
        }
    }
}

```

### class FoodTruckCreateFragment.kt

```
// 여러 권한을 요청하고 결과를 처리하는 런처
// 사용자가 미디어 관련 권한을 모두 허용한 경우 갤러리를 열고, 그렇지 않으면 스낵바 메시지를 표시합니다.

@RequiresApi(Build.VERSION_CODES.TIRAMISU)
private val requestPermissionsLauncher = registerForActivityResult(
    ActivityResultContracts.RequestMultiplePermissions()
){permissions ->
    val isReadMediaImagesGranted = permissions[Manifest.permission.READ_MEDIA_IMAGES]?:false
    val isReadMediaAudioGranted = permissions[Manifest.permission.READ_MEDIA_AUDIO]?:false
    val isReadMediaVideoGranted = permissions[Manifest.permission.READ_MEDIA_VIDEO]?:false

    if (isReadMediaImagesGranted && isReadMediaAudioGranted && isReadMediaVideoGranted){
        openGallery()
    }else {
        Snackbar.make(requireView(),"미디어 권한이 필요합니다.",Snackbar.LENGTH_SHORT).show()
    }
}
```

### class FoodTruckCreateFragment.kt

```
'등록' 버튼을 클릭하면 입력된 푸드트럭 정보를 서버에 전송하여 등록을 시도합니다.
binding.button.setOnClickListener{
    valname=binding.editTextText.text.toString()
    valdescription=binding.editTextText2.text.toString()
    valfoodType=binding.spinner.selectedItem.toString()

    selectedImageUri?.let{uri->
        valimageFile=getFileFromUri(uri)
        valtoken=MySharedPreferences.getToken(requireContext())

        lifecycleScope.launch{
            valresponse=
                viewModel.registerFoodTruck(token,name,description,imageFile,foodType)
            if(response.isSuccessful){
                findNavController().navigate(R.id.action_foodTruckCreateFragment_to_myPageFragment)

                Log.d(TAG,"Foodtruckregistrationsuccessful.")
            }else{
                Log.d(TAG,"${response.errorBody()?.string()}")
                Log.d(TAG,"${response.headers()}")
                Log.d(TAG,"${response.body()}")
                Log.d(TAG,"Foodtruckregistrationfailed.")
            }
        }
    }?:run{
        Snackbar.make(requireView(),"이미지를 선택하세요.",Snackbar.LENGTH_SHORT).show()
    }
}
```

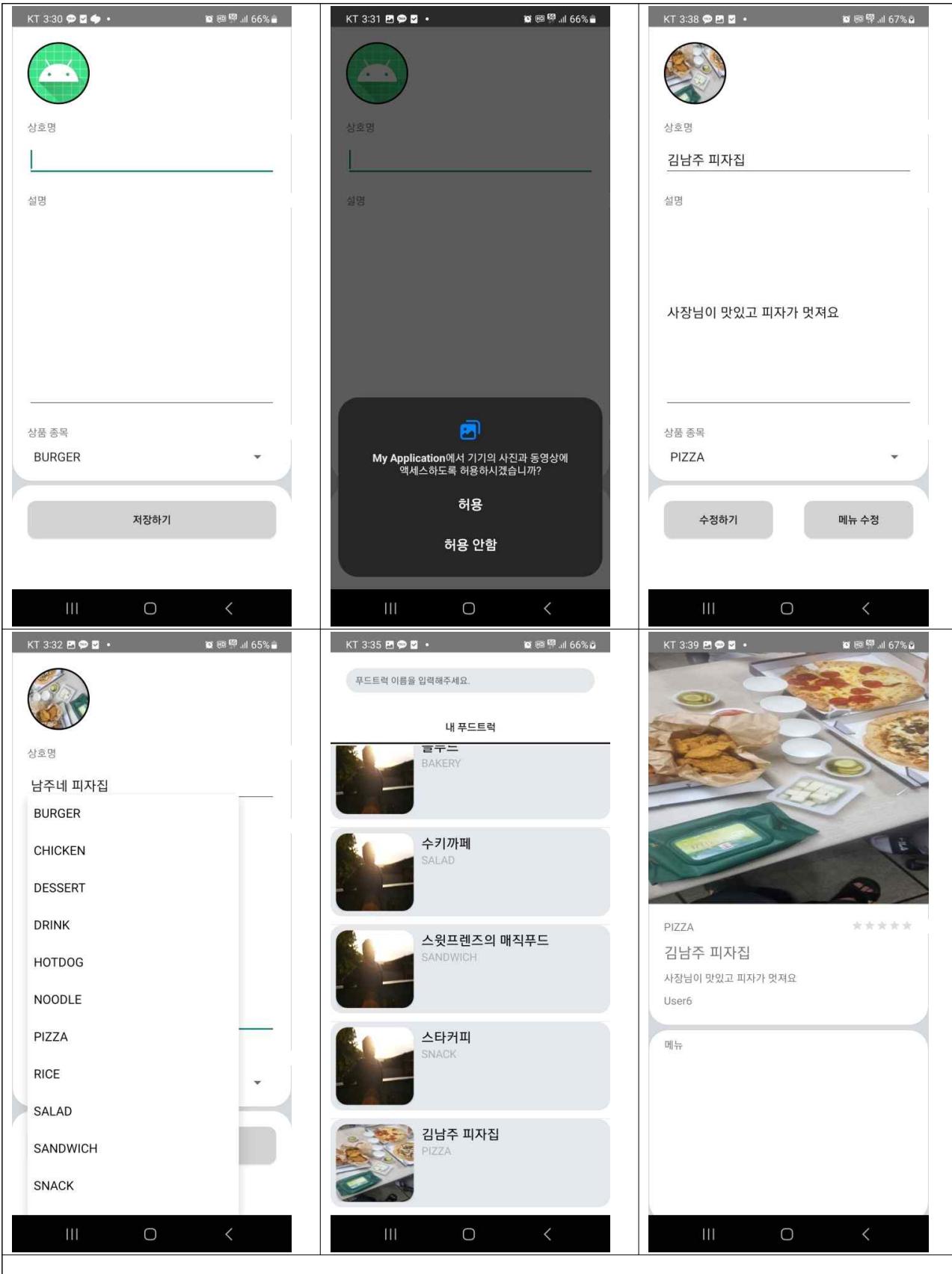
### class MyPageViewModel.kt

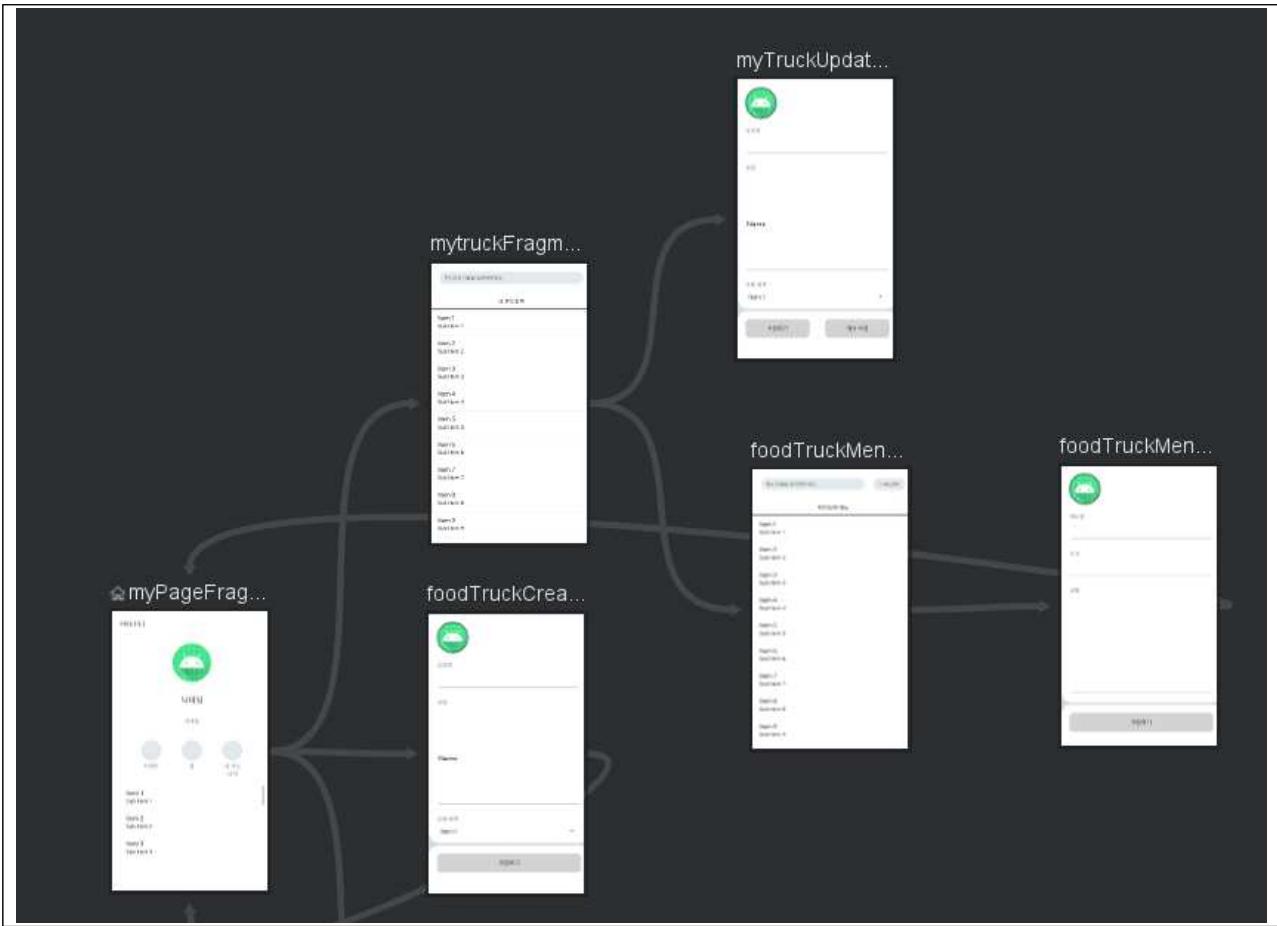
```
suspend fun registerFoodTruck(  
    token:String,  
    name:String,  
    description:String,  
    imageFile:File,  
    foodType:String  
):Response<ResponseBody>{  
    val nameRequestBody =name.toRequestBody("text/plain".toMediaTypeOrNull())  
    val descriptionRequestBody =description.toRequestBody("text/plain".toMediaTypeOrNull())  
    val foodTypeRequestBody =foodType.toRequestBody("text/plain".toMediaTypeOrNull())  
  
    val imageRequestBody =imageFile.asRequestBody("image/*".toMediaTypeOrNull())  
    val imagePart  
        =MultipartBody.Part.createFormData("image",imageFile.name,imageRequestBody)  
  
    val bearerToken ="Bearer $token"  
  
    r e t u r n  
foodTruckApi.registerFoodTruck(bearerToken,nameRequestBody,foodTypeRequestBody,description  
RequestBody,imagePart)  
}
```

### interface FoodTruckApi.kt

```
/*  
*푸드트럭을등록하는 API요청을보냅니다.  
*  
*@param token인증토큰 ("Bearer {token}"형식)  
*@param name푸드트럭이름 (RequestBody형식)  
*@param foodType음식종류 (RequestBody형식)  
*@param description푸드트럭설명 (RequestBody형식)  
*@param image이미지파일 (MultipartBody.Part형식)  
*@return서버응답 (ResponseBody형식)  
*/  
@Multipart  
    @POST("/api/food-truck")  
    suspend fun registerFoodTruck(  
        @Header("Authorization")token:String,  
        @Part("name")name:RequestBody,  
        @Part("foodType")foodType:RequestBody,  
        @Part("description")description:RequestBody,  
        @Part image:MultipartBody.Part  
    ):Response<ResponseBody>
```

사용자가 버튼 클릭하면 사용자가 입력한 푸드트럭 정보를 가져오고, 선택된 이미지 파일 가져와서 이미지 파일이 있는 경우에만 서버에 정보를 전송  
정보를 서버에 전송하기 위해 토큰과 함께 푸드트럭 이름, 설명, 음식 종류 및 이미지 파일 전송. 이 때 이미지 파일은 멀티파트 폼 데이터 전송





## 5) 푸드트럭 수정 기능

### (1) 분석 및 설계

푸드트럭의 정보를 수정하는 기능을 제공해야 합니다. 해당 기능을 사용하려면 2가지의 전제 조건이 필요합니다. 1번 - 푸드트럭 매니저 역할을 가지고 있는가, 2번 - 수정하려는 푸드트럭이 내가 생성한 푸드트럭인가입니다. 해당 기능을 통해서 앞에서 생성한 푸드트럭 정보를 수정할 수 있습니다.

### (2) 구현

Server
FoodTruckRequestV2.kt
<pre>@Schema(description ="푸드트럭 등록 요청 DTO") data class FoodTruckRequestV2(     @field:Schema(description ="상호명",example ="맛있는 푸드트럭")     var name:String,     @field:Schema(description ="설명",example ="맛있는 음식을 판매하는 푸드트럭입니다.")     var description:String,     @field:Schema(description ="이미지",nullable =true)     var image:MultipartFile?,     @field:Schema(description ="음식 종류",example ="KOREAN")     var foodType:FoodType, ){     override fun toString():String {         return "FoodTruckRequestV2(name='\$name', description='\$description', image=\$image, foodType=\$foodType)"     } }</pre>
FoodTruckService.java
<pre>/**  * 푸드트럭 정보를 수정합니다. 수정할 정보가 없는 경우 기존 정보를 유지합니다.  *  * @param foodTruckId     푸드트럭 ID  * @param foodTruckRequest 수정할 푸드트럭 정보  * @return 수정된 푸드트럭 정보  */ public FoodTruckDetailResponse saveFoodTruck(Long foodTruckId,FoodTruckRequestV2 foodTruckRequest){     // Search food truck     FoodTruck foodTruck =foodTruckRepository.findById(foodTruckId).orElseThrow(         ()-&gt;new CustomException(CustomErrorCode.FOOD_TRUCK_NOT_FOUND)     );      foodTruck.setName(foodTruckRequest.getName());     foodTruck.setDescription(foodTruckRequest.getDescription());     foodTruck.setFoodType(foodTruckRequest.getFoodType());      try {         if (foodTruckRequest.getImage() !=null){  foodTruck.setImageUrl(imageUploader.uploadImage(foodTruckRequest.getImage()).getData().getUri());         }     }catch (Exception e){         log.error("Failed to upload image",e);     }      // 수정된 푸드트럭 정보 저장 }</pre>

```

FoodTruck saved =foodTruckRepository.save(foodTruck);
    return foodTruckMapper.toFoodTruckDetailResponse(saved);
}

```

FoodTruckController.java

```

@PostMapping(value ="/{id}/edit",consumes ={MediaType.MULTIPART_FORM_DATA_VALUE})
@PreAuthorize("hasRole('ROLE_FOOD_TRUCK_MANAGER')") and
@checker.isFoodTruckManager(#id)
public String editFoodTruckPost(
    @PathVariable Long id,Principal principal,
    FoodTruckRequestV2 foodTruckRequest
){
    foodTruckService.saveFoodTruck(id,foodTruckRequest);
    return "redirect:/food-truck/my";
}

```

FoodTruckRestController.kt

```

@Operation(
    summary ="푸드트럭 정보를 업데이트 합니다.",description ="푸드트럭 정보를 업데이트 합니다.  
해당 푸드트럭 관리자만 가능합니다.<br>값을 변경하지 않더라도 값을 꼭 넣어야 합니다. (이미지는 예외)"
)
@ApiResponses(
    value =[

        ApiResponse(
            responseCode ="200",
            description ="푸드트럭 정보 업데이트 성공",
            content =[Content(mediaType =MediaType.APPLICATION_JSON_VALUE,schema
=Schema(implementation =FoodTruckDetailResponse::class))]
        ),
        ApiResponse(
            responseCode ="400",
            description ="푸드트럭 정보 업데이트 실패",
            content =[Content(mediaType =MediaType.APPLICATION_JSON_VALUE,schema
=Schema(implementation =CustomErrorResponse::class))]
        ),
    ]
)

```

```

@PatchMapping(value ="/{id}",consumes =[MediaType.MULTIPART_FORM_DATA_VALUE])
@PreAuthorize("hasRole('ROLE_FOOD_TRUCK_MANAGER')") and
@checker.isFoodTruckManager(#truckId)
fun updateFoodTruck(
    @Parameter(description ="푸드트럭 ID")@PathVariable("id")truckId:Long,
    @ModelAttribute foodTruckRequestV2:FoodTruckRequestV2,
    principal:Principal
):ResponseEntity<FoodTruckDetailResponse>{
    return ResponseEntity.ok(foodTruckService.saveFoodTruck(truckId,foodTruckRequestV2))
}

```

Checker.java

해당 코드는 스프링 시큐리티에서 사용할 내 푸드트럭인지 확인하는 메소드입니다.  
SecurityContextHolder를 통해서 어떤 사용자가 요청하는지를 파악할 수 있으며, 요청자와 푸드트럭 소유자의 이메일을 확인함으로써 검증하게 됩니다.

```

/**
 * 요청자가 해당 푸드트럭의 매니저인지 확인
 *
 * @param id 푸드트럭 ID
 * @return 매니저이면 true, 아니면 false
 */
public boolean isFoodTruckManager(Long id){
    FoodTruck foodTruck =foodTruckRepository.findById(id).orElseThrow(
        ()->new IllegalArgumentException("해당 푸드트럭이 존재하지 않습니다."))
}

```

```

);
Authentication authentication =SecurityContextHolder.getContext().getAuthentication();
return authentication.getName().equals(foodTruck.getManager().getEmail());
}

```

<b>Web</b>
my_food_truck.list.html
<form method="post" th:action="/food-truck/\${foodTruckOriginal.id}/edit" th:object="\${foodTruck}" enctype="multipart/form-data">     <div class="form-group">         <label for="name">이름:</label>         <input class="form-control" id="name" name="name" required th:value="\${foodTruck.name}" type="text">     </div>     <div class="form-group">         <label for="description">설명:</label>         <textarea class="form-control" id="description" name="description" required th:text="\${foodTruck.description}"></textarea>     </div>      <div class="form-group">         <label for="foodType">카테고리:</label>         <select class="form-control" id="foodType" name="foodType" required>             <option th:each="foodType : \${T(ac.kr.deu.connect.luck.food_truck.entity.FoodType).values()}" th:selected="\${foodType == foodTruck.foodType.toString()}" th:text="\${foodType}" th:value="\${foodType}"></option>         </select>     </div>     <div class="form-group">         <label for="image">이미지:</label>         <input accept="image/*" class="form-control" id="image" name="image" type="file">     </div>      <button class="btn btn-primary" type="submit">수정</button>     <button class="btn btn-danger" th:id="delete" th:onclick=" location.href='/food-truck/' + \${foodTruckOriginal.id} + '/delete' " type="button">삭제     </button> </form>

## 폼 수정



이름:

설명:

카테고리:

이미지:  
 선택된 파일 없음

### Android

**class** FoodTruckUpdateFragment.kt

'수정' 버튼을 클릭하면 입력된 푸드트럭 정보를 서버에 전송하여 업데이트를 시도합니다.

