결과 보고서

오운완: 오늘 운동 완료

1. 서론

- 개발 배경

현대 사회에서 운동과 건강한 생활 습관은 점점 더 중요해지고 있다. 많은 사람들이 일상에서 운동 목표를 세우지만, 이를 지속하고 동기를 유지하는 데 어려움을 겪고 있다. 이러한 문제를 해결하기 위해, 사용자들이 서로의 운동 성과를 공유하고 독려할 수 있는 플랫폼의 필요성이 대두되었다.

그래서 '오운완' ("오늘운동완료")이라는 인증 캠패인을 활용하는 애플리케이션을 개발해보았다. 이 애플리케이션은 사용자들이 일일 운동 목표를 달성하고, 그 과정을 인증하며, 이를 통해 서로를 독려하고 격려하는 커뮤니티를 형성할 수 있도록 돕는다.

- 작품에 대한 설명

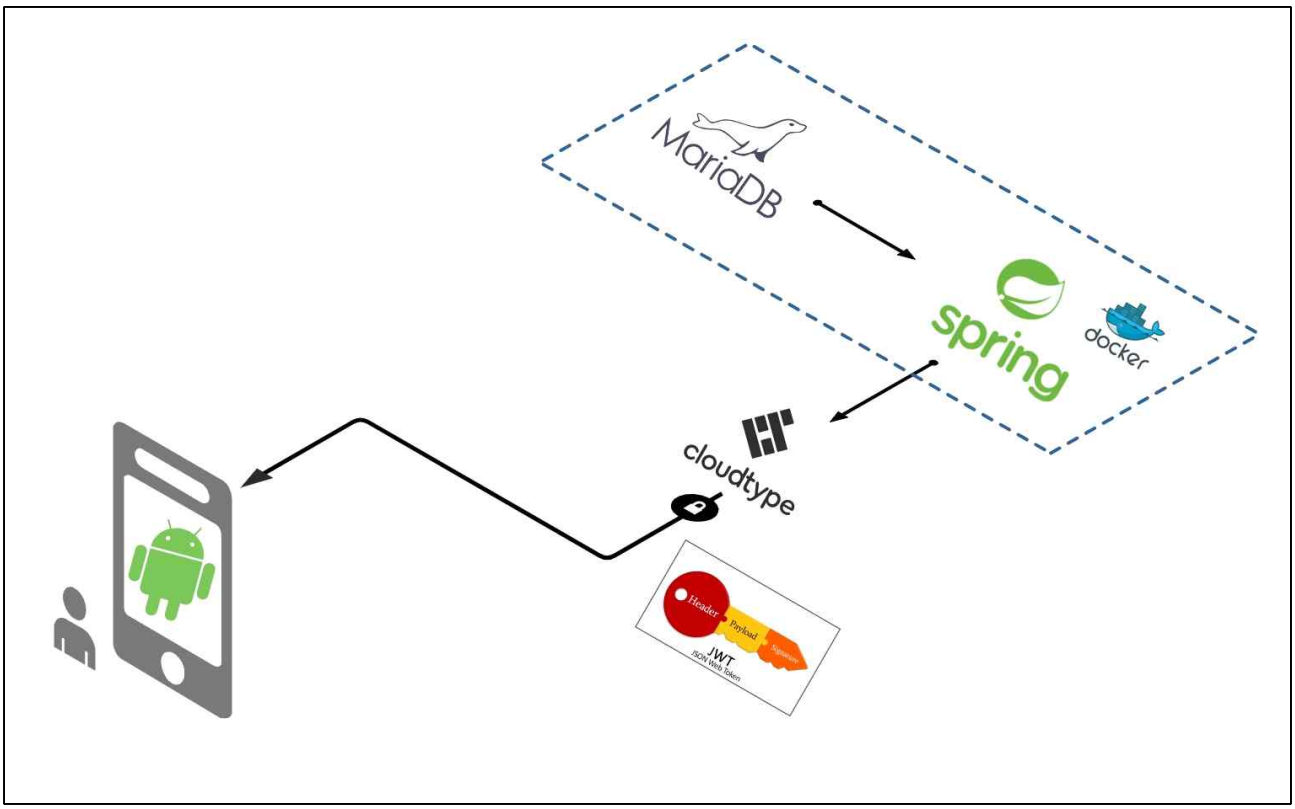
'오운완' 앱은 사용자들이 일일 운동 목표를 달성하고 이를 커뮤니티와 공유할 수 있는 플랫폼이다. 이 앱은 운동 목표 달성을 인증하는 기능과 함께, 사용자들이 서로의 성과를 칭찬하고 격려할 수 있는 커뮤니티 기반의 상호작용 공간을 제공한다. 이를 통해 사용자들은 서로를 독려하고, 운동에 대한 동기를 증가시키며, 건강한 생활 습관을 형성하는 데 도움을 받을 수 있다.