```
binding.button.setOnClickListener{
    valupdatedName=binding.editTextText.text.toString()
    valupdatedDescription=binding.editTextText2.text.toString()
    valupdatedFoodType=binding.spinner.selectedItem.toString()

    selectedImageUri?.let{uri->
        valimageFile=getFileFromUri(uri)
        valtoken=MySharedPreferences.getToken(requireContext())

        lifecycleScope.launch{
            valfoodTruckId=selectedFoodTruck.id.toInt()

            valresponse=viewModel.updateFoodTruck(token,foodTruckId,updatedName,updatedFoodType,updatedDescription,imageFile)
            if(response.isSuccessful){
                Log.d(TAG,"Foodtruckupdatedsuccessfully")
            }else{
                Log.e(TAG,"Failedtoupdatefoodtruck.Error:${response.errorBody()?.string()}")
            }
        }?:run{
            Snackbar.make(requireView(),"이 미 지 를 선 택 하 세 요.",Snackbar.LENGTH_SHORT).show()
        }
    }
}
```

```

    }

class MyPageViewModel.kt

/*
@updateFoodTruck:
--푸드트럭정보를업데이트하는서버요청을보냅니다.
*
*@param token인증토큰
*@param id푸드트럭 ID
*@param name업데이트할푸드트럭이름
*@param foodType업데이트할음식종류
*@param description업데이트할푸드트럭설명
*@param imageFile업데이트할이미지파일
*@return서버응답
*/
suspend fun updateFoodTruck(
    token:String,
    id:Int,
    name:String,
    foodType:String,
    description:String,
    imageFile:File
):Response<ResponseBody>{
    // 요청 바디 생성
    val nameRequestBody =name.toRequestBody("text/plain".toMediaTypeOrNull())
    val descriptionRequestBody =description.toRequestBody("text/plain".toMediaTypeOrNull())
    val foodTypeRequestBody =foodType.toRequestBody("text/plain".toMediaTypeOrNull())

    // 이미지 파일을 요청 바디로 변환
    val imageRequestBody =imageFile.asRequestBody("image/*".toMediaTypeOrNull())
    // 이미지 파일을 Multipart 형식으로 생성
    val imagePart =MultipartBody.Part.createFormData("image",imageFile.name,imageRequestBody)

    // 인증 토큰을 헤더에 포함하여 요청 전송
    val bearerToken ="Bearer $token"
    foodTruckApi.updateFoodTruck(bearerToken,id,nameRequestBody,foodTypeRequestBody,descriptionRequestBody,imagePart)
}

interface FoodTruckApi.kt

/*
 * updateFoodTruck:
 * - 특정 푸드트럭의 정보를 업데이트하는 서버 요청을 보냅니다.
 *
 * @param token 인증 토큰
 * @param id 업데이트할 푸드트럭의 ID
 * @param name 업데이트할 푸드트럭 이름 (RequestBody 형식)
 * @param foodType 업데이트할 음식 종류 (RequestBody 형식)
 * @param description 업데이트할 푸드트럭 설명 (RequestBody 형식)
 * @param image 업데이트할 이미지 파일 (MultipartBody.Part 형식)
 * @return 서버 응답
 */
@Multipart

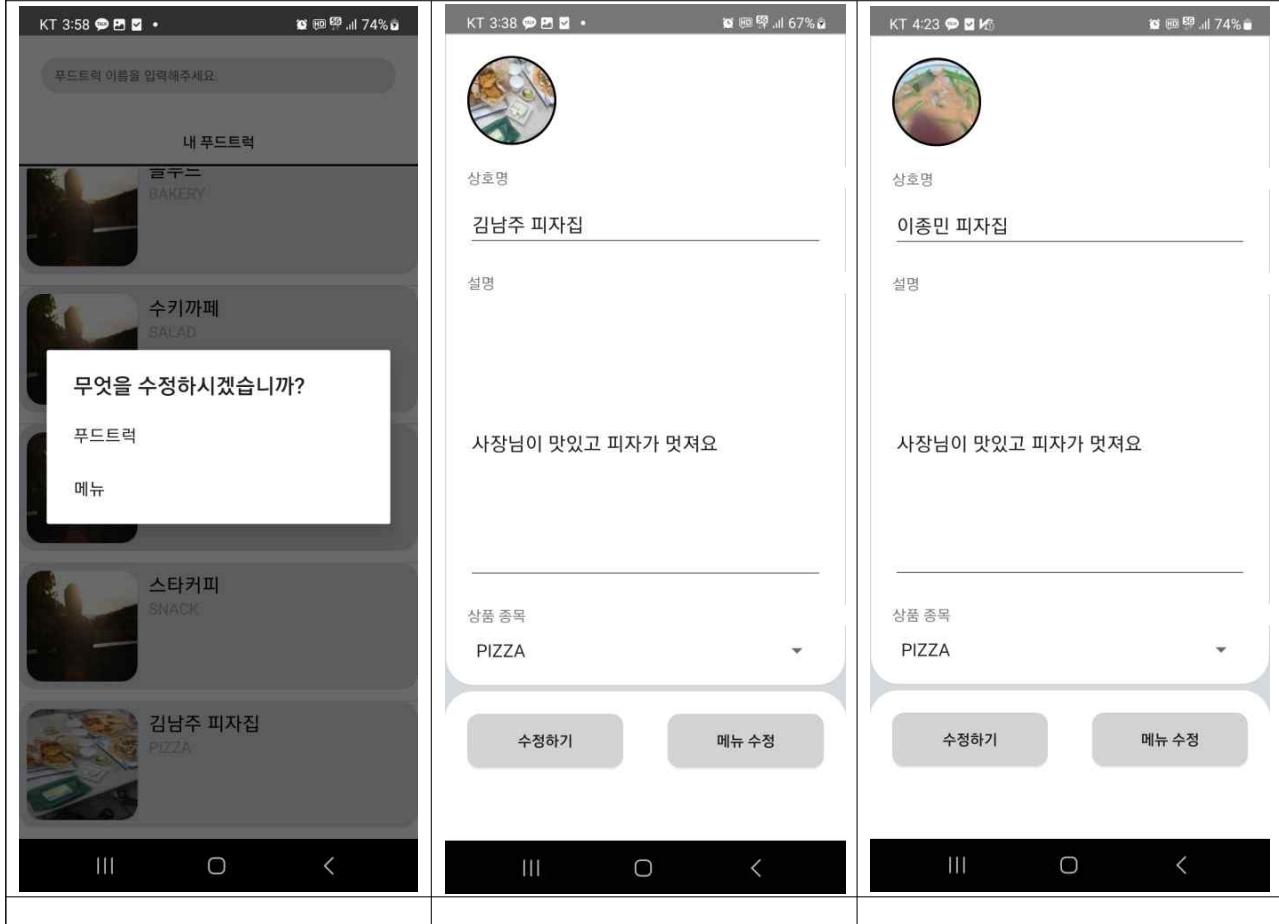
```

```

@PATCH("/api/food-truck/{id}")
suspend fun updateFoodTruck(
    @Header("Authorization") token: String,
    @Path("id") id: Int,
    @Part("name") name: RequestBody,
    @Part("foodType") foodType: RequestBody,
    @Part("description") description: RequestBody,
    @Part image: MultipartBody.Part
): Response<ResponseBody>

```

'수정' 버튼을 클릭하면 사용자가 입력한 푸드트럭 정보와 선택한 이미지를 서버에 전송하여 업데이트시도. 먼저 사용자가 입력한 정보와 이미지 파일을 가져와서 필요한 형식으로 변환한 후, 토큰과 업데이트할 푸드트럭의 ID를 서버에 요청





## 6) 푸드트럭 메뉴 등록 기능

### (1) 분석 및 설계

푸드트럭 메뉴를 등록하는 기능입니다. 해당 기능도 앞에서 말한 전제조건 2가지 푸드트럭 매니저, 해당 푸드트럭 매니저인지의 조건이 필요합니다.

### (2) 구현

Server
FoodTruckMenuRequest.kt
<pre>@Schema(description ="푸드트럭 메뉴 등록 요청 DTO") data class FoodTruckMenuRequest(     @field:Schema(description ="메뉴명",example ="햄버거")     val name:String,     @field:Schema(description ="가격",example ="5000")     val price:Int,     @field:Schema(description ="설명",example ="맛있는 햄버거")     val description:String,     @field:Schema(description ="이미지",example ="image.jpg")     val image:MultipartFile? )</pre>
FoodTruckMenuService.java
<pre>/*  * 푸드트럭 메뉴 등록  * &lt;p&gt;컨트롤러에서 검증을 마치고 요청되는 메소드 이므로 믿고 사용할 수 있다.&lt;/p&gt;  *  * @param foodTruckId          푸드트럭 ID  * @param foodTruckMenuRequest 푸드트럭 메뉴 요청  * @return FoodTruckMenu  */ public FoodTruckMenuResponse saveFoodTruckMenu(Long foodTruckId,FoodTruckMenuRequest foodTruckMenuRequest){     // 객체 생성     FoodTruck foodTruck =FoodTruck.builder().id(foodTruckId).build();     FoodTruckMenu foodTruckMenu =foodTruckMapper.toFoodTruckMenu(foodTruckMenuRequest);      // 푸드트럭 메뉴에 푸드트럭 설정     foodTruckMenu.setFoodTruck(foodTruck);      // 이미지가 있으면 업로드     if (foodTruckMenuRequest.getImage() !=null){         String imageUrl =imageUploader.uploadImage(foodTruckMenuRequest.getImage()).getData().getImage().getUrl();         foodTruckMenu.setImageUrl(imageUrl);     }      // 저장 후 응답     return     foodTruckMapper.toFoodTruckMenuResponse(foodTruckMenuRepository.save(foodTruckMenu)); }</pre>
FoodTruckMenuController.java
<pre>@PostMapping(value ="/{truckId}/add",consumes ="multipart/form-data") @PreAuthorize("hasRole('ROLE_FOOD_TRUCK_MANAGER')") and @checker.isFoodTruckManager(#truckId") public String addFoodTruckMenu(     @PathVariable(value ="truckId")Long truckId,     FoodTruckMenuRequest foodTruckMenuRequest,     Model model</pre>

```

){
    foodTruckMenuService.saveFoodTruckMenu(truckId,foodTruckMenuRequest);
    model.addAttribute("foodTruckId",truckId);
    return "redirect:/food-truck/menu/" + truckId;
}

```

FoodTruckMenuRestController.java

```

@Operation(summary ="푸드트럭 메뉴 등록",description ="신규 푸드트럭 메뉴를 등록합니다.  
푸드트럭 매니저의 역할이 필요합니다.")
@ApiResponses(value ={
    @ApiResponse(responseCode ="201",description ="생성 완료",content
    ={@Content(mediaType =MediaType.APPLICATION_JSON_VALUE,schema
    =@Schema(implementation =FoodTruckMenuResponse.class))}),
    @ApiResponse(responseCode ="403",description ="권한 없음",content
    ={@Content(mediaType =MediaType.APPLICATION_JSON_VALUE,schema
    =@Schema(implementation =CustomErrorResponse.class))}),
})
@PostMapping(consumes =MediaType.MULTIPART_FORM_DATA_VALUE)
@PreAuthorize("hasRole('ROLE_FOOD_TRUCK_MANAGER') and
@checker.isFoodTruckManager(#foodTruckId)")
public ResponseEntity<FoodTruckMenuResponse> saveFoodTruckMenu(
    @Parameter(description ="푸드트럭 UID") @PathVariable("foodTruckId") Long
foodTruckId,
    @ModelAttribute FoodTruckMenuRequest foodTruckMenuRequest
){
    FoodTruckMenuResponse foodTruckMenuResponse
    =foodTruckMenuService.saveFoodTruckMenu(foodTruckId,foodTruckMenuRequest);
    return
    ResponseEntity.created(URI.create("api/food-truck/" + foodTruckMenuResponse.id())).body(foodTruckMenuResponse);
}

```

### Android

#### class FoodTruckMenuCreateFragment.kt

'등록' 버튼을 클릭하면 입력된 푸드트럭 메뉴 정보를 서버에 전송하여 등록을 시도합니다.

```

binding.button.setOnClickListener {
    val name = binding.editTextText.text.toString()
    val description = binding.editTextText2.text.toString()
    val price = binding.editTextText3.text.toString().toDoubleOrNull()

    if (price ==null){
        Snackbar.make(requireView(),"가격을 올바르게 입력하세요.",Snackbar.LENGTH_SHORT).show()
        return
    }

    selectedImageUri?.let {uri ->
        val imageFile = getFileFromUri(uri)
        val token = MySharedPreferences.getToken(requireContext())
    }

    lifecycleScope.launch {
        val response = viewModel.addFoodTruckMenu(token,id,name,
            price.toInt(),description,imageFile)
        if (response.isSuccessful){
    }

```

```

findNavController().navigate(R.id.action_foodTruckMenuCreateFragment_to_myPageFragment)

        Log.d(TAG,"Menu registration successful.")
    }else {
        Log.d(TAG,"Menu registration failed.")
    }
}
}?:run {
    Snackbar.make(requireView(),"이미지를 선택하세요.",Snackbar.LENGTH_SHORT).show()
}
}

```

### class MyPageViewModel.kt

```

/*
*addFoodTruckMenu:
*-푸드트럭에새로운메뉴를추가하는서버요청을보냅니다.
*
*@param token인증토큰
*@param foodTruckId메뉴를추가할푸드트럭의 ID
*@param name추가할메뉴의이름
*@param price추가할메뉴의가격
*@param description추가할메뉴의설명
*@param imageFile추가할메뉴의이미지파일
*@return서버응답
*/
suspend fun addFoodTruckMenu(
    token:String,
    foodTruckId:Int,
    name:String,
    price:Int,
    description:String,
    imageFile:File
):Response<ResponseBody>{
    // 요청 바디 생성
    val nameRequestBody =name.toRequestBody("text/plain".toMediaTypeOrNull())
    val priceRequestBody =price.toString().toRequestBody("text/plain".toMediaTypeOrNull())
    val descriptionRequestBody =description.toRequestBody("text/plain".toMediaTypeOrNull())

    // 이미지 파일을 요청 바디로 변환
    val imageRequestBody =imageFile.asRequestBody("image/*".toMediaTypeOrNull())
    // 이미지 파일을 Multipart 형식으로 생성
    val imagePart =MultipartBody.Part.createFormData("image",imageFile.name,imageRequestBody)
    // 인증 토큰을 헤더에 포함하여 요청 전송
    val bearerToken ="Bearer $token"
    return foodTruckApi.addFoodTruckMenu(
        bearerToken,
        foodTruckId,
        nameRequestBody,
        priceRequestBody,
        descriptionRequestBody,
        imagePart
    )
}

```

### interface FoodTruckApi.kt

```

/*
*addFoodTruckMenu:
*-특정푸드트럭에새로운메뉴를추가하는서버요청을보냅니다.
*/

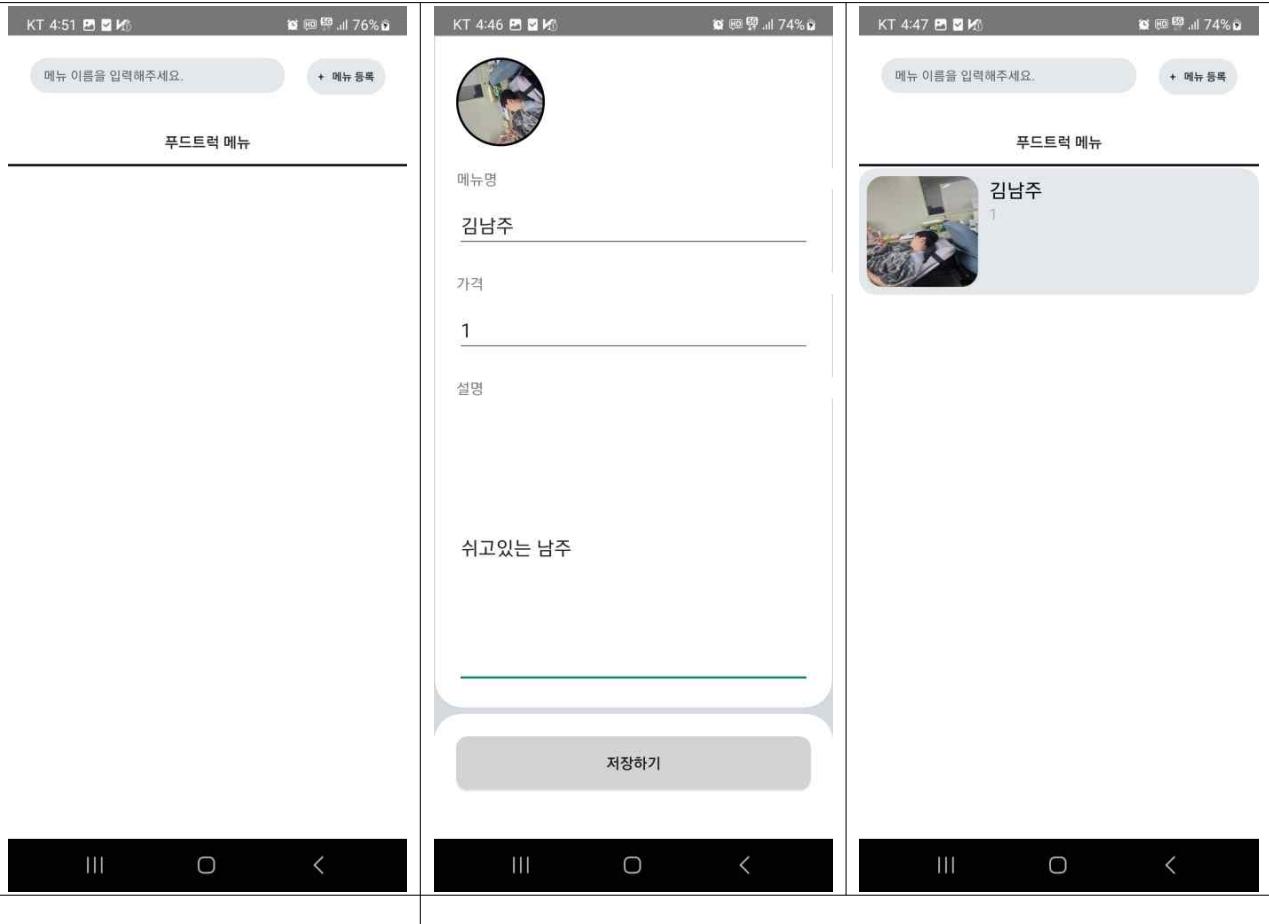
```

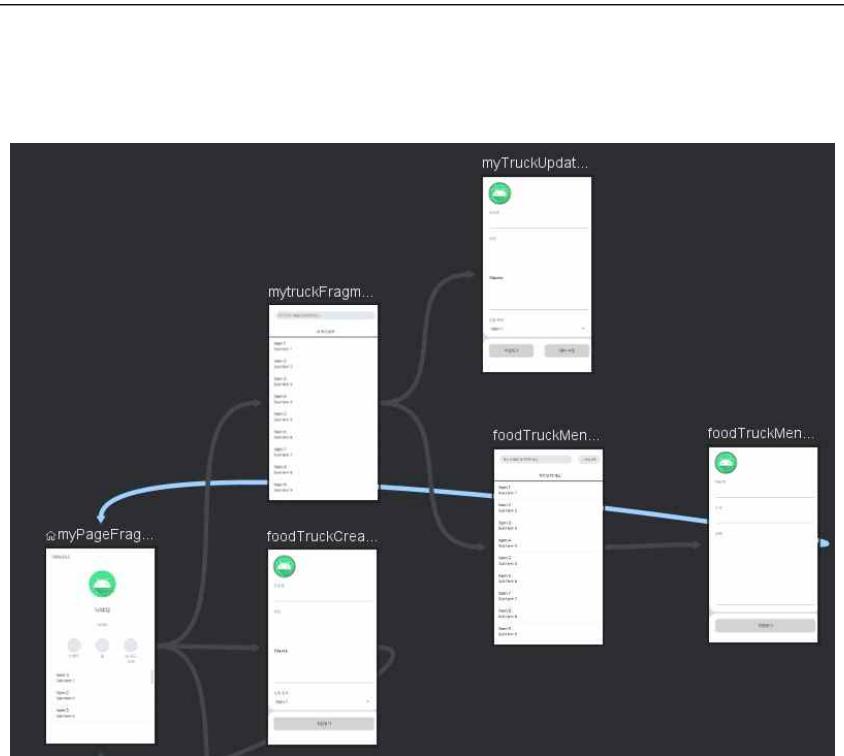
```

*@param token인증토큰
*@param foodTruckId메뉴를 추가할 푸드트럭의 ID
*@param name추가할 메뉴의 이름 (RequestBody형식)
*@param price추가할 메뉴의 가격 (RequestBody형식)
*@param description추가할 메뉴의 설명 (RequestBody형식)
*@param image추가할 메뉴의 이미지 파일 (MultipartBody.Part형식)
*@return 서버응답
*/
@Multipart
@POST("/api/food-truck/{foodTruckId}/menu")
suspend fun addFoodTruckMenu(
    @Header("Authorization")token:String,
    @Path("foodTruckId")foodTruckId:Int,
    @Part("name")name:RequestBody,
    @Part("price")price:RequestBody,
    @Part("description")description:RequestBody,
    @Part image:MultipartBody.Part
):Response<ResponseBody>

```

버튼을 클릭하면 사용자가 입력한 푸드트럭 메뉴 정보와 선택한 이미지를 서버에 전송하여 메뉴를 등록할 수 있습니다. 텍스트 필드에는 메뉴명, 가격, 설명을 입력하고, 이미지는 사진이나 그림을 선택해주세요.





## 7) 푸드트럭 메뉴 수정

### (1) 분석 및 설계

간략한 설명

### (2) 구현

Server
FoodTruckMenuService.java
<pre>/*  * 푸드트럭 메뉴 수정  *  * @param foodTruckId      푸드트럭 ID  * @param foodTruckMenuRequest 푸드트럭 메뉴 요청  * @param foodTruckMenuld    푸드트럭 메뉴 ID  * @return FoodTruckMenu 응답  */ public FoodTruckMenuResponse saveFoodTruckMenu(Long foodTruckId,FoodTruckMenuRequest foodTruckMenuRequest,Long foodTruckMenuld){     // 객체 생성     FoodTruckMenu foodTruckMenu =foodTruckMenuRepository.findById(foodTruckMenuld)         .orElseThrow(RuntimeException::new);      // 푸드트럭 메뉴 ID 설정     foodTruckMenu.setName(foodTruckMenuRequest.getName());     foodTruckMenu.setDescription(foodTruckMenuRequest.getDescription());     foodTruckMenu.setPrice(foodTruckMenuRequest.getPrice());      // 이미지가 있으면 업로드     if (foodTruckMenuRequest.getImage() != null){         String imageUrl         =imageUploader.uploadImage(foodTruckMenuRequest.getImage()).getData().getImage().getUrl();         foodTruckMenu.setImageUrl(imageUrl);     }      // 저장 후 응답     return     foodTruckMapper.toFoodTruckMenuResponse(foodTruckMenuRepository.save(foodTruckMenu)); }</pre>
FoodTruckMenuRestController.java
<pre>@PatchMapping(value ="/{foodTruckMenuld}",consumes =MediaType.MULTIPART_FORM_DATA_VALUE) @PreAuthorize("hasRole('ROLE_FOOD_TRUCK_MANAGER') and @checker.isFoodTruckManager(#foodTruckId) and @checker.isFoodTruckMenu(#foodTruckId, #foodTruckMenuld") @Operation(summary ="푸드트럭 메뉴 수정",description ="특정 푸드트럭 메뉴를 수정합니다. 푸드트럭 매니저의 역할이 필요합니다.") public ResponseEntity&lt;FoodTruckMenuResponse&gt;updateFoodTruckMenu(     @Parameter(description ="푸드트럭 UID")@PathVariable("foodTruckId")Long foodTruckId,     @Parameter(description ="메뉴 UID")@PathVariable("foodTruckMenuld")Long foodTruckMenuld,     @ModelAttribute FoodTruckMenuRequest foodTruckMenuRequest ){     return     ResponseEntity.ok(foodTruckMenuService.saveFoodTruckMenu(foodTruckId,foodTruckMenuReque st,foodTruckMenuld)); }</pre>

**PATCH** /api/food-truck/{foodTruckId}/menu/{foodTruckMenuId}

푸드트럭 메뉴 수정

특정 푸드트럭 메뉴를 수정합니다. 푸드트럭 매니저의 역할이 필요합니다.

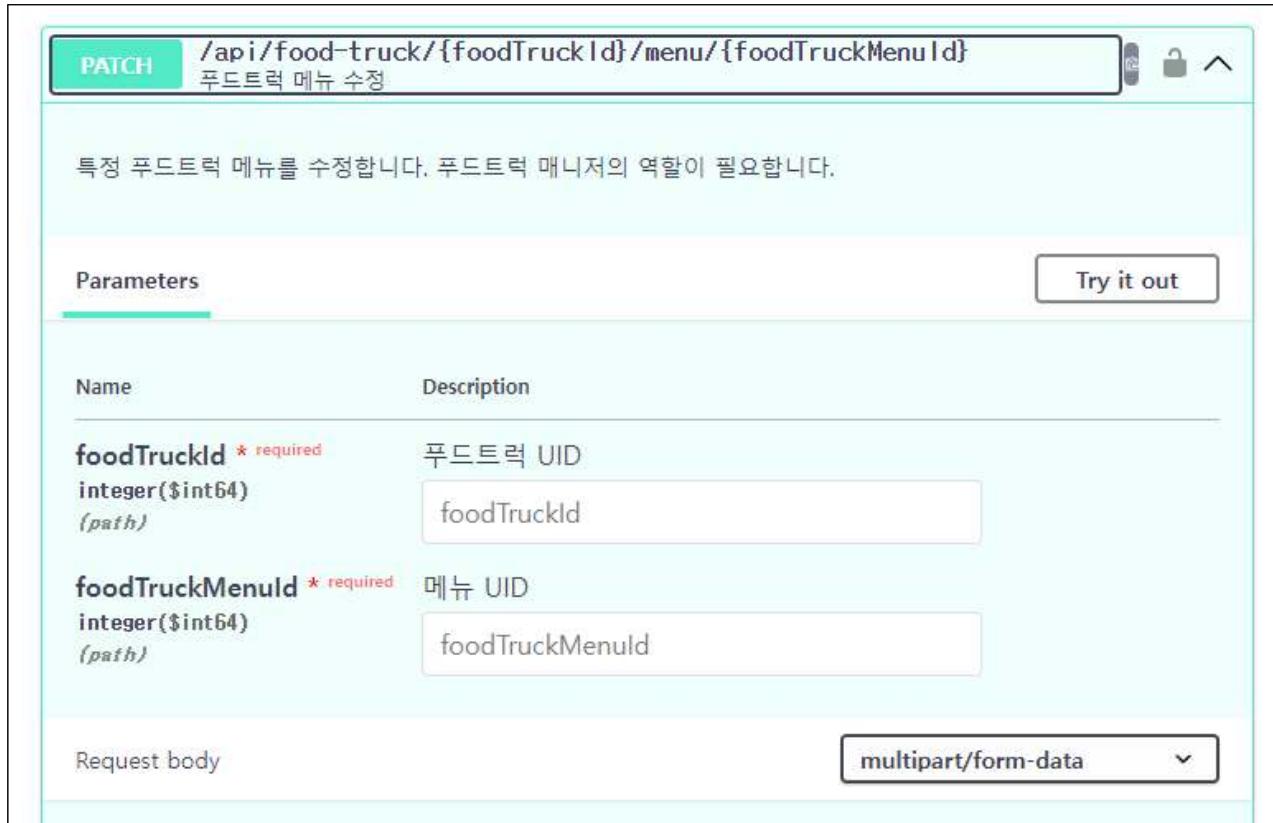
**Parameters**

**Try it out**

Name	Description
<b>foodTruckId</b> * required integer(\$int64) (path)	푸드트럭 UID foodTruckId
<b>foodTruckMenuId</b> * required integer(\$int64) (path)	메뉴 UID foodTruckMenuId

Request body

multipart/form-data



Web

## 8) 푸드트럭 후기 작성

### (1) 분석 및 설계

간략한 설명

### (2) 구현

Server
FoodTruckReviewRequest.kt
<pre>@Schema(description ="푸드트럭 리뷰 등록 요청 DTO") data class FoodTruckReviewRequest(     @field:Schema(description ="리뷰 내용",example ="맛있어요")     val content:String,     @field:Schema(description ="평점 1~5 사이값만 입력하십시오",example ="5")     @field:Min(1)@field:Max(5)     val rating:Int,     @field:Schema(description ="이미지",example ="image.jpg")     val image:MultipartFile? ){}     override fun toString():String {         return "FoodTruckReviewRequest(content='\$content', rating=\$rating image=\${image?.originalFilename})"     } }</pre>
FoodTruckReviewResponse.java
<pre>/**  * 푸드트럭 리뷰 등록  *  * @param foodTruckId          푸드트럭 ID  * @param foodTruckReviewRequest 푸드트럭 리뷰 요청  * @param userEmail             사용자 이메일  */ public FoodTruckReviewResponse saveFoodTruckReview(Long foodTruckId,FoodTruckReviewRequest foodTruckReviewRequest,String userEmail){     FoodTruck foodTruck =FoodTruck.builder().id(foodTruckId).build();     User author =userRepository.findByEmail(userEmail).orElseThrow();      FoodTruckReview review =foodTruckMapper.toFoodTruckReview(foodTruckReviewRequest);      review.setFoodTruck(foodTruck);     review.setAuthor(author);      if (foodTruckReviewRequest.getImage() !=null){         String imageUrl         =imageUploader.uploadImage(foodTruckReviewRequest.getImage()).getData().getUrl();         review.setImageUrl(imageUrl);     }      return     foodTruckMapper.toFoodTruckReviewResponse(foodTruckReviewRepository.save(review)); }</pre>
FoodTruckReviewRestController.java
<pre>@PreAuthorize("isAuthenticated()") @PostMapping(consumes =MediaType.MULTIPART_FORM_DATA_VALUE) public ResponseEntity&lt;FoodTruckReviewResponse&gt;createFoodTruckReview(     @Parameter(description ="푸드트럭 ID")@PathVariable("foodTruckId")Long foodTruckId,     @ModelAttribute @Valid FoodTruckReviewRequest foodTruckReviewRequest,     Principal principal</pre>

```
){
    FoodTruckReviewResponse foodTruckReviewResponse
    =foodTruckReviewService.saveFoodTruckReview(foodTruckId,foodTruckReviewRequest,principal
    .getName());
    return ResponseEntity.created(URI.create("api/food-truck/"+foodTruckId
    +"/review/")).body(foodTruckReviewResponse);
}
```

## 9) 푸드트럭 후기 답글 작성하기

### (1) 분석 및 설계

푸드트럭에 후기를 작성하는 기능을 제공합니다. 해당 기능을 사용하기 위해서는 인증된 유저만 게시글을 작성할 수 있습니다. 평점같은 경우에는 1~5점 숫자만 입력할 수 있으며, 게시글을 조회할 때 평균 점수를 볼 수 있습니다.

### (2) 구현

Server
FoodTruckReviewService.java
<pre>/*  * 푸드트럭 리뷰 답글 등록  *  * @param reviewId 리뷰 ID  * @param content 답글 내용  * @return 답글이 달린 리뷰  */ public FoodTruckReviewResponse addReplyToReview(Long reviewId, String content){     FoodTruckReview review = foodTruckReviewRepository.findById(reviewId).orElseThrow();     review.setReply(content);     return         foodTruckMapper.toFoodTruckReviewResponse(foodTruckReviewRepository.save(review)); }</pre>
FoodTruckReviewRestController.java
<pre>@Operation(summary ="푸드트럭 리뷰 답글 등록",description ="푸드트럭 리뷰에 답글을 등록합니다.&lt;br&gt;푸드트럭 매니저의 권한이 필요합니다.") @ApiResponses(value ={     @ApiResponse(responseCode ="201",description ="리뷰 답글 등록 성공",content =@Content(mediaType ="application/json",schema =@Schema(implementation =FoodTruckReviewResponse.class))),     @ApiResponse(responseCode ="403",description ="권한 없음",content =@Content(mediaType ="application/json",schema =@Schema(implementation =String.class))) }) @PostMapping("/{reviewId}/reply") @PreAuthorize("@checker.isFoodTruckManager(#foodTruckId)") public ResponseEntity&lt;FoodTruckReviewResponse&gt;createFoodTruckReviewReply(     @Parameter(description ="푸드트럭 ID")@PathVariable("foodTruckId")Long foodTruckId,     @Parameter(description ="리뷰 ID")@PathVariable("reviewId")Long reviewId,     @Parameter(description ="답글 내용")@RequestParam String content ){     FoodTruckReviewResponse foodTruckReviewResponse     = foodTruckReviewService.addReplyToReview(reviewId, content);     return ResponseEntity.created(URI.create("api/food-truck/" + foodTruckId     + "/review")).body(foodTruckReviewResponse); }</pre>
코드 설명
푸드트럭 댓글을 등록하면 Spring Data Jpa를 활용해서 데이터를 저장하고, 저장된 값을 반환하게 됩니다

### Web

**POST** /api/food-truck/{foodTruckId}/review/{reviewId}/reply

푸드트럭 리뷰 답글 등록

푸드트럭 리뷰에 답글을 등록합니다.  
푸드트럭 매니저의 권한이 필요합니다.

Parameters

Name Description

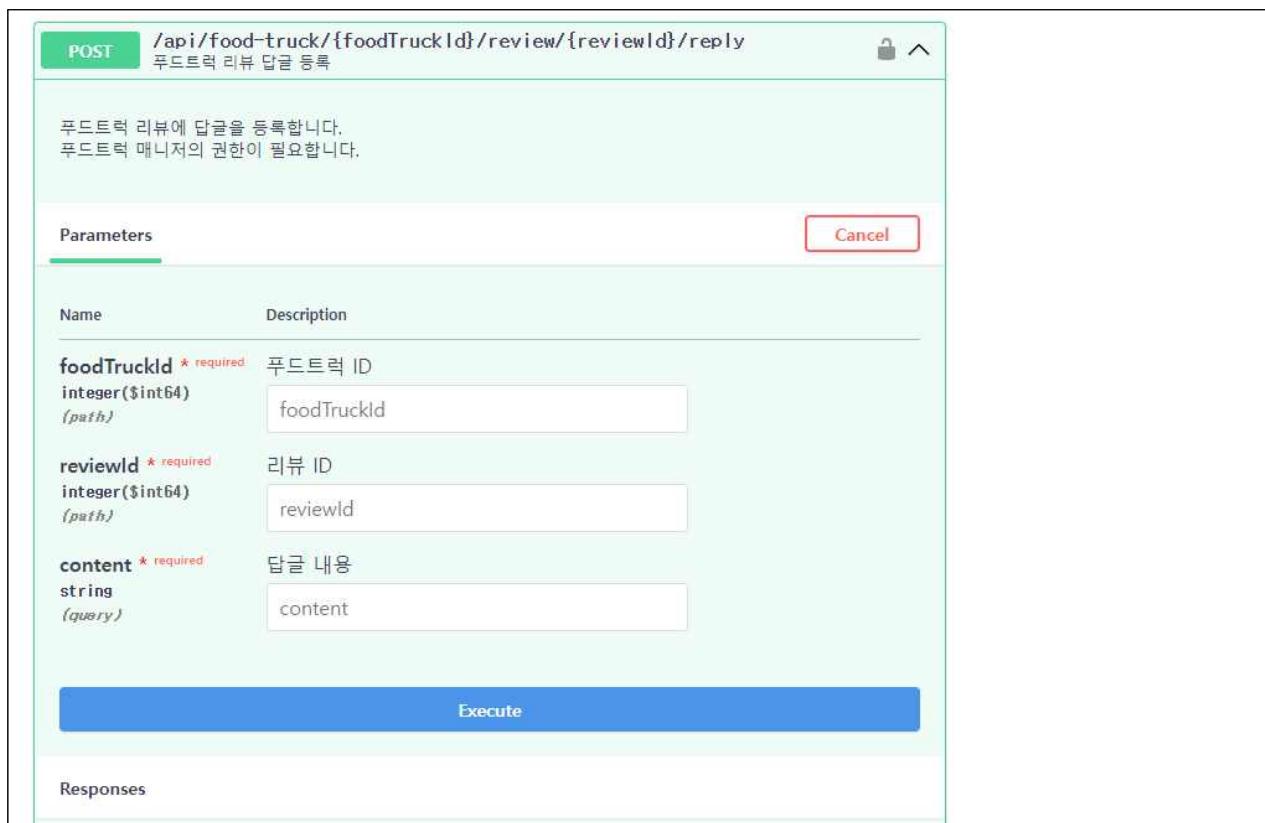
**foodTruckId** \* required 푸드트럭 ID  
integer(\$int64)  
(path)

**reviewId** \* required 리뷰 ID  
integer(\$int64)  
(path)

**content** \* required 답글 내용  
string  
(query)

**Execute**

Responses



## 10) 행사 등록하기 기능

### (1) 분석 및 설계

행사 매니저가 행사를 등록하는 기능을 제공한다. 해당 기능은 행사 매니저만 사용할 수 있으며 제목, 내용, 주소, 일자, 이미지 등을 입력받아 행사를 등록한다.

### (2) 구현

Server
EventController.java
...
<pre>    ...     @GetMapping("/register")     public String getEventRegister(Model model){         model.addAttribute("eventRequest",             new EventRequestV2("", "", "", "", "",                 LocalDateTime.now(), LocalDateTime.now(),null));         return "event/event_register";     }      @PostMapping(value ="/create",consumes ="multipart/form-data")     public String registerEventPost(         Principal principal,         EventRequestV2 eventRequest     ){         eventService.createEvent(eventRequest,principal.getName());         return "redirect:/event/my";     } ... </pre>
EventRestController.java
...
<pre>    /**      * 이벤트 생성 Rest API      *      * @param eventRequest 이벤트 생성 요청 정보      * @param principal   로그인 한 사용자 정보      * @return 생성된 이벤트 정보      */     @PostMapping(consumes =MediaType.MULTIPART_FORM_DATA_VALUE)     @PreAuthorize("hasRole('ROLE_EVENT_MANAGER')")     @Operation(summary ="이벤트 생성",description ="이벤트 생성\n이벤트 주소입력 시 카카오 우편 번호 서비스를 사용해서 주소를 입력받아야함.")     public ResponseEntity&lt;Event&gt;createEvent(         @ModelAttribute EventRequestV2 eventRequest,         Principal principal     ){         return ResponseEntity.ok(eventService.createEvent(eventRequest,principal.getName()));     } ... </pre>
EventService.java
...
<pre>    /**      * 이벤트 생성      *      * @param eventRequest 이벤트 생성 요청 폼      *                     null이면 기본 이미지 사용      * @param managerEmail 매니저 이름      * @return 생성된 이벤트      */ </pre>

```

public Event createEvent(EventRequestV2 eventRequest, String managerEmail){
    Event eventSaved = eventMapper.toEvent(eventRequest);

    User user = userRepository.findByEmail(managerEmail).orElseThrow(
        () -> new IllegalArgumentException("해당 유저가 존재하지 않습니다.")
    );
    // 기본 이미지 설정 후 이미지가 있으면 업로드
    String image = "https://picsum.photos/1600/900";
    log.info("image : {}", eventRequest.getImage());
    if (eventRequest.getImage() != null) {
        image = imageUploader.uploadImage(eventRequest.getImage()).getData().getUrl();
    }
    eventSaved.setImageUrl(image);
    eventSaved.setManager(user);
    eventSaved.setStatus(EventStatus.OPEN_FOR_APPLICATION);

    return eventRepository.save(eventSaved);
}
...

```

### EventRequestV2.kt

```

...
@Schema(description = "이벤트 요청 V2")
data class EventRequestV2(
    @field:Schema(description = "제목", example = "이벤트 제목")
    val title: String,
    @field:Schema(description = "내용", example = "이벤트 내용")
    val content: String,
    @field:Schema(description = "우편번호", example = "12345")
    val zipCode: String,
    @field:Schema(description = "도로명 주소", example = "서울시 강남구 테헤란로 123")
    val streetAddress: String,
    @field:Schema(description = "상세 주소", example = "역삼동 123-456")
    val detailAddress: String,
    @field:Schema(description = "이벤트 시작일", example = "2021-01-01T00:00:00Z")
    @field:DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
    val startAt: LocalDateTime,
    @field:Schema(description = "이벤트 종료일", example = "2021-01-01T23:59:59Z")
    @field:DateTimeFormat(iso = DateTimeFormat.ISO.DATE_TIME)
    val endAt: LocalDateTime,
    @field:Schema(description = "이미지", format = "binary")
    var image: MultipartFile?
) {

    override fun toString(): String {
        return "EventRequestV2(title='$title', " +
            "content='$content', zipCode='$zipCode', " +
            "streetAddress='$streetAddress', " +
            "detailAddress='$detailAddress', startAt=$startAt, " +
            "endAt=$endAt, image=${image?.originalFilename})"
    }
}

```

### Event.java

```

@Entity
@Getter
@Setter
@Builder
@AllArgsConstructor
@NoArgsConstructor
public class Event extends BaseEntity {
    @Id

```

```

    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String title;
    @Column(columnDefinition ="LONGTEXT")
    private String content;
    private String zipCode;
    private String streetAddress;
    private String detailAddress;
    private LocalDateTime startAt;
    private LocalDateTime endAt;
    private String imageUrl;
    @ManyToOne
    private User manager;
    @Enumerated(EnumType.STRING)
    private EventStatus status;
}

```

#### 코드 설명

이벤트 생성 요청을 받으면 이벤트 정보가 Request 객체에 담겨 이벤트 생성 컨트롤러에 맵핑되며 EventService의 createEvent 메서드를 호출하여 데이터베이스에 이벤트 생성을 요청한다.

#### Web

event-register.html

```

...
<form
th:action="@{/event/create}" method="post" enctype="multipart/form-data" th:object="${eventRequest}"
>
    <div class="form-group">
        <label for="title">이벤트 제목</label>
        <input type="text" class="form-control" id="title" name="title" th:field="*{title}" required>
    </div>

    <div class="form-group">
        <label for="content">이벤트 내용</label>
        <textarea
class="form-control" id="content" name="content" rows="3" th:field="*{content}" required></textarea>
    </div>

    <div class="form-group">
        <label for="zipCode">우편번호</label>
        <input
type="text" class="form-control" id="zipCode" name="zipCode" th:field="*{zipCode}" required>
    </div>

    <div class="form-group">
        <label for="streetAddress">도로명 주소</label>
        <input
type="text" class="form-control" id="streetAddress" name="streetAddress" th:field="*{streetAddress}" required>
    </div>