- 작품명 결정 이유

오운완'이라는 이름은 '오늘의 운동을 완료'라는 의미를 담고 있다. 이는 앱의 주요 기능인 일일 운동 목표의 달성과 인증을 강조하며, 사용자들이 서로의 성공을 축하하고 독려하는 커뮤니티의 중요성을 반영한다. 이름을 통해 사용자들에게 앱의 목적과 가치를 명확하게 전달하고자 하였다.

2. 작품 개요

2.1 전체 구성도



- 클라이언트 애플리케이션

오운완 애플리케이션은 안드로이드 기반의 모바일 애플리케이션으로 제공된다. 이를 통해 사용자들은 손쉽게 운동 목표를 설정, 인증하고, 커뮤니티와의 상호작용을 할 수 있다.

- 인증 시스템

애플리케이션의 보안은 JWT 인증 방식을 통해 강화되었다. 이 방식은 사용자의 안전한 인증을 보장하며, 보안성이 높은 사용자 경험을 제공한다.

- 클라우드 호스팅

오운완의 백엔드 서버와 데이터베이스 서버는 '클라우드타입'이라는 클라우드 환경에 배포되어 있다. 이를 통해 안정적이고 효율적인 서비스 운영이 가능한다.

- 백엔드 서버

서버 측 로직은 스프링(Spring) 백엔드 서버에서 처리되며, 이 서버는 도커(Docker)를 이용하여 배포된다. 이는 스프링 애플리케이션의 배포와 관리를 보다 효율적으로 만들어 준다. 스프링은 자바 기반의 웹 애플리케이션 개발에 널리 사용되는 프레임워크로, 강력한 성능과 유연성을 제공한다.

- 데이터베이스 서버

스프링 백엔드 서버는 마리아DB(MariaDB) 데이터베이스 서버와 통신한다. 마리아DB는 MySQL에서 파생된 커뮤니티 주도의 오픈소스 데이터베이스 관리 시스템으로, 높은 성능과 안정성을 제공한다.

2.2 작품 설명

'오운완'은 안드로이드 모바일 애플리케이션으로, 사용자들이 자신의 일일 운동 목표를 달성하고 인증하는 것을 도와준다. 이 앱의 목적은 사용자들이 건강한 운동 습관을 형성하고, 이를 커뮤니티와 공유함으로써 서로를 격려하고 동기를 부여하는 것이다.

- 주요 기능

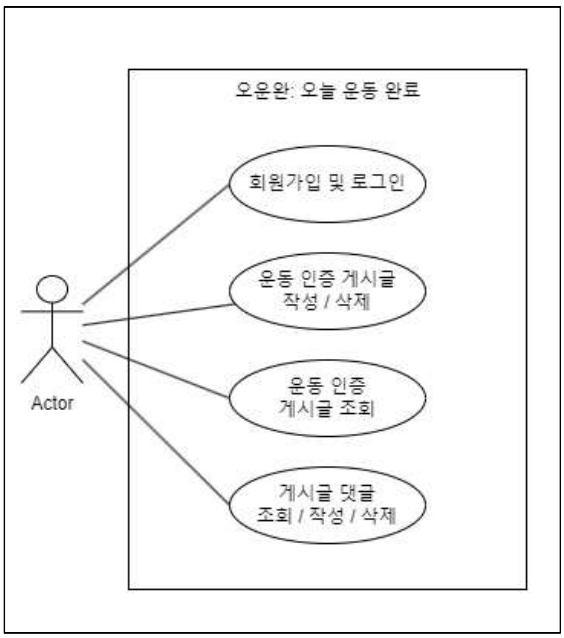
운동 목표 설정과 인증: 사용자는 앱을 통해 자신의 일일 운동 목표를 설정하고, 운동을 완료한 후 이를 앱을 통해 인증할 수 있다.
커뮤니티 상호작용: 사용자들은 자신의 운동 성과를 커뮤니티와 공유하고, 다른 사용자들의 성과를 보며 서로를 격려하고 동기부여를 할 수 있다.
개인화된 피드백: 앱은 사용자의 운동 데이터를 분석하여 개인화된 피드백과 조언을 제공한다.

- 기술적 특징

안드로이드 기반: 모바일 사용자에게 친숙한 안드로이드 플랫폼을 기반으로 하여 넓은 사용자 접근성을 제공한다.
JWT 인증: 사용자의 데이터 보안을 위해 JWT(제이슨 웹 토큰) 인증 방식을 사용한다.
클라우드 호스팅: 서비스의 안정성과 확장성을 위해 '클라우드타입' 환경에 백엔드와 데이터베이스 서버를 배포한다.
스프링 백엔드와 도커 배포: 자바 기반의 스프링 프레임워크를 사용하며, 백엔드 서버는 도커를 이용하여 배포된다. 이는 높은 성능과 유연성을 제공한다.
마리아DB 데이터베이스: 오픈소스이며 MySQL에서 파생된 마리아DB를 데이터베이스 서버로 사용하여 높은 성능과 안정성을 보장한다.