    <div class="form-group">
        <label for="detailAddress">상세 주소</label>
        <input
type="text" class="form-control" id="detailAddress" name="detailAddress" th:field="*{detailAddress}" required>
    </div>

    <div class="form-group">

```

```

        <label for="startAt">시작 일시</label>
        <input
type="datetime-local" class="form-control" id="startAt" name="startAt" th:field="*{startAt}" required>
    </div>

    <div class="form-group">
        <label for="endAt">종료 일시</label>
        <input
type="datetime-local" class="form-control" id="endAt" name="endAt" th:field="*{endAt}" required>
    </div>

    <div class="form-group">
        <label for="image">이벤트 대표 이미지</label>
        <input
type="file" class="form-control-file" id="image" name="image" th:field="*{image}" required>
    </div>

        <button type="submit" class="btn btn-primary">이벤트 생성</button>
        <button type="button" class="btn btn-secondary" onclick="location.href='/event/my'">취소하기</button>
    </form>
...

```

### 코드 설명

이벤트 생성에 필요한 정보들을 입력받아 form 태그를 통해 서버에 해당 내용들을 전송한다.

### 실행 결과

The screenshot shows a split-screen view. On the left, there is a 'Create Event' form with fields for event title, content, location, date, time, and address. On the right, the results of the event creation are displayed, showing the event details and a success message.

### IOS

#### MyEventService.swift

```

import Foundation
import Alamofire
import SwiftUI

struct MyEventService {
    static let shared = MyEventService()

    func fetchMyEvents(token: String, completion: @escaping (MyEventList?, Error?) -> Void) {
        let headers: HTTPHeaders = ["Authorization": "Bearer \(token)"]
        NetworkService.shared.performRequest("/event/my", method: .get, headers: headers)
    { (response: MyEventList?, error) in
            completion(response, error)
            print("test::: \(response) ?? NO response")
        }
    }

    func updateEvent(event: MyEventResponse, image: UIImage?, token: String, completion: @escaping (MyEventResponse?, Error?) -> Void) {
        let url = "https://connect-luck.store/api/event/\(event.id)"
        let headers: HTTPHeaders = ["Authorization": "Bearer \(token)"]

        let parameters: [String: String] = [
            "title": event.title.bound,

```

```

        "content": event.content.bound,
        "zipCode": event.zipCode.bound,
        "streetAddress": event.streetAddress.bound,
        "detailAddress": event.detailAddress.bound,
        "startAt": event.startAt.bound + "T00:00:00Z",
        "endAt": event.endAt.bound + "T23:59:59Z"
    ]
}

NetworkService.shared.uploadRequest(url, method: .patch, parameters: parameters,
headers: headers, image: image, completion: completion)
}

func addEvent(event: MyEventResponse, image: UIImage?, token: String, completion: @escaping (MyEventResponse?, Error?) -> Void) {
    let headers: HTTPHeaders = [
        "Authorization": "Bearer \(token)"
    ]

    AF.upload(
        multipartFormData: { multipartFormData in
            if let title = event.title {
                multipartFormData.append(Data(title.utf8), withName: "title")
            }
            if let content = event.content {
                multipartFormData.append(Data(content.utf8), withName: "content")
            }
            if let zipCode = event.zipCode {
                multipartFormData.append(Data(zipCode.utf8), withName: "zipCode")
            }
            if let streetAddress = event.streetAddress {
                multipartFormData.append(Data(streetAddress.utf8), withName: "streetAddress")
            }
            if let detailAddress = event.detailAddress {
                multipartFormData.append(Data(detailAddress.utf8), withName: "detailAddress")
            }
            if let startAt = event.startAt {
                multipartFormData.append(Data(startAt.utf8), withName: "startAt")
            }
            if let endAt = event.endAt {
                multipartFormData.append(Data(endAt.utf8), withName: "endAt")
            }

            if let managerName = event.managerName {
                multipartFormData.append(Data(managerName.utf8), withName: "managerName")
            }

            if let status = event.status {
                multipartFormData.append(Data(status.utf8), withName: "status")
            }

            if let imageData = image?.jpegData(compressionQuality: 0.8) {
                multipartFormData.append(imageData, withName: "image", fileName: "event.jpg", mimeType: "image/jpeg")
            }
        },

```

```

        to: "https://api.example.com/events",
        headers: headers
    ).responseDecodable(of: MyEventResponse.self) { response in
        switch response.result {
        case .success(let event):
            completion(event, nil)
        case .failure(let error):
            completion(nil, error)
        }
    }
}

}

```

ImagePicker.swift

```

import SwiftUI
import UIKit

struct ImagePicker: UIViewControllerRepresentable {
    @Binding var image: UIImage?
    @Environment(\.presentationMode) var presentationMode

    class Coordinator: NSObject, UINavigationControllerDelegate, UIImagePickerControllerDelegate {
        let parent: ImagePicker

        init(parent: ImagePicker) {
            self.parent = parent
        }

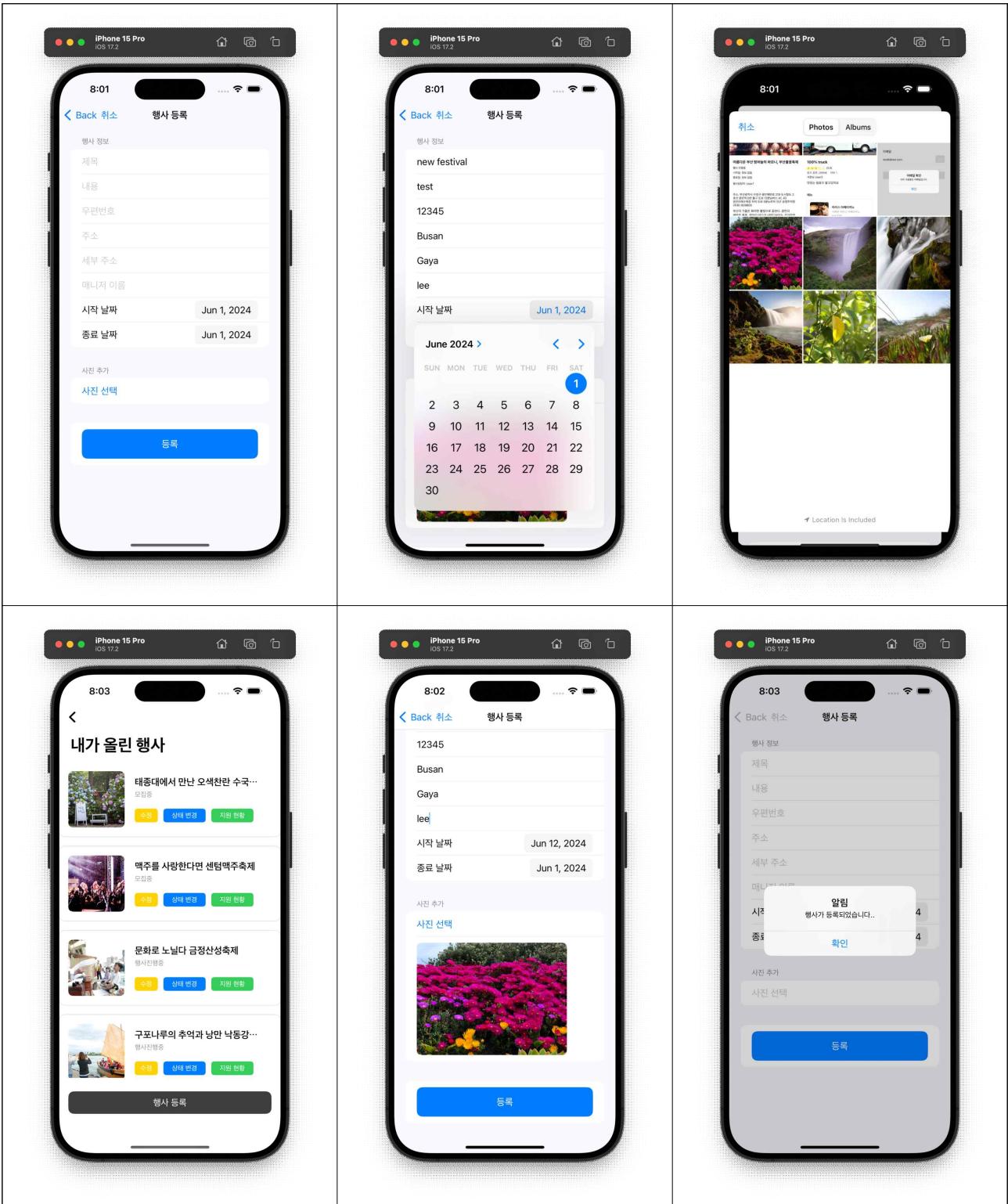
        func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
            if let uiImage = info[.originalImage] as? UIImage {
                parent.image = uiImage
            }
            parent.presentationMode.wrappedValue.dismiss()
        }
    }

    func makeCoordinator() -> Coordinator {
        Coordinator(parent: self)
    }

    func makeUIViewController(context: Context) -> UIImagePickerController {
        let picker = UIImagePickerController()
        picker.delegate = context.coordinator
        return picker
    }

    func updateUIViewController(_ uiViewController: UIImagePickerController, context: Context)
    {}
}

```



## 11) 행사 수정 기능

### (1) 분석 및 설계

행사 매니저가 행사를 수정하는 기능을 제공한다. 해당 기능은 행사 매니저만 사용할 수 있으며 제목, 내용, 주소, 일자, 이미지 등을 입력받아 행사를 수정하고 수정되지 않은 부분들은 기존에 저장되어있던 정보로 유지된다.

### (2) 구현

Server
EventController.java
...
<pre>...     @GetMapping("/edit/{id}")     public String getEventUpdate(@PathVariable("id")Long id,Model model){         model.addAttribute("event",eventService.getEvent(id));         return "event/event_update";     }      @PostMapping("/update")     @PreAuthorize("hasRole('ROLE_EVENT_MANAGER')")     public String updateEventPost(         @RequestParam("eventId")Long id,         Principal principal,         EventRequestV2 eventRequest     ){         eventService.updateEvent(id,eventRequest,principal.getName());         return "redirect:/event/my";     } ...</pre>
EventRestController.java
...
<pre>...     @PatchMapping(value ="/{id}",consumes =MediaType.MULTIPART_FORM_DATA_VALUE)     @PreAuthorize("hasRole('ROLE_EVENT_MANAGER')")     @Operation(summary ="이벤트 수정",description ="이벤트 수정")     public ResponseEntity&lt;Event&gt;updateEvent(         @Parameter(description ="이벤트 UID")@PathVariable(value ="id",required =false)Long id,         @ModelAttribute EventRequestV2 eventRequest,         Principal principal){         return ResponseEntity.ok(eventService.updateEvent(id,eventRequest,principal.getName()));     } ...</pre>
EventService
...
<pre>/**  * 이벤트 수정  *  * @param id          이벤트 UID  * @param eventRequest 이벤트 생성 요청 품  * @param managerEmail 이벤트 매니저 이메일  * @return 수정된 이벤트  */ public Event updateEvent(Long id,EventRequestV2 eventRequest,String managerEmail){     Event findEvent =eventRepository.findById(id).orElseThrow(         ()-&gt;new IllegalArgumentException("해당 이벤트가 존재하지 않습니다.")     );      if (!findEvent.getManager().getEmail().equals(managerEmail)){         throw new IllegalArgumentException("해당 이벤트의 매니저가 아닙니다.");     }</pre>

```

    if (eventRequest.getImage() != null){
        String image
        =imageUploader.uploadImage(eventRequest.getImage()).getData().getUrl();
        findEvent.setImageUrl(image);
    }

    findEvent.setTitle(eventRequest.getTitle());
    findEvent.setContent(eventRequest.getContent());
    findEvent.setZipCode(eventRequest.getZipCode());
    findEvent.setStreetAddress(eventRequest.getStreetAddress());
    findEvent.setDetailAddress(eventRequest.getDetailAddress());
    findEvent.setStartAt(eventRequest.getStartAt());
    findEvent.setEndAt(eventRequest.getEndAt());

    return eventRepository.save(findEvent);
}
...

```

#### 코드 설명

이벤트 수정 요청을 받으면 이벤트 정보가 Request 객체에 담겨 이벤트 수정 컨트롤러에 맵핑되며 EventService의 updateEvent 메서드를 호출하여 데이터베이스에 이벤트 정보 수정을 요청한다.

Web
event_update.html
...
<pre> &lt;form th:action="@{/event/update}" method="post" enctype="multipart/form-data"&gt;     &lt;!-- 이벤트 ID (숨겨진 필드로, 수정할 이벤트를 식별하기 위해 사용됩니다) --&gt;     &lt;input type="hidden" id="eventId" name="eventId" th:value="\${event.id}" /&gt;      &lt;div class="form-group"&gt;         &lt;label for="title"&gt;이벤트 제목&lt;/label&gt;         &lt;input             type="text"             class="form-control"             id="title"             name="title"             th:value="\${event.title}"             required         &gt;&lt;/div&gt;          &lt;div class="form-group"&gt;             &lt;label for="content"&gt;이벤트 내용&lt;/label&gt;             &lt;textarea                 class="form-control"                 id="content"                 name="content"                 rows="3"                 required                 th:text="\${event.content}"             &gt;&lt;/textarea&gt;         &lt;/div&gt;          &lt;div class="form-group"&gt;             &lt;label for="zipCode"&gt;우편번호&lt;/label&gt;             &lt;input                 type="text"                 class="form-control"                 id="zipCode"                 name="zipCode"                 th:value="\${event.zipCode}"                 required             &gt;&lt;/div&gt;          &lt;div class="form-group"&gt;             &lt;label for="streetAddress"&gt;도로명 주소&lt;/label&gt;             &lt;input                 type="text"                 class="form-control"                 id="streetAddress"                 name="streetAddress"                 th:value="\${event.streetAddress}"                 required             &gt;&lt;/div&gt;          &lt;div class="form-group"&gt;             &lt;label for="detailAddress"&gt;상세 주소&lt;/label&gt;             &lt;input                 type="text"                 class="form-control"                 id="detailAddress"                 name="detailAddress"                 th:value="\${event.detailAddress}"             &gt;&lt;/div&gt; </pre>

```

ss}"required>
</div>

<div class="form-group">
<label for="startAt">시작 일시</label>
<input type="datetime-local" class="form-control" id="startAt" name="startAt" th:value="${event.startAt}" required>
</div>

<div class="form-group">
<label for="endAt">종료 일시</label>
<input type="datetime-local" class="form-control" id="endAt" name="endAt" th:value="${event.endAt}" required>
</div>

<div class="form-group">
<label for="multipartFile">이벤트 대표 이미지</label>
<input type="file" class="form-control" id="multipartFile" name="multipartFile">
<!-- 기존에 업로드된 이미지가 있다면 표시 -->
<div th:if="${event.imageUrl}">
    
</div>
</div>

<button type="submit" class="btn btn-primary">이벤트 수정</button>
<button type="button" class="btn btn-secondary" onclick="location.href='/event/my'">취소하기</button>
</form>
...

```

### 코드 설명

이벤트 수정에 필요한 정보들을 입력받아 form 태그를 통해 서버에 해당 내용들을 전송한다.

### 실행 결과

	
---	--

## 12) 행사 상태 변경 기능

### (1) 분석 및 설계

행사 매니저가 행사의 상태를 변경하는 기능을 제공한다. 해당 기능은 행사 매니저만 사용할 수 있으며 행사의 상태를 모집 전, 신청 가능, 신청 만료, 행사 시작, 행사 끝 중 하나로 변경할 수 있다.

### (2) 구현

Server
EventController.java
...
@PostMapping("/statusUpdate/{id}") public String updateEventPost( @PathVariable("id")Long id, Principal principal, @RequestParam("status")EventStatus eventStatus ){ eventService.changeEventStatus(id,eventStatus,principal.getName()); return "redirect:/event/my"; }
...
EventRestController.java
...
@PostMapping("/{id}/status") @PreAuthorize("hasRole('ROLE_EVENT_MANAGER')") @Operation(summary ="이벤트 상태 변경",description ="이벤트 상태 변경") public ResponseEntity<String>changeEventStatus( @Parameter(description ="이벤트 UID")@PathVariable Long id, @Parameter(description ="이벤트 상태")@RequestParam EventStatus eventStatus, Principal principal){ eventService.changeEventStatus(id,eventStatus,principal.getName()); return ResponseEntity.ok("이벤트 상태 변경 성공"); }
...
EventService.java
...
/** * 이벤트 상태 변경 * * @param id                    이벤트 UID * @param eventStatus          이벤트 상태 * @param managerEmail         이벤트 매니저 이메일 */ public void changeEventStatus(Long id,EventStatus eventStatus,String managerEmail){ Event event =eventRepository.findById(id).orElseThrow( ()>new IllegalArgumentException("해당 이벤트가 존재하지 않습니다.")); if (!event.getManager().getEmail().equals(managerEmail)){ throw new IllegalArgumentException("해당 이벤트의 매니저가 아닙니다."); } event.setStatus(eventStatus); eventRepository.save(event); }
...
EventStatus.java
public enum EventStatus { BEFORE_APPLICATION, // 모집 전 OPEN_FOR_APPLICATION,// 신청 가능 APPLICATION_FINISHED, // 신청 만료

```

        EVENT_START,          // 행사 시작
        EVENT_END            // 행사 끝
    }

```

#### 코드 설명

이벤트 상태 변경 요청을 받으면 이벤트 정보가 Request 객체에 담겨 이벤트 상태 변경 컨트롤러에 맵핑되며 EventService의 changeEventStatus 메서드를 호출하여 데이터베이스에 이벤트 상태 변경을 요청한다.

#### Web

my\_event\_list.html

```

...
<form th:action="@{/event/statusUpdate/" + ${event.id}]" method="post">
    <div class="mt-3">
        <label for="eventStatus">이벤트 상태:</label>
        <select class="form-select" name="status" id="eventStatus">
            <option value="BEFORE_APPLICATION" th:selected="${event.status == 'BEFORE_APPLICATION'}">모집 전</option>
            <option value="OPEN_FOR_APPLICATION" th:selected="${event.status == 'OPEN_FOR_APPLICATION'}">신청 가능</option>
            <option value="APPLICATION_FINISHED" th:selected="${event.status == 'APPLICATION_FINISHED'}">신청 만료</option>
            <option value="EVENT_START" th:selected="${event.status == 'EVENT_START'}">행사 시작</option>
            <option value="EVENT_END" th:selected="${event.status == 'EVENT_END'}">행사 끝</option>
        </select>
        <button type="submit" class="btn btn-secondary btn-sm mt-2">변경</button>
    </div>
</form>
...

```

#### 코드 설명

현재 이벤트 상태를 기본 선택으로 하는 드롭다운 박스를 띄워주고 드롭다운으로 변경할 상태를 선택 후 변경 버튼을 누르면 변경할 상태가 form 태그를 통해 서버로 전송된다.

#### 실행 결과



## 태종대에서 만난 오색찬란 수국의 매력

시작일: 2024년 06월 01일

장소: 부산광역시 영도구 전망로 119 버스 186, 30, 66, 8, 88, 101, 1006 태종대(태종대 온천) 하차\n주차 태종대유원지 주차장 (유료)

담당자: User7

[신청서 조회](#) [이벤트 수정](#) [이벤트 삭제](#)

이벤트 상태: [신청 가능](#) [변경](#)



## 태종대에서 만난 오색찬란 수국의 매력

시작일: 2024년 06월 01일

장소: 부산광역시 영도구 전망로 119 버스 186, 30, 66, 8, 88, 101, 1006 태종대(태종대 온천) 하차\n주차 태종대유원지 주차장 (유료)

[모집 전](#)  
담당자: User7  
[신청 가능](#)  
[신청 만료](#)  
[행사 시작](#)  
[행사 끝](#)

[이벤트 삭제](#)

이벤트 상태: [모집 전](#) [변경](#)

## iOS

### MyEventService.swift

```
import Foundation
import Alamofire
import SwiftUI

struct MyEventService {
    static let shared = MyEventService()

    func fetchMyEvents(token: String, completion: @escaping (MyEventList?, Error?) -> Void) {
        let headers: HTTPHeaders = ["Authorization": "Bearer \(token)"]
        NetworkService.shared.performRequest("/event/my", method: .get, headers: headers)
    { (response: MyEventList?, error) in
            completion(response, error)
            print("test::: \(response) ?? \"NO response\"")
        }
    }

    func updateEvent(event: MyEventResponse, image: UIImage?, token: String, completion: @escaping (MyEventResponse?, Error?) -> Void) {
        let url = "https://connect-luck.store/api/event/\(event.id)"
        let headers: HTTPHeaders = ["Authorization": "Bearer \(token)"]
    }
```