3. 설계

3.1 Use Case



3.1.1 회원가입 및 로그인

시나리오명: 애플리케이션 회원가입
기본 흐름:
1. 사용자는 회원가입 페이지에 접속한다.
2. 사용자는 사용할 계정의 아이디를 입력한후 중복 검사를 한다.
3. 사용자는 비밀번호 및 비밀번호 확인 및 사용자 이름을 입력한다.
4. 사용자는 회원가입 버튼을 누른다.
5. 유효성 검사후, 시스템은 데이터베이스에 사용자 정보를 저장한다.

시나리오명: 애플리케이션 로그인
사전조건: 사용자는 계정을 보유하고 있다.
기본 흐름:
1. 사용자는 로그인 페이지에 접속한다.
2. 사용자는 보유한 계정의 아이디와 비밀번호를 입력한다.
3. 사용자는 로그인 버튼을 클릭한다.
4. 인증 후, 시스템은 휴대폰에 사용자 정보를 저장한다.
5. 로그인 절차를 마친 사용자를 홈 화면으로 리다이렉트 한다.

3.1.2 운동 인증 게시글 조회/삭제

시나리오명: 운동 인증 게시글 조회
사전조건: 사용자는 로그인 상태이다.
기본 흐름:
1. 사용자는 "운동 인증 게시글 화면"에 접속한다.
2. 사용자는 해당 화면에서 원하는 게시글을 확인하고 선택한다.
3. 시스템은 데이터베이스에서 해당 게시글의 정보를 받아온다.
4. 시스템은 받아온 게시글 정보를 사용자에게 표시한다.

시나리오명: 운동 인증 게시글 삭제
사전조건: 사용자는 로그인 상태이다.
기본 흐름:
1. 사용자는 "운동 인증 게시글 화면"에 접속한다.
2. 사용자는 본인이 작성한 게시글을 확인하고 선택한다.
3. 시스템은 데이터베이스에서 해당 게시글의 정보를 받아온다.
4. 시스템은 받아온 게시글 정보를 사용자에게 표시한다.
5. 사용자는 삭제 버튼을 클릭해 본인이 작성한 글을 삭제한다.

3.1.3 운동 인증 게시글 작성

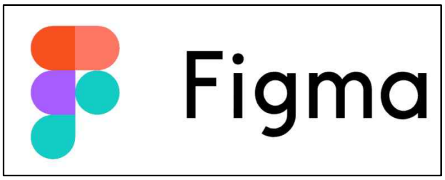
시나리오명: 운동 인증 게시글 작성
사전조건: 사용자는 로그인 상태이다.
기본 흐름:
1. 사용자는 운동 인증 게시판에 접속한다.
2. 시스템은 운동 인증 게시글을 제공한다.
3. 사용자는 '게시글 작성' 버튼을 클릭하여 게시글을 작성한다.
4. 게시글 작성 후, 시스템은 게시글 정보를 데이터베이스에 저장한다

3.1.4 운동 인증 게시글 댓글 조회/작성/삭제

시나리오명: 운동 인증 게시글 댓글 작성/조회/삭제
사전조건: 사용자는 로그인 상태이다.
기본 흐름:
1. 사용자는 운동 인증 게시글에 접속한다.
2. 시스템은 운동 인증 게시글 정보 및 댓글 목록을 제공한다.
3. 사용자는 해당 게시글에 댓글을 입력한다.
4. 사용자는 '댓글 작성' 버튼을 클릭하여 댓글을 작성한다.
5. 댓글 작성 후, 시스템은 게시글 정보를 데이터베이스에 저장한다
6. 사용자는 본인이 작성한 댓글을 포함한 댓글 목록을 확인할 수 있다.
7. 사용자는 본인이 작성한 댓글을 삭제할 수 있다.

3.2 UI Design

본 프로젝트의 UI Design은 Figma라는 디자인 툴을 활용하여 구현되었다.



3.2.1 로그인 및 회원가입 화면

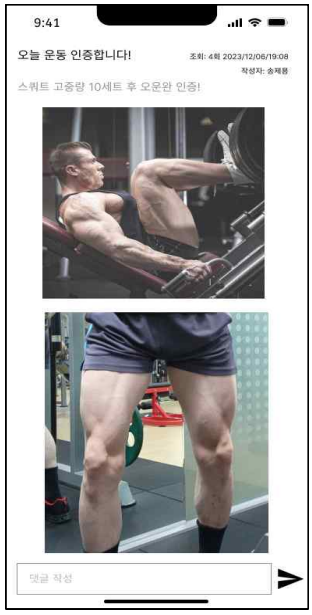
- 로그인 화면

이 화면은 사용자가 앱에 로그인할 수 있도록 디자인된 인터페이스를 보여준다. 상단에는 "로그인"이라는 제목이 있고, 사용자에게 다른 사람들에게 운동을 인증하고 인증받으라는 독려의 메시지가 있다. 이메일과 비밀번호를 입력할 수 있는 두 개의 입력 필드가 있으며, 사용자는 '로그인' 버튼을 통해 접속하거나 '회원가입 하러 가기' 버튼으로 새 계정을 생성할 수 있다.