```

let parameters: [String: String] = [
    "title": event.title.bound,
    "content": event.content.bound,
    "zipCode": event.zipCode.bound,
    "streetAddress": event.streetAddress.bound,
    "detailAddress": event.detailAddress.bound,
    "startAt": event.startAt.bound + "T00:00:00Z",
    "endAt": event.endAt.bound + "T23:59:59Z"
]

NetworkService.shared.uploadRequest(url, method: .patch, parameters: parameters,
headers: headers, image: image, completion: completion)
}

func addEvent(event: MyEventResponse, image: UIImage?, token: String, completion: @escaping (MyEventResponse?, Error?) -> Void) {
    let headers: HTTPHeaders = [
        "Authorization": "Bearer \(token)"
    ]

    AF.upload(
        multipartFormData: { multipartFormData in
            if let title = event.title {
                multipartFormData.append(Data(title.utf8), withName: "title")
            }
            if let content = event.content {
                multipartFormData.append(Data(content.utf8), withName: "content")
            }
            if let zipCode = event.zipCode {
                multipartFormData.append(Data(zipCode.utf8), withName: "zipCode")
            }
            if let streetAddress = event.streetAddress {
                multipartFormData.append(Data(streetAddress.utf8), withName: "streetAddress")
            }
            if let detailAddress = event.detailAddress {
                multipartFormData.append(Data(detailAddress.utf8), withName: "detailAddress")
            }
            if let startAt = event.startAt {
                multipartFormData.append(Data(startAt.utf8), withName: "startAt")
            }
            if let endAt = event.endAt {
                multipartFormData.append(Data(endAt.utf8), withName: "endAt")
            }

            if let managerName = event.managerName {
                multipartFormData.append(Data(managerName.utf8), withName: "managerName")
            }

            if let status = event.status {
                multipartFormData.append(Data(status.utf8), withName: "status")
            }

            if let imageData = image?.jpegData(compressionQuality: 0.8) {
                multipartFormData.append(imageData, withName: "image", fileName: "event.jpg", mimeType: "image/jpeg")
            }
        }
    )
}

```

```
        }
    },
    to: "https://api.example.com/events",
    headers: headers
).responseDecodable(of: MyEventResponse.self) { response in
    switch response.result {
    case .success(let event):
        completion(event, nil)
    case .failure(let error):
        completion(nil, error)
    }
}
}

}
```



### 13) 행사 삭제 기능

#### (1) 분석 및 설계

행사 매니저가 행사를 삭제하는 기능을 제공한다. 해당 기능은 행사 매니저만 사용할 수 있으며 등록한 행사를 삭제하는 기능을 제공한다.

#### (2) 구현

Server
EventController.java
...
@GetMapping("/delete/{id}") public String getEventDelete(@PathVariable("id")Long id,Principal principal){ eventService.deleteEvent(id,principal.getName()); return "redirect:/event/my"; } ...
EventRestController.java
...
@DeleteMapping("/{id}") @PreAuthorize("hasRole('ROLE_EVENT_MANAGER')") @Operation(summary ="이벤트 삭제",description ="이벤트 삭제") public ResponseEntity<String> deleteEvent( @Parameter(description ="이벤트 UID")@PathVariable("id")Long id, Principal principal){ eventService.deleteEvent(id,principal.getName()); return ResponseEntity.ok("이벤트 삭제 성공"); } ...
EventService.java
...
/** * 이벤트 삭제 * * @param id 이벤트 UID */ public void deleteEvent(Long id,String managerEmail){ Event event =eventRepository.findById(id).orElseThrow( ()>new IllegalArgumentException("해당 이벤트가 존재하지 않습니다.")) ); if (!event.getManager().getEmail().equals(managerEmail)){ throw new IllegalArgumentException("해당 이벤트의 매니저가 아닙니다."); } eventRepository.deleteById(id); }
...
코드 설명
이벤트 삭제 요청을 받으면 컨트롤러의 이벤트 삭제 메서드와 매팅되어 EventService 의 deleteEvent 메서드를 호출하여 데이터베이스에 이벤트 삭제를 요청한다.

#### Web

#### my\_event\_list.html

...

    <a class="btn btn-primary btn-sm" th:href="@{'/event/delete/' + \${event.id}}>이벤트 삭제</a>

...

#### 코드 설명

버튼을 클릭하면 서버의 이벤트 삭제 컨트롤러를 호출한다.

실행 결과

**내 이벤트 목록**



서울대학교 미용관연 축제  
시작일: 2023년 9월 10일

장소: 서울특별시 구로구 목동 대학교 캠퍼스  
도로명주소: 서울특별시 구로구 목동 140-1  
전화번호: 02-953-123-123, 010-12345678  
예매율: 90% (100명 중 90명이 예매) 2  
판권: 2 주제: 2023년 미용관연 축제  
인원수: 1000

[보기](#) [등록](#) [수정](#) [삭제](#)

[이벤트 상세 \[문서 다운\]](#) [\[문서\]](#)

**내 이벤트 목록**



서울대학교 미용관연 축제  
시작일: 2023년 9월 10일

장소: 서울특별시 구로구 목동 대학교 캠퍼스  
도로명주소: 서울특별시 구로구 목동 140-1  
전화번호: 02-953-123-123, 010-12345678  
예매율: 90% (100명 중 90명이 예매) 2  
판권: 2 주제: 2023년 미용관연 축제  
인원수: 1000

[보기](#) [등록](#) [수정](#) [삭제](#)

[이벤트 상세 \[문서 다운\]](#) [\[문서\]](#)

## 14) 행사 조회 기능

### (1) 분석 및 설계

모든 이용자가 사용할 수 있는 기능이며 현재 등록되어있는 모든 행사를 조회할 수 있는 기능을 제공한다. 신청 가능한 행사에 한해 신청하기 버튼이 활성화된다.

### (2) 구현

Server
EventController.java
...
@.GetMapping public String getEvent(Model model) { model.addAttribute("events", eventService.getEvents((EventStatus) null)); return "event/event-list"; }
...
EventRestController.java
...
@.GetMapping @Operation(summary = "이벤트 목록 조회", description = "모든 이벤트 목록 조회") public ResponseEntity<List<EventDetailResponse>> getEvent( @Parameter(description = "이벤트 상태 조회(Not Implement)") @RequestParam(required = false) EventStatus eventStatus ) { return ResponseEntity.ok(eventService.getEvents(eventStatus)); }
...
EventService.java
...
/** * 이벤트 목록 조회 * * @param eventStatus 이벤트 상태 조회 null이면 전체 조회 * @return 이벤트 목록 */ public List<EventDetailResponse> getEvents(EventStatus eventStatus) { List<Event> events; if (eventStatus == null) { events = eventRepository.findAll(); } else { events = eventRepository.findAllByStatus(eventStatus); } return events.stream().map(eventMapper::toEventResponse).toList(); }
...
코드 설명
이벤트 삭제 요청을 받으면 컨트롤러의 이벤트 조회 메서드와 맵핑되어 EventService 의 getEvents 메서드를 호출하여 데이터베이스에 저장되어있는 모든 이벤트 정보를 받아와 모델에 이벤트 리스트를 넣어 반환한다.

## Web

### event-list.html

...
<div class="container mt-5"> <h1 class="text-center">이벤트 목록</h1> <div class="text-left mb-3"> <input id="filterOpenForApplication" onchange="filterEvents()" type="checkbox"> <label for="filterOpenForApplication">신청 가능만 보기</label>
...

```

</div>
<div class="row mt-4" id="eventsContainer">
    <!-- 이벤트 카드 반복 -->
    <div class="col-md-4 mb-4" th:each="event : ${events}">
        <div class="card">
            
            <div class="card-body">
                <h5 class="card-title" th:text="${event.title}"></h5>
                <p class="card-text" th:text="시작일: ' +
${#temporals.format(event.startAt, 'yyyy년 MM월 dd일')}'"></p>
                <p class="card-text" th:text="장소: ' + ${event.streetAddress}"></p>
                <p class="card-text" th:text="담당자: +' + ${event.managerName}"></p>
                <p class="card-text">
                    이벤트 상태:
                    <span th:switch="${event.status}">
                        <span th:case="BEFORE_APPLICATION" th:text="모집
전"></span>
                        <span th:case="OPEN_FOR_APPLICATION" th:text="신청
가능"></span>
                        <span th:case="APPLICATION_FINISHED" th:text="신청
만료"></span>
                        <span th:case="EVENT_START" th:text="행사 시작"></span>
                        <span th:case="EVENT_END" th:text="행사 끝"></span>
                    </span>
                </p>
                <a class="btn btn-primary btn-sm" th:href="@{'/event/' +
${event.id}}">자세히 보기</a>
                <a class="btn btn-primary btn-sm" th:href="@{'/contact/' +
${event.managerName}}">문의하기</a>
                <!-- 신청하기 버튼 활성화 조건 -->
                <a class="btn btn-primary btn-sm" th:href="@{'/apply/' + ${event.id}}"
th:if="${event.status == 'OPEN_FOR_APPLICATION'}">신청하기</a>
                <!-- 비활성화 상태의 신청하기 버튼 -->
                <a class="btn btn-secondary btn-sm" disabled="disabled"
th:if="${event.status != 'OPEN_FOR_APPLICATION'}">신청하기</a>
            </div>
        </div>
    </div>
</div>
<script>
    function filterEvents() {
        var checkbox = document.getElementById('filterOpenForApplication');
        var eventsContainer = document.getElementById('eventsContainer');
        var events = eventsContainer.getElementsByClassName('col-md-4');

        for (var i = 0; i < events.length; i++) {
            var event = events[i];
            // data-status 속성을 이용하여 이벤트 상태 확인
            var status = event.getAttribute("data-status");

            // 체크박스가 체크되어 있고, 이벤트 상태가 'OPEN_FOR_APPLICATION'이 아니면
            // 카드를 숨깁니다.
            if (checkbox.checked && status !== "OPEN_FOR_APPLICATION") {
                event.style.display = "none";
            } else {
                event.style.display = ""; // 기본(display 속성을 설정하지 않음)으로 설정하여
                // 다시 표시합니다.
            }
        }
    }
</script>

```

```

        }
    }
}

</script>
...

```

**코드 설명**

서버로부터 받아온 이벤트 리스트 정보를 이용해 현재 이벤트들의 정보를 웹에 출력한다.

**실행 결과**

**IOS**

EventViewModel.swift

```

import Foundation
import Combine

class EventViewModel: ObservableObject {
    @Published var events: [EventResponse] = []

    @Published var isLoading: Bool = false
    @Published var errorMessage: String? = nil

    @Published var searchQuery: String = ""

    init() {
        fetchEvents()
    }

    func fetchEvents() {
        self.isLoading = true
        EventService.shared.fetchEvents { [weak self] events, error in
            DispatchQueue.main.async {
                self?.isLoading = false
                if let error = error {
                    self?.errorMessage = error.localizedDescription
                } else {
                    self?.events = events ?? []
                }
            }
        }
    }

    var filteredEvents: [EventResponse] {
        if searchQuery.isEmpty {
            return events
        } else {
            return events.filter { event in
                event.title?.lowercased().contains(searchQuery.lowercased()) ?? false
            }
        }
    }
}

```

```

        }
    }

}

EventService.swift

import Foundation
import Alamofire

struct EventService {
    static let shared = EventService()

    func fetchEvents(completion: @escaping (EventList?, Error?) -> Void) {
        NetworkService.shared.performRequest("/event", method: .get) { (response: EventList?, error) in
            completion(response, error)
        }
    }

    func fetchMyEvents(token: String, completion: @escaping (EventList?, Error?) -> Void) {
        let headers: HTTPHeaders = ["Authorization": "Bearer \(token)"]
        NetworkService.shared.performRequest("/event/my", method: .get, headers: headers)
    { (response: EventList?, error) in
            completion(response, error)
            print(response ?? "NO response")
        }
    }
}

EventResponse.swift

import Foundation

struct EventResponse: Codable, Identifiable {
    let id: Int?
    let title, content, zipCode, streetAddress: String?
    let detailAddress, startAt, endAt: String?
    let imageURL: String?
    let managerName, status: String?

    enum CodingKeys: String, CodingKey {
        case id, title, content, zipCode, streetAddress, detailAddress, startAt, endAt
        case imageURL = "imageUrl"
        case managerName, status
    }
}

typealias EventList = [EventResponse]

```

6:30

어떤 행사를 찾고 계세요?

검색

필터: 모든 상태

-  태종대에서 만난 오색찬란...  
부산광역시 영도구 전망로 119  
모집중  
\*\* 2024년 수국축제는 열...
-  맥주를 사랑한다면 센텀맥...  
부산광역시 해운대구 수영강변대로 120  
모집중  
农药 속에서 젊음의 열기...
-  문화로 노닐다 금정산성축제  
부산광역시 금정구 창전온천천로 144  
행사진행중  
5월은 축제의 계절, 부산...
-  구포나루의 추억과 낭만 낙...  
부산광역시 북구 낙동대로1739번길 2...  
행사진행중  
싱그러운 봄, 솔솔 부는 강...

6:30

어떤 행사를 찾고 계세요?

검색

필터: 모든 상태

-  부산광역시 사상구 삼락동 29-46 삼...  
행사진행중  
무더위와 태풍에 지친 일상...
-  명품 토마토를 사수하라!...  
부산광역시 강서구 체육공원로 43(강...  
행사진행중  
3월이 되면 화려한 개막식...
-  몽환의 순간을 담다. 낙...  
낙동제방 벚꽃길 일원, 르네시떼 야외...  
행사진행중  
农药 만개한 벚꽃이 12킬로미...
-  사상강변축제  
부산광역시 사상구 삼락생태공원 럭비...  
행사진행중  
낙동제방 벚꽃과 더불어 열...

6:31

어떤 행사를 찾고 계세요?

검색

필터: 모든 상태

-  모든 상태
-  모집 대기
-  모집 중
-  모집 미감
-  행사 진행중
-  행사 종료

6:31

어떤 행사를 찾고 계세요?

검색

필터: 행사 진행중

-  문화로 노닐다 금정산성축제  
부산광역시 금정구 창전온천천로 144  
행사진행중  
5월은 축제의 계절, 부산...
-  구포나루의 추억과 낭만 낙...  
부산광역시 북구 낙동대로1739번길 2...  
행사진행중  
싱그러운 봄, 솔솔 부는 강...
-  아름다운 부산 밤하늘의 하모니, 부산불꽃축제  
부산광역시 수영구 광안해변로 219 도시철도 2호선 광안역 5번 출구 도보 12분\버스 41, 42 광안리해수욕장 하차 도보 5분\주차 인근 공영주차장 (유료) 609800  
부산의 가을은 화려한 불빛으로 꽂힌다. 광안리 해변은 물론, 광안리 바다가 내려다보이는 곳이라면 부산 곳곳에 사람들이 모인다. 사람들의 수많은 이목이 광안리 바다와 하늘에 활짝 핀 불꽃에 하나로 모였다. 지금은 바로 부산불꽃축제의 시간이다.\n\n반짝이는 광안대교가 보이기 시작한다. 광안대교가 가까워질수록 사람들은 점점 모여들어서 점차 발 디딜 틈도 보이지 않는다. 하지만 가을밤 기분 좋을 정도로 쌀쌀한 바닷바람을 맞으며 불꽃을 기다리는
-  오색찬란 연등티널 삼광사...  
부산광역시 부산진구 초읍천로43번길...  
행사진행중  
부처님 오신 날을 전후로...

6:31

어떤 행사를 찾고 계세요?

new

필터: 모든 상태

-  2023 New Year Conce...  
부산광역시 영도구 힐지로79번길 6  
행사종료  
새로운 2023년을 맞이하...

7:06

행사 정보



아름다운 부산 밤하늘의 하모니, 부산불꽃축제

행사 진행중  
시작일: 정보 없음  
종료일: 정보 없음  
행사담당자: User7

주소: 부산광역시 수영구 광안해변로 219 도시철도 2호선 광안역 5번 출구 도보 12분\버스 41, 42 광안리해수욕장 하차 도보 5분\주차 인근 공영주차장 (유료) 609800

부산의 가을은 화려한 불빛으로 꽂힌다. 광안리 해변은 물론, 광안리 바다가 내려다보이는 곳이라면 부산 곳곳에 사람들이 모인다. 사람들의 수많은 이목이 광안리 바다와 하늘에 활짝 핀 불꽃에 하나로 모였다. 지금은 바로 부산불꽃축제의 시간이다.\n\n반짝이는 광안대교가 보이기 시작한다. 광안대교가 가까워질수록 사람들은 점점 모여들어서 점차 발 디딜 틈도 보이지 않는다. 하지만 가을밤 기분 좋을 정도로 쌀쌀한 바닷바람을 맞으며 불꽃을 기다리는

<b>Android</b> <b>class HomeFragment</b> <pre>eventAdapter = EventAdapter(requireContext(), emptyList()) binding.eventListView.adapter = eventAdapter  // EventViewModel에서 초기 데이터 가져와서 ListView에 표시 eventViewModel.loadEvents() // 초기 데이터 로드</pre>
<b>class EventAdapter.kt</b> <pre>/*  * RecyclerView의 각 아이템을 표시하는 ViewHolder 클래스입니다.  */  inner class EventViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {     val imageView: ImageView = itemView.findViewById(R.id.imageView)     val textView: TextView = itemView.findViewById(R.id.title1)     val textView1: TextView = itemView.findViewById(R.id.streetAddress)     val textView2: TextView = itemView.findViewById(R.id.zipcode)     val textView3: TextView = itemView.findViewById(R.id.startAt)     val textView4: TextView = itemView.findViewById(R.id.endAt)     val textView5: TextView = itemView.findViewById(R.id.status) }</pre>
<b>class EventAdapter.kt</b> <pre>/*  * EventAdapter 클래스는 RecyclerView.Adapter를 상속하여 이벤트 목록을 표시하는데 필요한 기능  * 제공합니다.  * @param context 앱의 Context  * @param eventList 이벤트 목록  */ class EventAdapter(private val context: Context, private var eventList: List&lt;EventHeader&gt;) :     RecyclerView.Adapter&lt;EventAdapter.EventViewHolder&gt;() {      /**      * 새로운 ViewHolder 객체를 생성합니다.      * @param parent RecyclerView의 부모 ViewGroup      * @param viewType 뷰의 타입      * @return 생성된 EventViewHolder 객체      */     override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): EventViewHolder {         val view = LayoutInflater.from(parent.context).inflate(R.layout.item_event, parent, false)         return EventViewHolder(view)     }      /**      * ViewHolder에 데이터를 바인딩합니다.      * @param holder ViewHolder 객체      * @param position 아이템의 위치      */     override fun onBindViewHolder(holder: EventViewHolder, position: Int) {</pre>

```

    val event = eventList[position]
    Glide.with(context)
        .load(event.imageUrl)
        .into(holder.imageView)
    holder.textView.text = event.title
    holder.textView1.text = event.streetAddress
    holder.textView2.text = event.zipCode
    holder.textView3.text = event.startAt
    holder.textView4.text = event.endAt
    holder.textView5.text = event.status

}

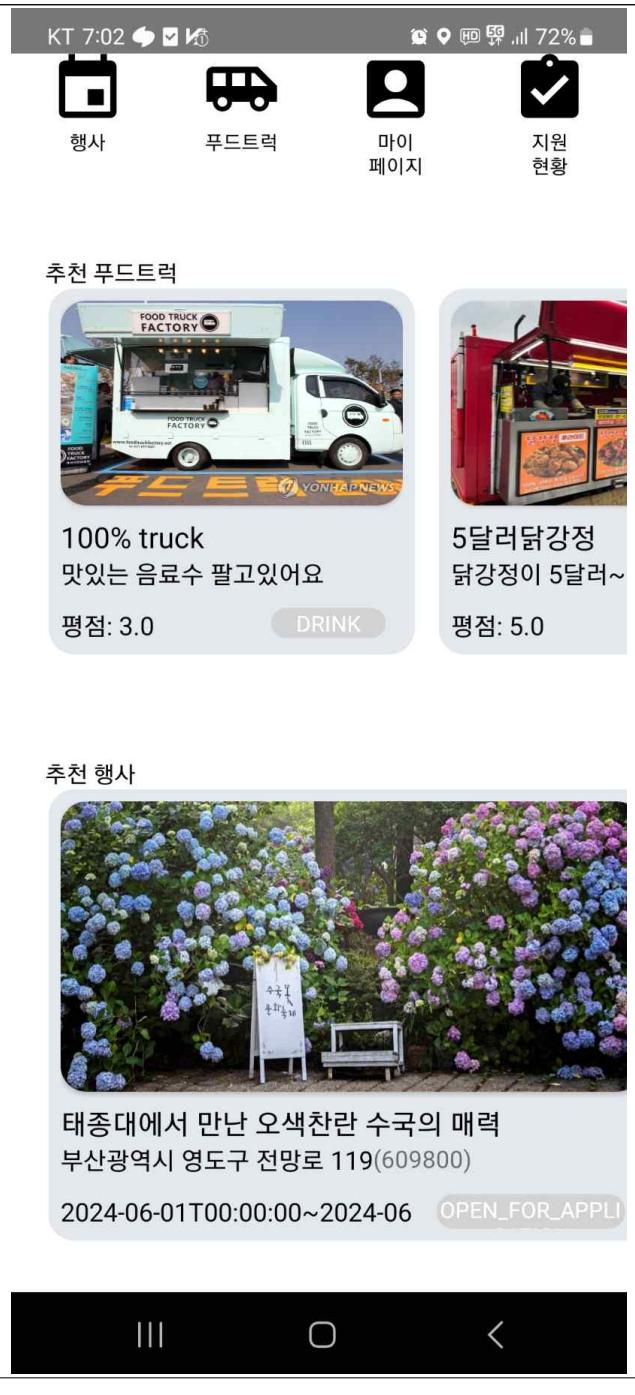
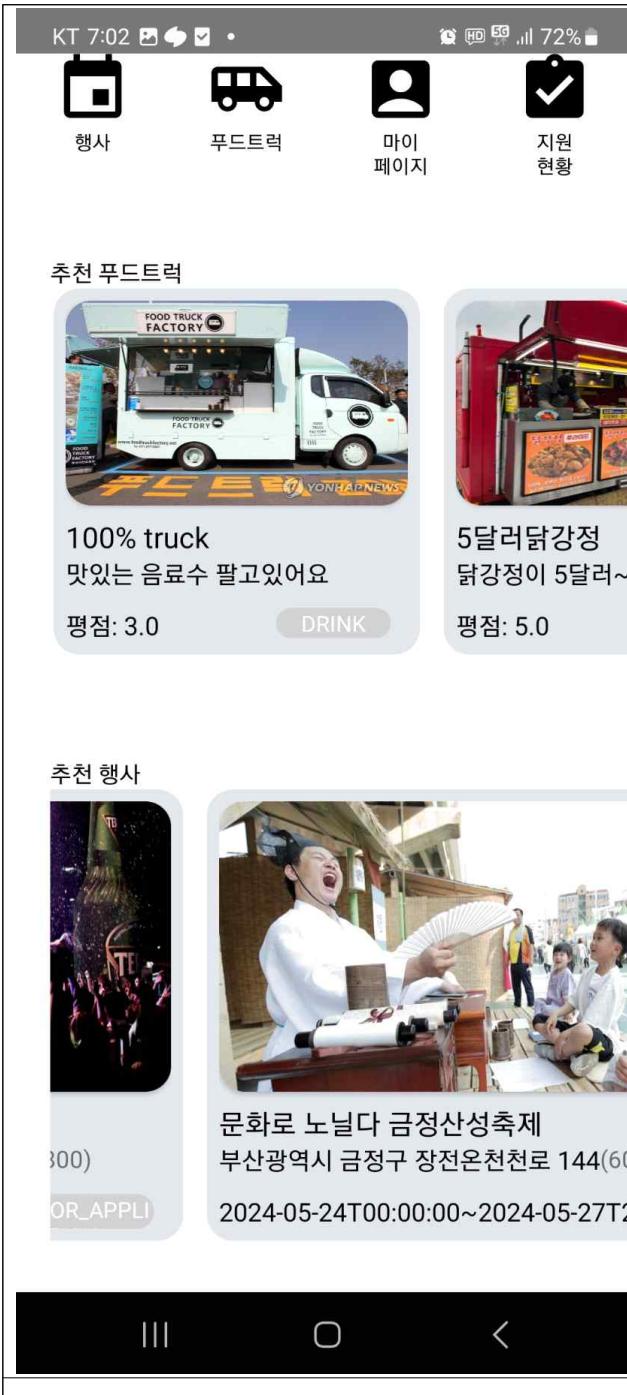
class EventAdapter.kt
/**
 * RecyclerView에 표시할 아이템의 총 개수를 반환합니다.
 * @return 아이템 개수
 */
override fun getItemCount(): Int {
    return eventList.size
}

class EventAdapter.kt
/**
 * 이벤트 목록을 업데이트합니다.
 * @param newEventList 새로운 이벤트 목록
 */
fun updateData(newEventList: List<EventHeader>) {
    eventList = newEventList
    notifyDataSetChanged()
}

class EventViewModel
// 이벤트 데이터를 로드하는 메서드
fun loadEvents() {
    viewModelScope.launch {
        val response = eventApi.getEvents()
        if (response.isSuccessful) {
            _eventList.postValue(response.body())
        } else {
            // 에러 처리
        }
    }
}

interface EventApi
/**
 * 모든 이벤트를 로드합니다.
 */
@GET("/api/event")
suspend fun getEvents(): Response<List<EventHeader>>
이벤트 데이터를 비동기적으로 로드, viewModelScope.launch를 사용하여 Coroutine을 시작하고,
eventApi.getEvents()를 호출하여 네트워크를 통해 이벤트 데이터를 가져옵니다.

```



## 15) 행사 자세히 보기 기능

### (1) 분석 및 설계

모든 이용자가 사용 가능한 기능이며 선택한 행사를 자세하게 모두 조회하는 기능을 제공한다.

### (2) 구현

Server
EventController.java
...
@GetMapping("/{id}") public String getEventDetail(@PathVariable("id")Long id,Model model){ model.addAttribute("event",eventService.getEvent(id)); return "event/event-form"; }
...
EventRestController.java
...
@GetMapping("/{id}") @Operation(summary ="이벤트 상세 조회",description ="이벤트 ID로 상세 조회") public ResponseEntity<EventDetailResponse>getEvent(@PathVariable Long id){ return ResponseEntity.ok(eventService.getEvent(id)); }
...
EventService.java
...
/** * 이벤트 상세 조회 * * @param id 이벤트 ID * @return 이벤트 상세 정보 */ public EventDetailResponse getEvent(Long id){ Event event =eventRepository.findById(id).orElseThrow( ()-> <b>new</b> IllegalStateException("해당 이벤트가 존재하지 않습니다.")) ; return eventMapper.toEventResponse(event); }
...
코드 설명
이벤트 삭제 요청을 받으면 컨트롤러의 이벤트 상세 조회 메서드와 매핑되어 EventService 의 getEvent 메서드를 호출하여 데이터베이스에서 이벤트 id 로 이벤트 정보를 받아와 모델에 해당 이벤트 정보를 넣어 반환한다.

## Web

### event-from.html

```
<div class="container mt-5">  
    <h1 class="text-center">이벤트 상세 조회</h1>  
  
    <h2 th:text="${event.title}"></h2>  
  
    <!-- 이벤트 대표 이미지 -->  
    <div th:if="${event.imageUrl}">  
          
    </div>
```

```
<!-- 이벤트 담당자 이름 -->
<p><strong>담당자:</strong><span th:text="${event.managerName}"></span></p>

<!-- 이벤트 상태 -->
<p class="card-text">
    <strong>이벤트 상태:</strong>
    <span th:switch="${event.status}">
        <span th:case="BEFORE_APPLICATION" th:text="모집 전"></span>
        <span th:case="OPEN_FOR_APPLICATION" th:text="신청 가능"></span>
        <span th:case="APPLICATION_FINISHED" th:text="신청 만료"></span>
        <span th:case="EVENT_START" th:text="행사 시작"></span>
        <span th:case="EVENT_END" th:text="행사 끝"></span>
    </span>
</p>

<!-- 이벤트 내용 -->
<div>
    <p><strong>이벤트 내용:</strong>
        <span th:text="${event.content}"></span>
    </p>
</div>

<!-- 이벤트 주소 -->
<div>
    <p><strong>이벤트 주소:</strong>
        <span th:text="${event.zipCode} + ', ' + ${event.streetAddress} + ', ' +
${event.detailAddress}"></span>
    </p>
</div>

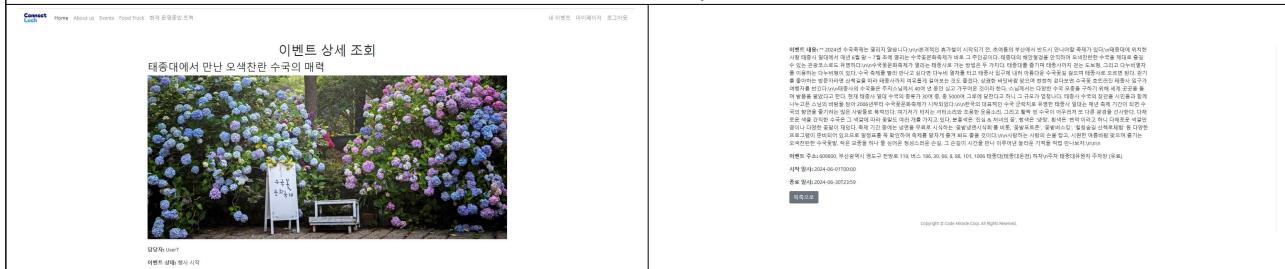
<!-- 이벤트 시작/종료 일시 -->
<div>
    <p><strong>시작 일시:</strong><span th:text="${event.startAt}"></span></p>
    <p><strong>종료 일시:</strong><span th:text="${event.endAt}"></span></p>
</div>

<button type="button" class="btn
btn-secondary" onclick="location.href='/event'">목록으로</button>
</div>
```

코드 설명

서버로부터 받아온 event 정보를 이용해 현재 이벤트의 정보를 웹에 표시한다.

## 실행 결과



IOS

## EventViewModel.swift

```
import Foundation
import Combine

class EventViewModel: ObservableObject {
    @Published var events: [EventResponse] = []
```

```

@Published var isLoading: Bool = false
@Published var errorMessage: String? = nil

@Published var searchQuery: String = ""

init() {
    fetchEvents()
}

func fetchEvents() {
    self.isLoading = true
    EventService.shared.fetchEvents { [weak self] events, error in
        DispatchQueue.main.async {
            self?.isLoading = false
            if let error = error {
                self?.errorMessage = error.localizedDescription
            } else {
                self?.events = events ?? []
            }
        }
    }
}

var filteredEvents: [EventResponse] {
    if searchQuery.isEmpty {
        return events
    } else {
        return events.filter { event in
            event.title?.lowercased().contains(searchQuery.lowercased()) ?? false
        }
    }
}

```

EventService.swift

```

import Foundation
import Alamofire

struct EventService {
    static let shared = EventService()

    func fetchEvents(completion: @escaping (EventList?, Error?) -> Void) {
        NetworkService.shared.performRequest("/event", method: .get) { (response: EventList?, error) in
            completion(response, error)
        }
    }

    func fetchMyEvents(token: String, completion: @escaping (EventList?, Error?) -> Void) {
        let headers: HTTPHeaders = ["Authorization": "Bearer \(token)"]
        NetworkService.shared.performRequest("/event/my", method: .get, headers: headers) { (response: EventList?, error) in
            completion(response, error)
            print(response ?? "NO response")
        }
    }
}

```

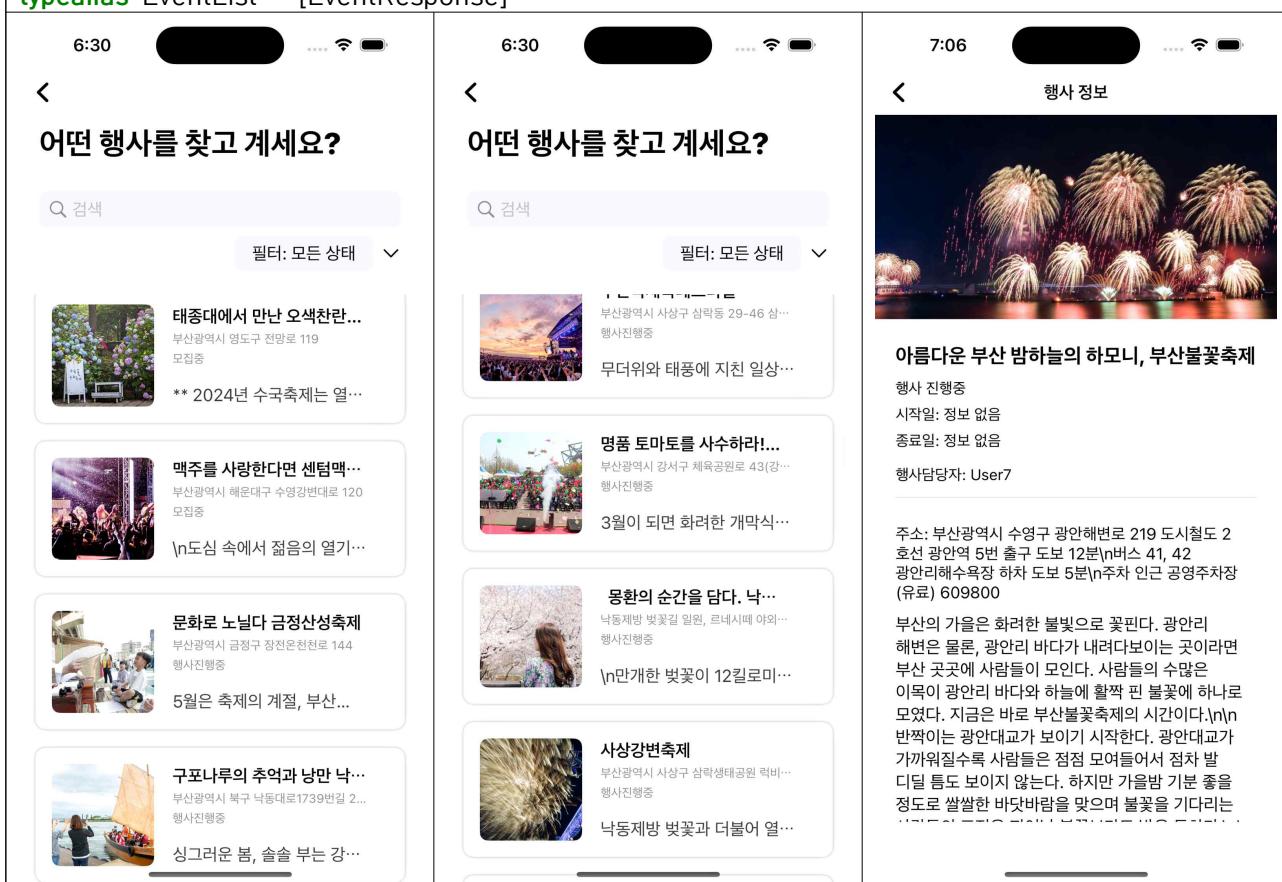
EventResponse.swift

```
import Foundation
```

```
struct EventResponse: Codable, Identifiable {
    let id: Int?
    let title, content, zipCode, streetAddress: String?
    let detailAddress, startAt, endAt: String?
    let imageURL: String?
    let managerName, status: String?

    enum CodingKeys: String, CodingKey {
        case id, title, content, zipCode, streetAddress, detailAddress, startAt, endAt
        case imageURL = "imageURL"
        case managerName, status
    }
}

typealias EventList = [EventResponse]
```



## Android

```
class EventDescriptionFragment.kt
```

```
/*
 * Fragment의 뷰를 생성합니다.
 * @param inflator 레이아웃 인플레이터
 * @param container 뷰그룹 컨테이너
 * @param savedInstanceState 저장된 인스턴스 상태
 * @return 생성된 뷰
 */
override fun onCreateView(
    inflator: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
```

```

        _binding = FragmentEventDescriptionBinding.inflate(inflater, container, false)
        return binding.root
    }

    class EventDescriptionFragment.kt

    /**
     * 뷰가 생성된 후 호출되며, 뷰 모델 초기화 및 이벤트 데이터 설정을 처리합니다.
     * @param view 생성된 뷰
     * @param savedInstanceState 저장된 인스턴스 상태
     */
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        // ViewModel을 초기화합니다.
        eventViewModel = ViewModelProvider(requireActivity()).get(EventViewModel::class.java)

        // 전달된 이벤트 데이터를 가져옵니다.
        val event = arguments?.getSerializable("selectedEvent") as? EventHeader

        event?.let { selectedEvent ->
            Log.d("DescriptionFragment", "Selected Event: ${selectedEvent.title}")
            binding.textView1.text = selectedEvent.streetAddress
            binding.textView2.text = selectedEvent.title
            binding.textView3.text = selectedEvent.startAt
            binding.textView4.text = selectedEvent.endAt
            binding.textView5.text = selectedEvent.managerName
            binding.textView7.text = selectedEvent.content
            binding.textView8.text = selectedEvent.detailAddress
            binding.textView9.text = selectedEvent.zipCode

            // 이미지를 로딩합니다.
            Glide.with(requireContext())
                .load(selectedEvent.imageUrl)
                .into(binding.imageView2)

            // apply 버튼 클릭 시 이벤트 데이터를 번들로 전달하여 네비게이션합니다.
            binding.applybtn.setOnClickListener {
                val bundle = Bundle().apply {
                    putSerializable("selectedEvent", selectedEvent)
                }
                findNavController().navigate(R.id.action_eventDescriptionFragment_to_eventApplicationFragment, bundle)
            }
        } ?: run {
            Log.d("DescriptionFragment", "No selected event")
        }
    }

    class EventDescriptionFragment.kt

    /**
     * 뷰가 파괴될 때 호출되며, 바인딩 객체를 null로 설정합니다.
     */
    override fun onDestroyView() {
        super.onDestroyView()
        _binding = null
    }

    class EventAdapterList.kt

    binding.recyclerViewFoodTrucks.adapter = foodTruckAdapter

    val token = MySharedPreferences.getToken(requireContext())

```

```

viewModel.fetchMyTruck(token)

viewModel.myTruckList.observe(viewLifecycleOwner) { foodTruckHeaders ->
    foodTruckAdapter.updateData(foodTruckHeaders)
}

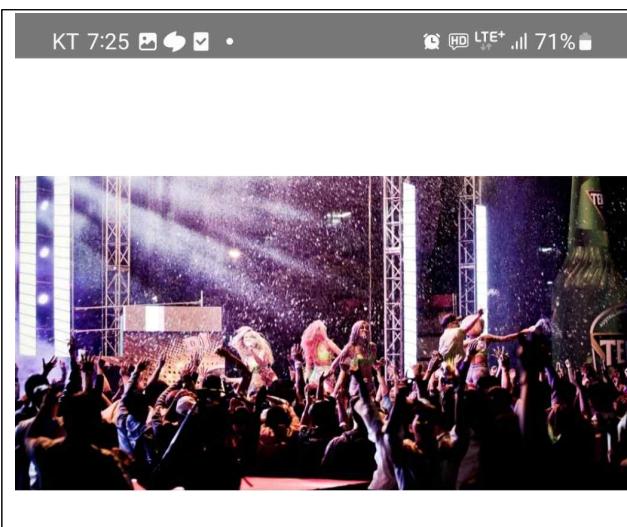
class MyPageViewModel.kt

fun fetchMyTruck(token: String) {
    viewModelScope.launch {
        try {
            val response = foodTruckApi.findMyTruck("Bearer $token")
            if (response.isSuccessful) {
                _myTruckList.value = response.body()
            } else {
                // Handle error
            }
        } catch (e: Exception) {
            // Handle error
        }
    }
}

interface FoodTruckApi.kt

@GET("/api/food-truck/my")
suspend fun findMyTruck(
    @Header("Authorization") token: String,
): Response<List<FoodTruckHeader>>

```



부산광역시 해운대구 수영강변대로 120

맥주를 사랑한다면 센텀맥주축제

2024-05-17T00:00:00 ~ 2024-05-26T23:59:00

User7

## 이벤트 내용

\n도심 속에서 젊음의 열기를 대방출하라!\n\n회를 더할수록 뜨거워지는 센텀맥주축제는 동서양 남녀노소 장벽 없이 모두 한마음으로 마음껏 즐길 수 있는 축제의 장이 되었다. 페이스페인팅으로 참가 분위기는 업, 생맥주 무료 시음으로 기분 또한 업.\n다양한 이벤트와 EDM파티는 참가한 모든 이들을 충분히 흥분의 도가니 속으로 빠져들게 한다. 그동안 쌓인 스트레스, 센텀맥주축제에서 시원하게 날려버리자.\n\n\*\*2024  
센텀맥주축제\n2024.5.30.(목) ~ 6.9.(일)

## 상세주소

도시철도 2호선 센텀시티역 12번 출구 도보  
12분\n버스 115, 181, 307, 해운대구 3-1, 해운대구 3-2  
시청자미디어센터 KNN방송국 히든드론 오브



부산광역시 중구 중앙대로 2

## 구립합창단 정기발표회

2024-09-01T19:00:00 ~ 2024-09-30T20:30:00

User9

## 이벤트 내용

구립합창단 합창+특별출연진 문화공연

### 상세주소

롯데백화점 광복점 문화홀

48944

신청하기



## 16) 내 행사 조회 기능

### (1) 분석 및 설계

행사 매니저만 사용 가능한 기능이다. 본인이 등록한 행사들의 목록을 조회할 수 있는 기능을 제공한다.

### (2) 구현

Server
EventController.java
...
@GetMapping("/my") public String getMyFoodTruck(Model model, Principal principal){ model.addAttribute("events", eventService.getEvents(principal.getName())); return "event/my_event_list"; }
...
EventRestController.java
...
@GetMapping("/my") @PreAuthorize("hasRole('ROLE_EVENT_MANAGER')") @Operation(summary ="내 이벤트 목록 조회", description ="내가 생성한 이벤트 목록 조회") public ResponseEntity<List<EventDetailResponse>> getMyEvent(Principal principal){ return ResponseEntity.ok(eventService.getEvents(principal.getName())); }
...
EventService.java
...
/** * email로 이벤트 목록 조회 * * @param email 이벤트 매니저 email * @return 이벤트 목록 */ public List<EventDetailResponse> getEvents(String email){ List<Event> events; User user = userRepository.findByEmail(email).orElseThrow( () -> new IllegalArgumentException("해당 유저가 존재하지 않습니다.")); events = eventRepository.findAllByManager(user); return events.stream().map(eventMapper::toEventResponse).toList(); }
...
코드 설명
내 이벤트 조회 요청을 받으면 컨트롤러의 내 이벤트 조회 메서드와 매핑되어 EventService에 인자를 String Email로 받는 getEvent 메서드로 오버로딩되어 데이터베이스에서 이벤트 id로 이벤트 정보를 받아와 모델에 해당 이벤트 정보를 넣어 반환한다.

Web
my_event_list.html
...
<div class="col-md-4 mb-4" th:each="event : \${events}"> <div class="card">  <div class="card-body"> <h5 class="card-title" th:text="\${event.title}"></h5> <p class="card-text" th:text="'"시작일: ' + \${#temporals.format(event.startAt, 'yyyy년 MM월 dd일')}'"></p> <p class="card-text" th:text="'"장소: ' + \${event.streetAddress} + ' ' +