- 회원가입 화면

이 화면은 새로운 사용자가 앱에 회원가입을 할 수 있는 페이지를 보여준다. "회원가입"이라는 제목 아래에는 이메일, 비밀번호, 비밀번호 확인, 사용자 이름을 입력할 수 있는 네 개의 텍스트 필드가 있다. 사용자는 필요한 정보를 입력한 후 '회원가입' 버튼을 통해 계정을 생성할 수 있고, 이미 계정이 있는 경우 '로그인 하러 가기' 버튼을 통해 로그인 페이지로 이동할 수 있다.

3.2.2/ 3.2.3/ 3.2.4운동 인증 게시글 목록 화면, 작성 화면, 세부 내용 화면



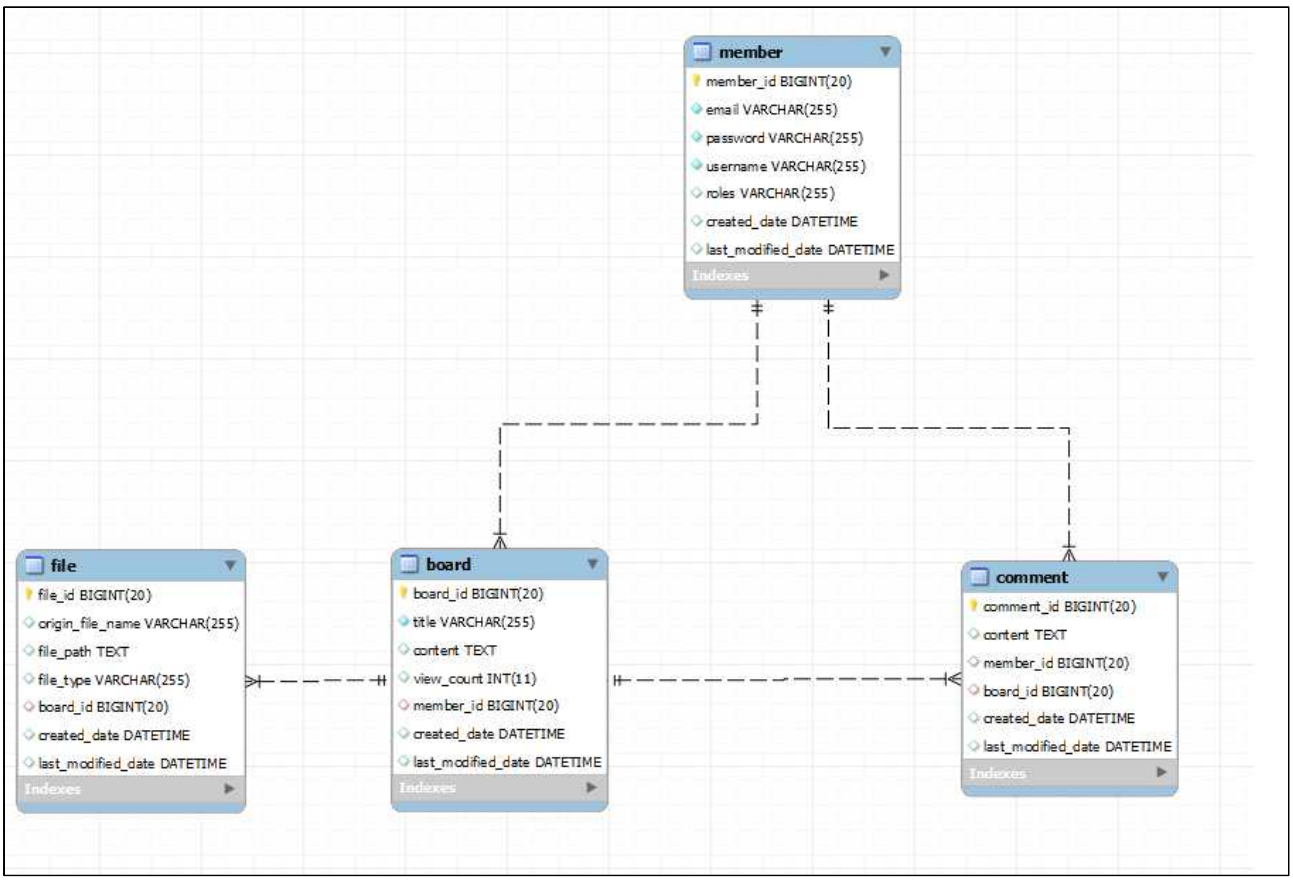
- 운동 인증 게시글 목록 화면
이 화면은 사용자가 다른 사람들이 올린 운동 인증 게시글의 목록을 볼 수 있는 페이지이다. 각 게시글은 제목, 게시 시간, 게시자의 닉네임으로 구성되어 있어 사용자가 누가 언제 어떤 운동 인증을 올렸는지 한눈에 파악할 수 있다. 사용자는 이 페이지에서 관심 있는 게시글을 선택하여 세부 내용을 볼 수 있다.

- 운동 인증 게시글 작성 화면
이 화면은 사용자가 자신의 운동 인증 게시글을 작성할 수 있는 인터페이스를 제공한다. 사용자는 제목과 내용을 입력한 후, '이미지 및 동영상 선택' 버튼을 통해 운동 중인 사진이나 동영상을 첨부할 수 있다. 작성을 완료한 후 '게시글 작성' 버튼을 클릭하여 게시글을 공유할 수 있다.

- 운동 인증 게시글 세부 내용 화면
이 화면은 사용자가 선택한 특정 운동 인증 게시글의 세부 내용을 볼 수 있게 해준다. 게시글에는 운동 중인 사람의 사진과 함께 운동에 대한 설명이나 응원의 메시지가 포함될 수 있다. 이는 커뮤니티 내에서 운동 동기부여를 공유하고, 서로를 격려하는 데 기여하는 기능이다.

3.3 DB Design

3.3.1 Conceptual Design



3.3.2 Logical Design

Member Table

```
CREATE TABLE member (
    member_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    email VARCHAR(255) NOT NULL,
    password VARCHAR(255) NOT NULL,
    username VARCHAR(255) NOT NULL,
    roles VARCHAR(255),
    created_date DATETIME,
    last_modified_date DATETIME
);
```

Board Table

```
CREATE TABLE board (
    board_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    content TEXT,
    view_count INT DEFAULT 0,
    member_id BIGINT,
    created_date DATETIME,
    last_modified_date DATETIME,
    FOREIGN KEY (member_id) REFERENCES member(member_id)
);
```

Comment Table

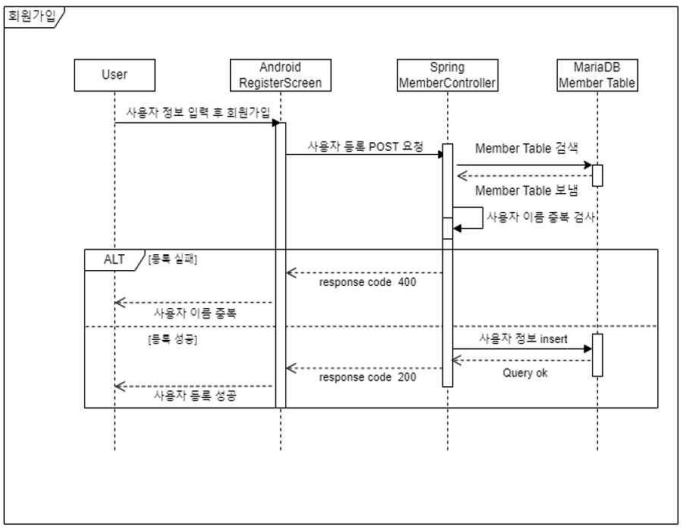
```
CREATE TABLE comment (
    comment_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    content TEXT,
    member_id BIGINT,
    board_id BIGINT,
    created_date DATETIME,
    last_modified_date DATETIME,
    FOREIGN KEY (member_id) REFERENCES member(member_id),
    FOREIGN KEY (board_id) REFERENCES board(board_id)
);
```

File Table

```
CREATE TABLE file (
    file_id BIGINT AUTO_INCREMENT PRIMARY KEY,
    origin_file_name VARCHAR(255),
    file_path TEXT,
    file_type VARCHAR(255),
    board_id BIGINT,
    created_date DATETIME,
    last_modified_date DATETIME,
    FOREIGN KEY (board_id) REFERENCES board(board_id)
);
```