```

event.detailAddress}"></p>
    <p class="card-text" th:text="">담당자: +'${event.managerName}'</p>

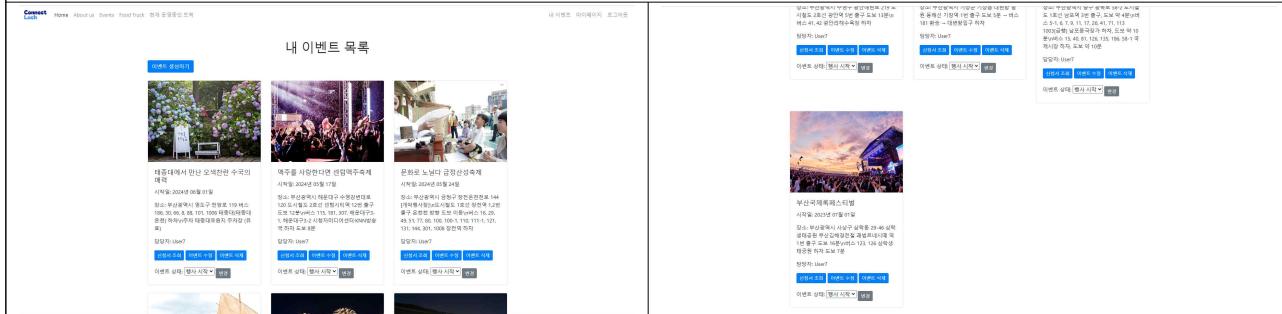
조회</a>           <a class="btn btn-primary btn-sm" th:href="@{'/apply/list/' + ${event.id}}">신청서
수정</a>           <a class="btn btn-primary btn-sm" th:href="@{'/event/edit/' + ${event.id}}">이벤트
                     <a class="btn btn-primary btn-sm" th:href="@{'/event/delete/' + ${event.id}}">이벤트 삭제</a>
                     <form method="post" th:action="@{'/event/statusUpdate/' + ${event.id}}">
                        <div class="mt-3">
                            <label for="eventStatus">이벤트 상태:</label>
                            <select class="form-select" id="eventStatus" name="status">
                                <option th:selected="${event.status == 'BEFORE_APPLICATION' ? true : false}" value="BEFORE_APPLICATION">모집 전</option>
                                <option th:selected="${event.status == 'OPEN_FOR_APPLICATION' ? true : false}" value="OPEN_FOR_APPLICATION">신청 가능</option>
                                <option th:selected="${event.status == 'APPLICATION_FINISHED' ? true : false}" value="APPLICATION_FINISHED">신청 만료</option>
                                <option th:selected="${event.status == 'EVENT_START' ? true : false}" value="EVENT_START">행사 시작</option>
                                <option th:selected="${event.status == 'EVENT_END' ? true : false}" value="EVENT_END">행사 끝</option>
                            </select>
                            <button class="btn btn-secondary btn-sm mt-2" type="submit">변경</button>
                        </div>
                    </form>
                </div>
            </div>
        </div>
    ...

```

### 코드 설명

받아온 이벤트 정보를 이용해 내가 등록한 이벤트 정보를 th:each 를 사용해 모두 웹 페이지에 표시한다.

### 실행 결과



### IOS

#### MyEventService.swift

```

import Foundation
import Alamofire
import SwiftUI

struct MyEventService {
    static let shared = MyEventService()

    func fetchMyEvents(token: String, completion: @escaping (MyEventList?, Error?) -> Void) {
        let headers: HTTPHeaders = ["Authorization": "Bearer \(token)"]

```

```

        NetworkService.shared.performRequest("/event/my", method: .get, headers: headers)
    { (response: MyEventList?, error) in
        completion(response, error)
        print("test::: \(response)" ?? "NO response")
    }
}

func updateEvent(event: MyEventResponse, image: UIImage?, token: String, completion: @escaping (MyEventResponse?, Error?) -> Void) {
    let url = "https://connect-luck.store/api/event/\(event.id)"
    let headers: HTTPHeaders = ["Authorization": "Bearer \(token)"]

    let parameters: [String: String] = [
        "title": event.title.bound,
        "content": event.content.bound,
        "zipCode": event.zipCode.bound,
        "streetAddress": event.streetAddress.bound,
        "detailAddress": event.detailAddress.bound,
        "startAt": event.startAt.bound + "T00:00:00Z",
        "endAt": event.endAt.bound + "T23:59:59Z"
    ]
}

NetworkService.shared.uploadRequest(url, method: .patch, parameters: parameters, headers: headers, image: image, completion: completion)
}

func addEvent(event: MyEventResponse, image: UIImage?, token: String, completion: @escaping (MyEventResponse?, Error?) -> Void) {
    let headers: HTTPHeaders = [
        "Authorization": "Bearer \(token)"
    ]

    AF.upload(
        multipartFormData: { multipartFormData in
            if let title = event.title {
                multipartFormData.append(Data(title.utf8), withName: "title")
            }
            if let content = event.content {
                multipartFormData.append(Data(content.utf8), withName: "content")
            }
            if let zipCode = event.zipCode {
                multipartFormData.append(Data(zipCode.utf8), withName: "zipCode")
            }
            if let streetAddress = event.streetAddress {
                multipartFormData.append(Data(streetAddress.utf8), withName: "streetAddress")
            }
            if let detailAddress = event.detailAddress {
                multipartFormData.append(Data(detailAddress.utf8), withName: "detailAddress")
            }
            if let startAt = event.startAt {
                multipartFormData.append(Data(startAt.utf8), withName: "startAt")
            }
            if let endAt = event.endAt {
                multipartFormData.append(Data(endAt.utf8), withName: "endAt")
            }

            if let managerName = event.managerName {
                multipartFormData.append(Data(managerName.utf8), withName: "managerName")
            }
        }
    )
}

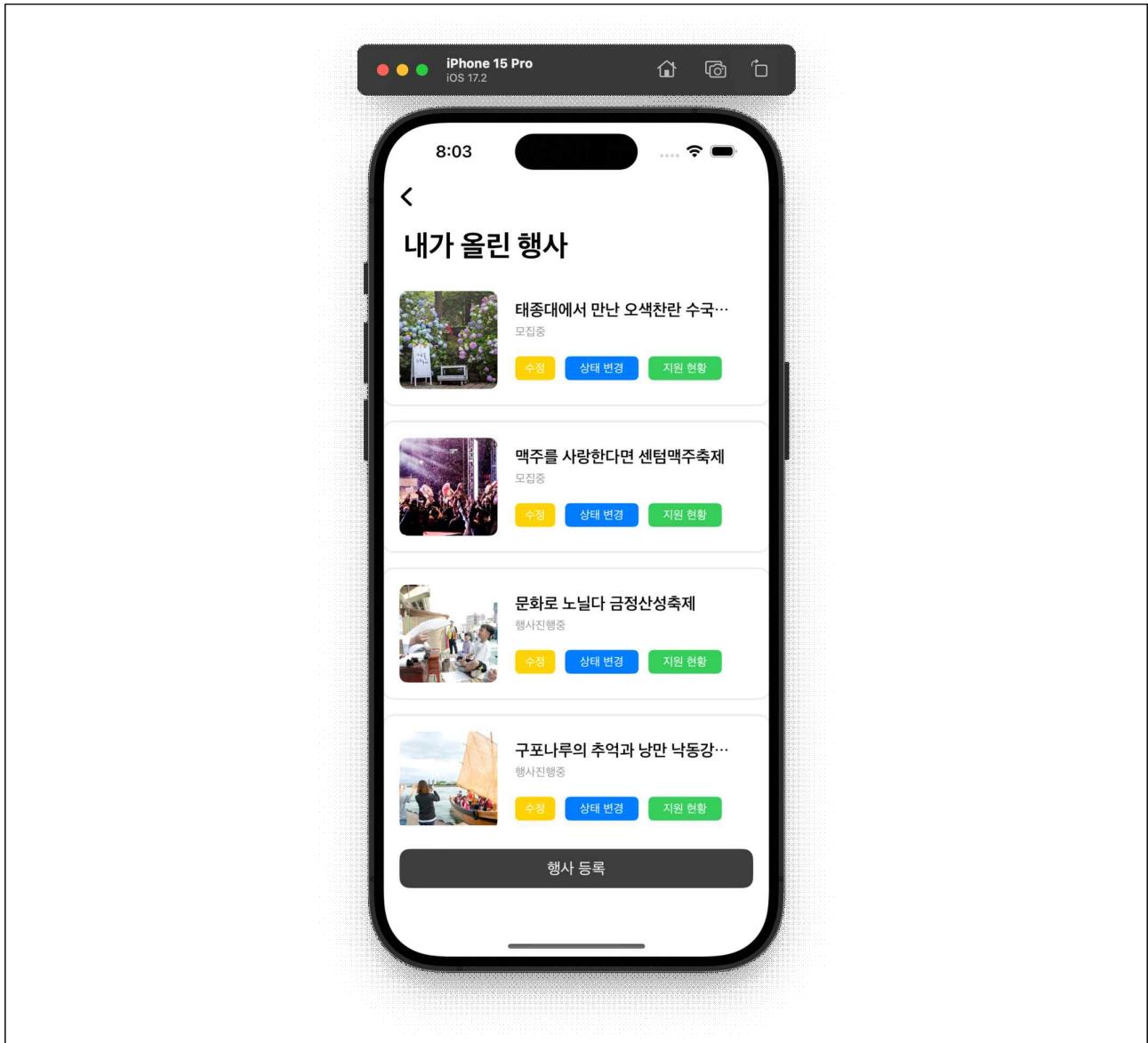
```

```
"managerName")
    }

    if let status = event.status {
        multipartFormData.append(Data(status.utf8), withName: "status")
    }

    if let imageData = image?.jpegData(compressionQuality: 0.8) {
        multipartFormData.append(imageData, withName: "image", fileName:
"event.jpg", mimeType: "image/jpeg")
    }
},
to: "https://api.example.com/events",
headers: headers
).responseDecodable(of: MyEventResponse.self) { response in
    switch response.result {
    case .success(let event):
        completion(event, nil)
    case .failure(let error):
        completion(nil, error)
    }
}
}

}
```



### 17) (푸드트럭) 행사 참여 신청

## (1) 분석 및 설계

푸드트럭 매니저만 사용 가능한 기능이다. 신청 가능한 행사에 한해 본인이 소유한 푸드트럭 중 하나를 선택하고 신청 내용을 작성하여 신청서를 등록하는 기능을 제공한다.

## (2) 구현

```
Server
Event ApplicationController.java
...
@.GetMapping("/{id}")
public String getEventDetail(@PathVariable("id")Long id,Model model,Principal principal){
    model.addAttribute("event",eventService.getEvent(id));
    model.addAttribute("user",userService.findUserByEmail(principal.getName()));
    return "event/event-apply";
}

@PostMapping("/create")
public String registerFoodTruckPost(
    Principal principal,
    EventApplicationRequest eventApplicationRequest
){
    eventApplicationService.createEventApplication(eventApplicationRequest,principal.getName());
    return "redirect:/event";
}
...

Event ApplicationRestController.java
...
@PostMapping("/application")
@PreAuthorize("hasRole('ROLE_FOOD_TRUCK_MANAGER')")
@Operation(summary ="이벤트 신청",description ="푸드트럭 매니저가 이벤트에 신청하는
API<br>푸드트럭 매니저만 사용 가능합니다")
public ResponseEntity<EventApplication> applyEvent(
    @RequestBody EventApplicationRequest eventApplicationRequest,
    Principal principal){
    ResponseEntity.ok(eventApplicationService.createEventApplication(eventApplicationRequest,principal.getName()));
}
...

Event ApplicationRequest.java
public record EventApplicationRequest(
    @Schema(description ="이벤트 ID",example ="1")
    Long eventId,
    @Schema(description ="푸드트럭 ID",example ="2")
    Long foodTruckId,
    @Schema(description ="신청 메시지",example ="죠희 타코야기 잘 만들어요")
    String comment){
}

Event ApplicationService.java
...
/***
 * 지원서 신규 작성 메소드
 *
 * @param eventApplicationRequest 이벤트 요청 품
 * @param email                  신청자(푸드트럭 매니저) 이메일
 */
```

```

    * @return 생성된 신규 지원서
    */
    public EventApplication createEventApplication(EventApplicationRequest
eventApplicationRequest, String email){
        EventApplication eventApplicationSaved
= eventApplicationMapper.toEventApplication(eventApplicationRequest);

        // 유저 정보 조회
        User user = userRepository.findByEmail(email).orElseThrow(() -> new
CustomException(CustomErrorCode.USER_ID_NOT_MATCH));

        if (user.getRoles().contains(UserRole.ADMIN)){
            throw new CustomException(CustomErrorCode.USER_ID_NOT_MATCH);
        }

        Event event
= eventRepository.findById(eventApplicationRequest.eventId()).orElseThrow(() -> new
CustomException(CustomErrorCode.EVENT_NOT_FOUND));

        eventApplicationSaved.setFoodTruckManager(user);
        eventApplicationSaved.setEvent(event);
        eventApplicationSaved.setStatus(ApplicationStatus.PENDING);

        return eventApplicationRepository.save(eventApplicationSaved);
    }
...

```

#### EventApplication.java

```

public class EventApplication extends BaseEntity {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    private Event event;

    private Long foodTruckId;

    @ManyToOne
    private User foodTruckManager;

    @Enumerated(EnumType.STRING)
    private ApplicationStatus status;

    private String comment;
}

```

#### 코드 설명

푸드트럭 신청 요청을 받으면 컨트롤러의 내 신청서 작성 메서드와 매핑되어 EventApplicationService의 createEventApplication 메서드가 호출되며 데이터베이스에 이벤트 신청 정보 생성을 요청한다.

Web
event-apply.html
...
<div class="container mt-5">     <h1 class="text-center">푸드트럭 신청</h1>     <form method="post" th:action="@{/apply/create}">         <input name="eventId" th:value="\${event.id}" type="hidden"/>         <div class="checkbox-group">

```

<h3>신청할 푸드트럭</h3>
<!-- 푸드트럭 콤보박스 -->
<select class="form-select" name="foodTruckId">
    <option value="">푸드트럭을 선택하세요</option>
    <option th:each="foodTruck : ${user.foodTrucks}" th:text="${foodTruck.name}"
        th:value="${foodTruck.id}"></option>
</select>
</div>

<div class="submit-form">
    <h3>신청서 작성</h3>
    <div class="form-group">
        <label for="comment">신청 내용</label>
        <textarea class="form-control" id="comment" name="comment"
            rows="3"></textarea>
    </div>
</div>

<button class="btn btn-primary" type="submit">신청하기</button>
<button class="btn
btn-secondary" onclick="location.href='/event'" type="button">취소하기</button>
</form>
</div>
...

```

#### 코드 설명

이벤트 신청 정보를 입력받고 form 태그를 이용해 입력받은 정보를 서버에 전송한다.

#### 실행 결과

	
---	--

## 18) (행사 매니저) 푸드트럭 신청 목록 조회

### (1) 분석 및 설계

행사 매니저만 사용 가능한 기능이다. 본인이 등록한 이벤트에 한해 신청서들을 조회할 수 있는 기능을 제공한다.

### (2) 구현

Server
EventApplication.java
...
<pre>...     @GetMapping("/list/{id}")     public String getApplyList(@PathVariable("id")Long id,Model model,Principal principal){         User user =userService.findUserByEmail(principal.getName());         List&lt;EventApplication&gt;applications =eventApplicationService.getEventApplications(id);          model.addAttribute("user",user);         model.addAttribute("applications",applications);         model.addAttribute("foodTrucks",applications.stream()             .map(app -&gt;foodTruckService.getFoodTruck(app.getFoodTruckId()))             .collect(Collectors.toList()));          return "event/my_apply_list";     } ...</pre>
EventApplicationRestController.java
...
<pre>...     @GetMapping("/application/event-manager")     @PreAuthorize("hasRole('ROLE_EVENT_MANAGER')")     @Operation(summary ="내 이벤트에 대한 신청 목록 조회",description ="내 이벤트에 대한 신청 목록을 조회합니다.&lt;br&gt;이벤트 관리자만 사용 가능합니다")     public List&lt;EventApplication&gt;getMyEventApplicationsForEventManager(         @Parameter(description ="이벤트 UID")@RequestParam(required =false)Long eventId,         Principal principal){         return eventApplicationService.getMyEventApplicationsForEventManager(eventId,principal.getName());     } ...</pre>
EventApplicationService.java
...
<pre>...     /**      * 이벤트에 대한 모든 지원서 조회      *      * @param eventId 조회 하려는 이벤트 ID      * @return 이벤트에 대한 모든 지원서 목록      */     List&lt;EventApplication&gt;getEventApplications(Long eventId){         return eventApplicationRepository.findAllByEventId(eventId);     }      /**      * 내 이벤트에 대한 신청 목록 조회      * 이벤트 관리자만 사용 가능합니다      *      * @param eventId 이벤트 UID      *          이벤트 관리자만 사용 가능합니다      * @param email 현재 사용자 정보      */     public List&lt;EventApplication&gt;getMyEventApplicationsForEventManager(Long eventId,String email){</pre>