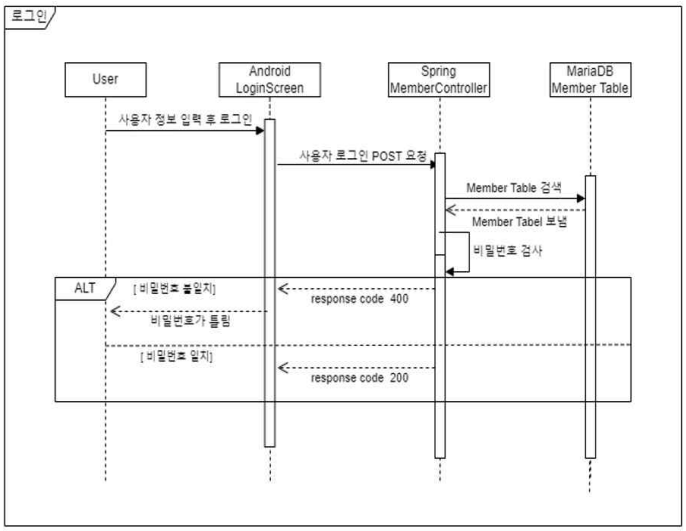
3.4 Use Case별 Sequence Diagram

3.1.1 회원가입



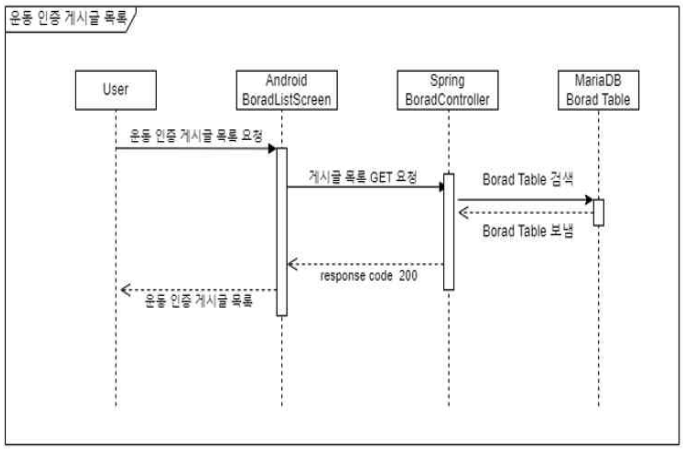
사용자는 안드로이드 앱에서 회원가입을 위해 필요한 개인 정보를 입력하고, '회원가입' 버튼을 누릅니다. 이 데이터는 서버의 Spring MemberController로 POST 요청을 통해 전송되며, 서버는 데이터의 유효성을 검사한다. 만약 입력된 정보에 문제가 없다면, 서버는 회원 정보를 데이터베이스의 Member Table에 저장하고, 성공 메시지와 함께 response code 200을 사용자의 안드로이드 앱으로 반환한다. 그러나 정보에 오류가 있거나 문제가 발생한 경우, response code 400과 함께 오류 메시지를 반환하여 사용자에게 알려준다.

3.1.2 로그인



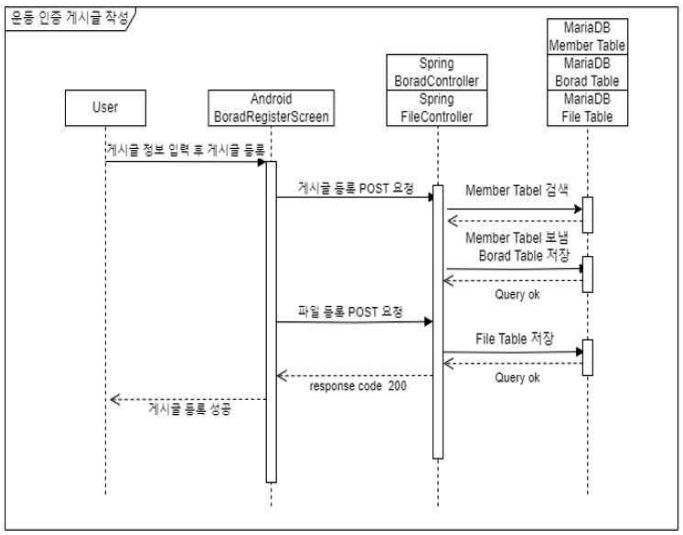
로그인 절차는 사용자가 로그인 화면에서 아이디와 비밀번호를 입력하고 '로그인' 버튼을 클릭하면 시작된다. 이 정보는 Spring MemberController로 POST 요청과 함께 전송된다. 서버는 요청을 받고, 데이터베이스에서 해당 사용자 정보를 조회한다. 로그인 정보가 정확하면 사용자는 response code 200을 받아 로그인에 성공한 것을 확인할 수 있다. 정보가 정확하지 않으면 response code 400을 받아 로그인 실패를 알 수 있다.

3.1.3 운동 인증 게시글 목록 조회



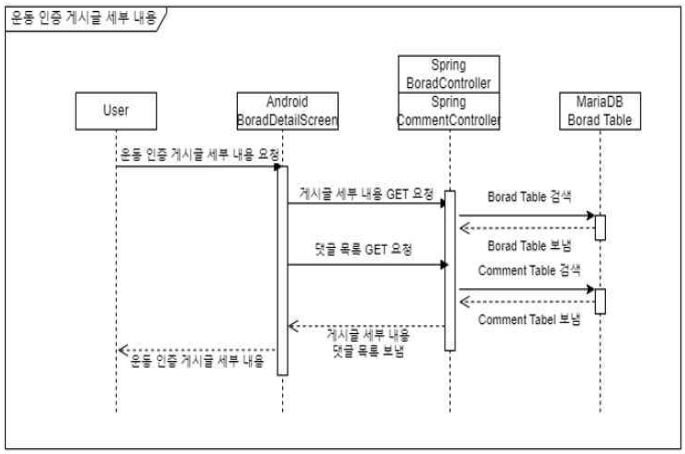
게시글 목록 조회는 사용자가 안드로이드 앱의 게시판 목록 화면에서 '새로고침' 또는 '목록 불러오기'를 선택하면 발생한다. 이 요청은 BoardController를 통해 GET 방식으로 서버로 전송된다. 서버는 요청을 처리하고, 게시글의 목록 데이터를 Board Table에서 가져온 후, 이를 사용자에게 response code 200과 함께 반환한다. 이를 통해 사용자는 최신의 게시글 목록을 볼 수 있다.

3.1.4 운동 인증 게시글 작성



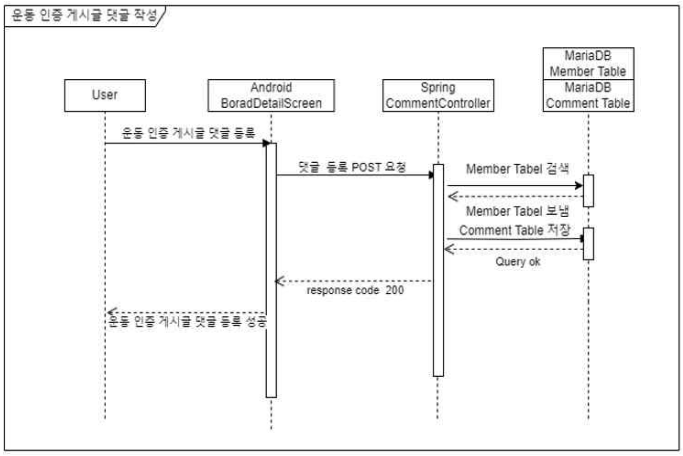
운동 인증 게시글 작성은 사용자가 게시글 작성 화면에서 제목, 내용, 운동 인증 사진 등을 입력하고 '작성 완료' 버튼을 클릭할 때 시작된다. 입력한 게시글 정보는 BoardController에 POST 요청으로 전송되며, 사진과 같은 파일은 FileController를 통해 별도로 처리된다. 데이터베이스의 Board Table과 File Table에 정보가 성공적으로 저장되면, 사용자는 response code 200을 받아 게시글 작성이 완료되었음을 알 수 있다.

3.1.5 운동 인증 게시글 세부 내용 조회



사용자가 특정 게시글을 클릭하여 세부 내용을 보고자 할 때, 안드로이드 앱은 BoardController에 GET 요청을 보내 해당 게시글의 세부 정보를 요청한다. 또한, 게시글에 달린 댓글들을 보기 위해 CommentController에도 GET 요청을 한다. 서버는 Board Table과 Comment Table에서 정보를 가져와 사용자에게 response code 200과 함께 제공한다. 이를 통해 사용자는 게시글의 모든 내용과 댓글을 볼 수 있다.

3.1.6 운동 인증 게시글 댓글 작성



댓글 작성은 사용자가 게시글 세부 화면에서 댓글 입력란에 텍스트를 입력하고 '댓글 달기'를 누를 때 이루어진다. 입력한 댓글은 CommentController로 POST 요청을 통해 전송되며, 서버는 이를 Comment Table에 저장한다. 댓글 저장이 성공적으로 이루어지면, 서버는 response code 200을 반환하여 사용자에게 작업 완료를 알린다. 이로써 사용자는 자신의 댓글이 게시글에 성공적으로 등록되었음을 확인할 수 있다.

4. 구현

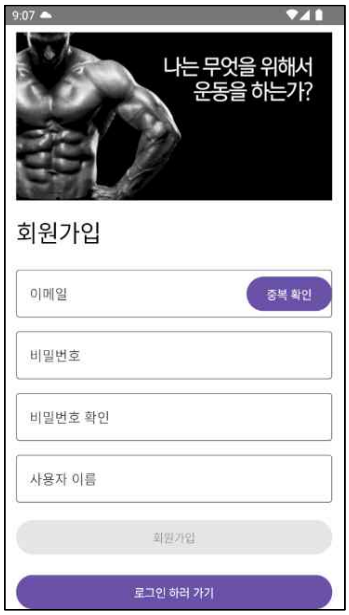
4.1 프른트엔드 (안드로이드)

4.1.1 회원가입 및 로그인

- 회원가입 실행 화면

성공 시나리오

- 사용자는 회원가입 페이지에 접속한다.



- 사용자는 사용할 계정의 아이디를 입력한후 중복 검사를 한다.