```

        User user =userRepository.findByEmail(email).orElseThrow(()->new
CustomException(CustomErrorCode.USER_ID_NOT_MATCH));
        if (eventId ==null){
            return eventApplicationRepository.findAllByEventManager(user);
        }
        return eventApplicationRepository.findAllByEventManagerAndEventId(user,eventId);
    }
...

```

#### 코드 설명

푸드트럭 신청서 조회 요청을 받으면 컨트롤러의 내 신청서 조회 메서드와 맵핑되어 EventApplicationService 의 이벤트 리스트를 받아오는 메서드가 호출되며 데이터베이스에 이벤트 신청 정보 조회를 요청한다.

Web
my_apply_list.html
<pre> ... &lt;div class="container mt-5"&gt;     &lt;!-- 요청 대기 중인 신청서 --&gt;     &lt;h1 class="text-center"&gt;승인 대기 요청&lt;/h1&gt;     &lt;div class="row"&gt;         &lt;div th:each="foodTruck,iterStat :\${foodTrucks}" th:if="\${applications[iterStat.index].status.name()=='PENDING'}"&gt;             &lt;div class="card"&gt;                 &lt;img alt="Food Truck Image" class="card-img-top" th:src="\${foodTruck.imageUrl}"&gt;                     &lt;div class="card-body"&gt;                         &lt;h5 class="card-title" th:text="\${foodTruck.name}"&gt;&lt;/h5&gt;                         &lt;p class="card-text" th:text="\${foodTruck.description}"&gt;&lt;/p&gt;                         &lt;p class="card-text" th:text="\${foodTruck.foodType.name()}"&gt;&lt;/p&gt;                         &lt;p class="card-text" th:text="\${foodTruck.managerName()}"&gt;&lt;/p&gt;                         &lt;p class="card-text" th:text="\${applications[iterStat.index].comment}"&gt;신청서 내용&lt;/p&gt;                         &lt;a class="btn btn-success" th:href="/apply/accept/\${applications[iterStat.index].id}?eventId=' + \${id}"&gt;수락&lt;/a&gt;                         &lt;a class="btn btn-danger" th:href="/apply/reject/\${applications[iterStat.index].id}?eventId=' + \${id}"&gt;거절&lt;/a&gt;                     &lt;th:href="/food-truck/\${foodTruck.id}"&gt;자세히 보기&lt;/a&gt;                 &lt;/div&gt;             &lt;/div&gt;         &lt;/div&gt;     &lt;!-- 승인된 요청 --&gt;     &lt;h1 class="text-center"&gt;승인된 요청&lt;/h1&gt;     &lt;div class="row"&gt;         &lt;div th:each="foodTruck,iterStat :\${foodTrucks}" th:if="\${applications[iterStat.index].status.name()=='APPROVED'}"&gt;             &lt;div class="card"&gt;                 &lt;img alt="Food Truck Image" class="card-img-top" th:src="\${foodTruck.imageUrl}"&gt;                     &lt;div class="card-body"&gt;                         &lt;h5 class="card-title" th:text="\${foodTruck.name}"&gt;&lt;/h5&gt;                         &lt;p class="card-text" th:text="\${foodTruck.description}"&gt;&lt;/p&gt;                         &lt;p class="card-text" th:text="\${foodTruck.foodType.name()}"&gt;&lt;/p&gt;                         &lt;p class="card-text" th:text="\${foodTruck.managerName()}"&gt;&lt;/p&gt;                         &lt;p class="card-text" th:text="\${applications[iterStat.index].comment}"&gt;신청서 내용&lt;/p&gt;                         &lt;a class="btn btn-primary" th:href="#"&gt;수락&lt;/a&gt;                     &lt;/div&gt;             &lt;/div&gt;         &lt;/div&gt;     &lt;/div&gt; </pre>

```
 자세히 보기 |
    </div>
    </div>
    </div>
</div>
</div>
...

```

### 코드 설명

푸드트럭 신청서 목록을 받아와 신청서의 상태정보를 확인하고 상태가 승인 대기 중인 신청서와 승인된 신청서 두 개를 구별하여 보여준다.

### 실행 결과

실행 결과	
 <p>승인 대기 요청</p> <p>CafeA(신청자) 신고자: 품종등록증 신청 등록번호: 12345 등록일자: 2023-01-01 승인 대기 요청</p>	 <p>승인된 요청</p> <p>CafeA(신청자) 신고자: 품종등록증 신청 등록번호: 12345 등록일자: 2023-01-01 승인된 요청</p>

## 19) (행사 매니저) 푸드트럭 신청 허가/거부

### (1) 분석 및 설계

행사 매니저만 사용 가능한 기능이다. 본인이 등록한 이벤트에 등록된 신청서들을 확인 후 허가, 거부를 하여 원하는 푸드트럭을 고를 수 있는 기능을 제공한다.

### (2) 구현

Server
Event ApplicationController.java
... @GetMapping("/accept/{id}") public String getApplicationAccept(@PathVariable("id")Long id,@RequestParam("eventId")Long eventId,Model model,Principal principal){ eventApplicationService.approveEventApplication(eventId,id,ApplicationStatus.APPROVED); return "event/my_apply_list"; }  @GetMapping("/reject/{id}") public String getApplicationReject(@PathVariable("id")Long id,@RequestParam("eventId")Long eventId,Model model,Principal principal){ eventApplicationService.approveEventApplication(eventId,id,ApplicationStatus.REJECTED); return "event/my_apply_list"; } ...
Event ApplicationRestController.java
... @PostMapping("/event/{eventId}/applications/{applicationId}") @PreAuthorize("hasRole('ROLE_EVENT_MANAGER')") @Operation(summary ="특정 이벤트에 대한 푸드트럭 신청 승인",description ="특정 이벤트에 대한 푸드트럭 신청을 승인합니다.  <b>이벤트 관리자</b>만 승인 가능합니다.") public ResponseEntity<EventApplication> approveEventApplication( @Parameter(description ="이벤트 UID")@PathVariable(name ="eventId")Long eventId, @Parameter(description ="신청 UID")@PathVariable(name ="applicationId")Long applicationId, @Parameter(description ="승인 여부")@RequestParam ApplicationStatus applicationStatus){  return ResponseEntity.ok(eventApplicationService.approveEventApplication(eventId,applicationId,applicationStatus)); } ...
Event ApplicationService.java
... /** * 특정 이벤트에 대한 푸드트럭 신청 승인 / 거절 * 이벤트 관리자만 승인 가능합니다 * * @param eventId         이벤트 UID *                         이벤트 관리자만 승인 가능합니다 * @param applicationId 신청 UID *                         이벤트 관리자만 승인 가능합니다 * @return 승인 결과 */ public EventApplication approveEventApplication(Long eventId,Long applicationId,ApplicationStatus applicationStatus){ EventApplication eventApplication =eventApplicationRepository.findById(applicationId).orElseThrow(()->new IllegalStateException("지원서가 존재하지 않습니다")); }

```

if (!eventApplication.getEvent().getId().equals(eventId)){
    throw new IllegalArgumentException("이벤트 ID가 일치하지 않습니다");
}
eventApplication.setStatus(applicationStatus);
return eventApplicationRepository.save(eventApplication);
}
...

```

#### 코드 설명

푸드트럭 신청서 허가, 거부 요청을 받으면 컨트롤러의 내 신청서 허가, 거부 메서드와 매핑되어 EventApplicationService 의 approveEventApplication 메서드가 호출되며 데이터베이스에 신청서 상태 변경을 요청한다.

#### Web

my\_apply\_list.html

```

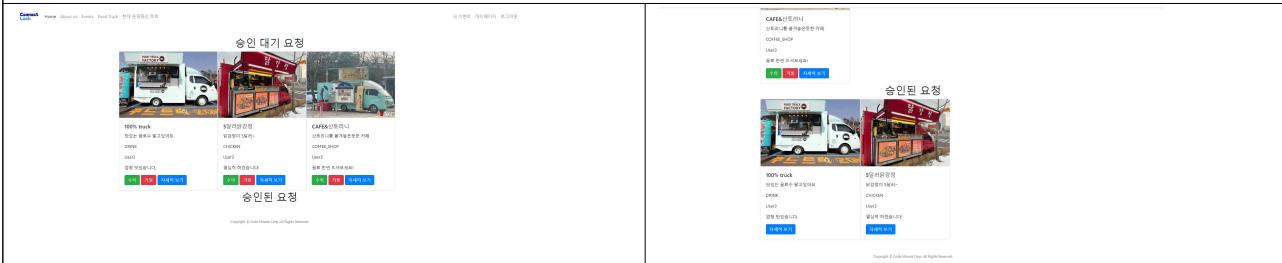
...
<a class="btn btn-success" th:href="/apply/accept/" + ${applications[iterStat.index].id} +
'?eventId=' + ${id}">수락</a>
<a class="btn btn-danger" th:href="/apply/reject/" + ${applications[iterStat.index].id} +
'?eventId=' + ${id}">거절</a>
...

```

#### 코드 설명

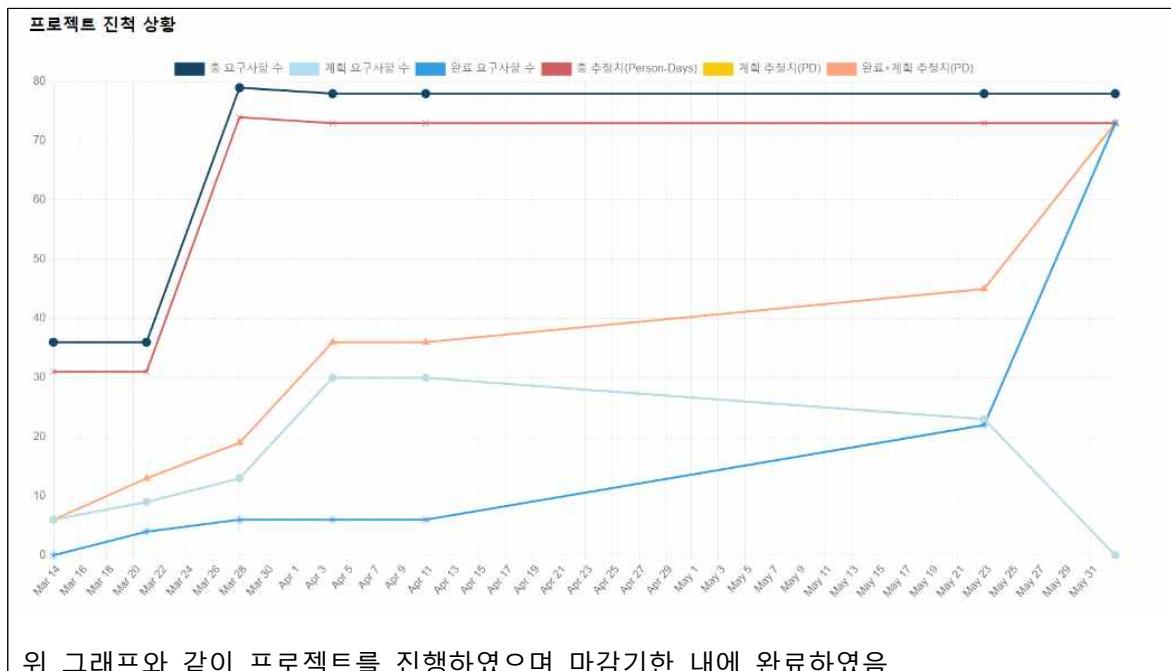
푸드트럭의 이벤트 신청 정보를 저장할 때 필요한 event의 id 속성 값을 쿼리로 주고 신청서의 id 값을 주소로 주어 서버에 전송한다.

#### 실행 결과



### 3. 프로젝트 수행 결과

#### 1) 프로젝트 완성도



위 그래프와 같이 프로젝트를 진행하였으며 마감기한 내에 완료하였습니다.

#### 2) 일정 계획 평가

프로젝트 진행 중 예기치 못한 기술적 문제와 요구사항 변경으로 일정에 차질이 생겼음. 문제 해결을 위해 시간을 더 투자하고 우선순위를 재조정하여 작업량이 증가했으나, 팀의 노력으로 주요 기능과 최종 일정을 준수하였음.

#### 3) 소스 코드 버전 제어 도구 및 협업 도구 사용

**GitHub:**  
분산 버전 제어 시스템을 사용하여 각 개발자가 로컬에서 전체 코드를 관리할 수 있으며, 이는 병렬 작업을 용이하게 하고, 병합 과정에서의 충돌을 최소화하였다. 또한 코드 리뷰, 이슈 트래킹, 풀 리퀘스트 등의 기능을 제공하여 팀원 간의 협업을 효율적으로 진행하였다.

**Notion:**  
프로젝트 계획, 문서화, 작업 관리 등을 하나의 플랫폼에서 통합적으로 관리하였으며 다양한 템플릿과 구성 요소를 사용하여 필요에 맞게 팀원들이 동시에 문서를 편집하고, 실시간으로 업데이트 상황을 확인할 수 있어 협업을 원활하게 진행하였음.

#### 4) 설계 구성요소

목표설정	푸드트럭과 행사의 필요한 정보를 빠르고 정확하게 교환하여 최적의 매칭을 이룰 수 있도록 한다
분석	푸드트럭과 행사의 세부 정보를 투명하게 제공하여 정보 비대칭 문제 해결하고, 사용자의 요구사항과 조건을 기반으로 자동화된 프로그램 개발
제작	<p>Server 및 Web :</p> <p>IntelliJ Ultimate 환경에서 외부라이브러리 (MapStruct, Springdoc-Webui)와 Spring Boot를 사용하였으며 더미데이터는 공공API를 활용하여 제작하였음.</p> <p>Android App :</p> <p>Android Studio 환경에서 라이브러리(Retrofit, OkHttp, Dagger Hilt)와 내부 라이브러리를 활용하여 제작하였음.</p> <p>IOS App :</p> <p>Xcode 환경에서 외부 라이브러리(Alamofire, KingFisher)와 내부 라이브러리를 활용하여 제작하였음.</p>
시험	웹 페이지 Url : <a href="https://connect-luck.store/">https://connect-luck.store/</a>
평가	<p><b>목표 달성:</b> 초기 목표와 주요 기능을 대부분 달성함.</p> <p><b>문제 해결:</b> 예기치 못한 기술적 문제와 요구사항 변경에도 불구하고, 시간을 투입하고 우선순위 재조정을 통해 문제를 해결함.</p> <p><b>일정 관리:</b> 일정 지연과 작업량 증가에도 불구하고, 팀의 노력으로 최종 일정을 준수함.</p> <p><b>팀 역량:</b> 팀원들의 혁신적인 노력과 효율적인 커뮤니케이션으로 성공적으로 프로젝트를 마무리함.</p>
기타	-

## 5) 설계 제한요소

원가	본 프로젝트를 진행함에 있어 5인 개발 프로젝트임을 고려하여 프로젝트 범위와 일감을 조절하였다.		
안정성	<p>Junit5와 swagger를 사용하여 Rest API 관련 요청 및 반환이 잘 되는지 확인하며 작업을 진행하였음.</p> <p>Docker 환경 배포 작업을 하여 서버를 보다 안정적으로 유지 및 보수가 가능토록 하였음.</p> <p>JWT 보안 처리를 적용하여 Rest API는 Header, Web은 Cookie에 적용하여 보안 유지 및 안정성을 향상 시켰음.</p>		
신뢰성	<p>Server, Web :</p> <p>작업 장치 Windows 10 Pro Intel i5-7500 DDR4 16GB</p> <p>작업 환경 IntelliJ Ultimate Spring Boot 3.2.3 Swagger (Rest API Docs)</p>	<p>Android App :</p> <p>작업 장치 LG 그램 17인치 윈도우 11 Home Intel i7 13세대 DDR4 16G SSD 512G</p> <p>작업 환경 Android Studio 2023.2.1</p>	<p>IOS App :</p> <p>작업 장치 MacBook Pro (16형, 2021년) M1 (10 Core, 16 GPU) 16GB 통합 메모리</p> <p>작업 환경 Xcode 15.2</p>
미학	<p>Server :</p> <p>MVC패턴과 도메인패키지를 적용하여 코드의 가독성을 증가시켰으며 효율적인 관리를 가능하도록 하였음.</p> <p>Web :</p> <p>BootStrap을 사용하여 웹 페이지 제작에 있어 좀 더 편리하도록 활용하였음</p> <p>Android App :</p> <p>RecyclerView와 어댑터, ViewBinding을 활용하여 데이터를 동적으로 View에 적재할 수 있고, ScrollView를 사용하여 화면보다 긴 정보량을 안정적으로 소화할 수 있다.</p> <p>IOS App :</p>		
기타	AndroidStudio, Xcode, intellij IDE를 사용하여 개발 및 배포하였음.		

### III. 결론

#### 1. 프로젝트 달성도

[표 3] 설계 목표의 중요도 및 달성도

ID	기능 요구사항	추정치(일)	달성도(%)	관련 주요 산출물
SFR-100	회원관리기능 ( Web, Android, IOS )	16	100%	main/java/ac/kr/deu/connect/luck/us er
SFR-200	푸드트럭 기능 ( Web, Android )	20	100%	main/java/ac/kr/deu/connect/luck/fo od_truck
SFR-300	후기 기능 ( Web )	10	100%	main/java/ac/kr/deu/connect/luck/co mmon,configuration
SFR-400	행사 관리자 ( Web, IOS )	10	100%	main/java/ac/kr/deu/connect/luck/ev ent, event_application
SFR-500	매칭 시스템 ( Web, Android, IOS )	17	100%	
합계		73	100%	-

## 프로젝트 구조 (Server Web)

```
connect-luck/
└ src
  └ main
    └ java
      └ ac
        └ kr
          └ deu
            └ connect
              └ luck
                └ auth # 인증 관련 패키지
                └ common # 공통 패키지
                └ configuration # 설정 관련 패키지
                └ event # 이벤트 관련 패키지
                └ food_truck # 푸드 트럭 관련 패키지
                └ user # 사용자 관련 패키지
                └ Application.java
    └ resources # 리소스 파일
      └ static # 정적 파일
        └ style # 스타일 파일
          └ css
        └ favicon.ico
      └ templates
        └ auth # 인증 관련 페이지
        └ event # 이벤트 관련 페이지
        └ food_truck # 푸드 트럭 관련 페이지
        └ fragments # 공통 페이지
        └ layout # 레이아웃 페이지
        └ user # 사용자 관련 페이지
```

## 2. 프로젝트 결과 논의

본 프로젝트에 관해 초기 개발 목표를 달성 하였으며, 구성 요소를 모두 구현하였음.

예외처리기능과 추천 기능에 대한 가능성을 작업하는 도중에 알게되어 기능을

추가하지 못한점이 아쉬움

- 추천 알고리즘을 적용하여 추천하는 푸드트럭의 종류를 보여주는 기능
- 푸드트럭 매니저가 본인 신청서 관리 기능 추가
- iOS 소셜 로그인 기능 도입
- 채팅 기능 도입, 실시간 위치 및 결제 기능 도입

## 3. 프로젝트 결과의 활용방안

- 지역 사회의 경제 활성화: 행사와 지역 푸드트럭 사업자의 직접 연결을 통해 지역 경제를 촉진하고, 지역 내 소비를 장려한다.

- 소규모 사업자의 성장 지원: 중개인 없이 직접 거래가 가능해짐으로써, 소규모 푸드트럭 사업자의 비용 절감과 수익 증대를 돋는다.
- 소상공인 지원: 중소규모 푸드트럭 사업자에게 공정한 기회를 제공함으로써 소상공인의 경제 활동을 지원하고, 사업 확장의 기회를 마련한다.
- 투명성 강화: 모든 거래를 플랫폼을 통해 기록함으로써, 행사 관계자와 푸드트럭 사업자 간의 투명성을 보장하고 신뢰를 구축한다.
- 전문 교육 프로그램 제공: 푸드트럭 운영자를 위한 교육 프로그램을 제공하여, 비즈니스 운영, 위생 관리, 고객 서비스 등의 역량을 강화한다.

#### 4. 프로젝트 결과의 기대 성과

기술적 측면	ConnectLuck은 푸드트럭 사업자와 행사 관계자 간의 효율적인 매칭을 가능하게 함으로써, 시간과 비용을 절약하고 매칭 과정을 간소화한다. 이를 통해 푸드트럭 사업자는 더 많은 행사에 참여할 기회를 얻을 수 있고, 행사 관계자는 원하는 조건에 부합하는 푸드트럭을 쉽게 발굴할 수 있다.
경제적 측면	본 프로젝트의 결과를 바탕으로 푸드트럭 산업의 활성화가 기대된다. 푸드트럭 매칭 플랫폼을 통해 행사 및 이벤트에서 푸드트럭을 운영하는 데 필요한 정보를 손쉽게 얻을 수 있게 되면, 이를 통해 푸드트럭 사업의 확대 및 경제적 효과를 기대할 수 있다.
사회적 측면	본 프로젝트의 결과로 인해 지역사회와 사회 전반에 긍정적인 영향이 있을 것으로 전망된다. 푸드트럭 사업자들의 활동을 지원하고 행사 및 이벤트의 다양성을 증대시킴으로써 지역 경제의 활성화와 지역 사회의 활동성이 증가할 것으로 예상된다. 또한, 푸드트럭을 통한 식문화의 다양성과 확산은 지역 사회의 문화적 발전에도 긍정적인 영향을 미칠 것이다. 이러한 사회적 영향은 지속 가능한 발전을 통해 지역사회의 발전에 도움을 줄 것으로 기대된다.

#### IV. 참고문헌

- [1] 국민67.2% “푸드트럭” 자주 이용할 의향 있다”, RealMeter, 2015-11-03 수정, 2024-05-30 접속, <https://www.realmeter.net/%ED%91%B8%EB%93%9C%ED%8A%B8%EB%9F%AD-%EC%9E%90%EC%A3%BC-%EC%9D%B4%EC%9A%A9%ED%95%A0-%EC%9D%98%ED%96%A5%EC%9E%88%EB%8B%A4-67-2/>
- [2] [와이라노]그많던 푸드트럭 어디로 갔을까, 국제 신문, 2022-08-15 수정, 2024-05-31 접속, <https://www.kookje.co.kr/news2011/asp/newsbody.asp?code=0300&key=20220815.99099003745>

[부록]