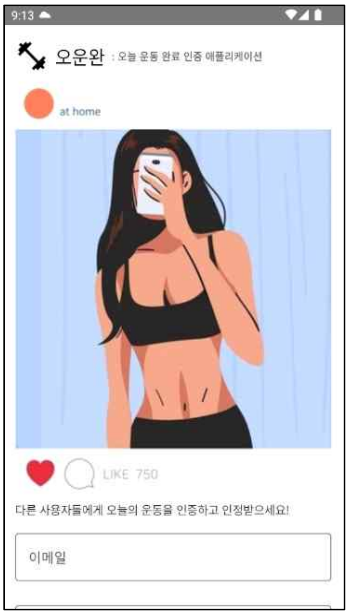
- 사용자는 비밀번호 및 비밀번호 확인 및 사용자 이름을 입력한다.

- 사용자는 회원가입 버튼을 누른다. 시스템은 데이터베이스에 사용자 정보를 저장한다.

- 로그인 실행 화면

성공 시나리오

- 사용자는 로그인 페이지에 접속한다.



- 사용자는 보유한 계정의 아이디와 비밀번호를 입력한다. 사용자는 로그인 버튼을 클릭한다.

- 소스코드

MemberApiService
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member.MemberLoginDto import retrofit2.Call import retrofit2.http.Body import retrofit2.http.POST

```
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member.MemberRegisterDto
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member.MemberResponse
import  kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member.MemberTokenRespo
nse
import retrofit2.http.GET
import retrofit2.http.Query

interface MemberApiService {
    // 회원 가입을 위한 POST 요청을 정의한 함수
    @POST("/user/register")
    fun register(@Body memberRegisterDto: MemberRegisterDto): Call<MemberResponse>

    // 이메일 중복 검사를 위한 GET 요청을 정의한 함수
    @GET("/user/checkId")
    fun checkIdDuplicate(@Query("email") email: String): Call<Boolean>

    // 로그인을 위한 POST 요청을 정의한 함수
    @POST("/user/login")
    fun login(@Body memberLoginDto: MemberLoginDto): Call<MemberTokenResponse>
}
```

- register 함수
- 회원 가입을 위한 POST 요청을 정의한다.
- @POST("/user/register") 어노테이션은 서버의 /user/register 엔드포인트에 POST 요청을 보내는 것을 나타낸다.
- @Body 어노테이션은 함수의 파라미터로 전달되는 memberRegisterDto 객체를 요청의 본문(body)으로 사용하겠다는 것을 나타낸다.

- checkIdDuplicate 함수
- 이메일 중복 검사를 위한 GET 요청을 정의한다.
- @GET("/user/checkId") 어노테이션은 서버의 /user/checkId 엔드포인트에 GET 요청을 보내는 것을 나타낸다.
- @Query("email") email: String 파라미터는 이메일을 검사하기 위한 쿼리 매개변수로 사용된다.

- login 함수
- 로그인을 위한 POST 요청을 정의한다.
- @POST("/user/login") 어노테이션은 서버의 /user/login 엔드포인트에 POST 요청을 보내는 것을 나타낸다.
- @Body 어노테이션은 함수의 파라미터로 전달되는 memberLoginDto 객체를 요청의 본문(body)으로 사용하겠다는 것을 나타낸다.

MemberViewModel

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel

import android.app.Application
import android.util.Log
import androidx.lifecycle.AndroidViewModel
```

```
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ApiService.MemberApiService
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member.MemberLoginDto
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member.MemberRegisterDto
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.NullOnEmptyConverterFactory
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.SharedPreferencesUtils
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class MemberViewModel(application: Application) : AndroidViewModel(application) {
    private val SERVER_URL = "https://port-0-todayfitcomplete-1drvf2llomgqfda.sel5.cloudtype.app"
    private val memberApiService: MemberApiService

    init {
        // Retrofit을 초기화하여 서버와 통신할 준비를 한다.
        val retrofit = Retrofit.Builder()
            .baseUrl(SERVER_URL)
            .addConverterFactory(NullOnEmptyConverterFactory()) // 빈 응답을 처리하기 위한 컨버터
            .addConverterFactory(GsonConverterFactory.create()) // JSON 응답을 파싱하기 위한 컨버터
            .build()

        memberApiService = retrofit.create(MemberApiService::class.java) // Retrofit 인터페이스 구현 생성
    }

    // 회원가입과 관련된 기능
    private val _registrationStatus = MutableLiveData<String?>()
    val registrationStatus: LiveData<String?> = _registrationStatus
    fun register(memberRegisterDto: MemberRegisterDto) {
        viewModelScope.launch {
            try {
                // 회원가입 API를 호출하고 응답을 받아온다.
                val response = withContext(Dispatchers.IO) {
                    memberApiService.register(memberRegisterDto).execute()
                }
                if (response.isSuccessful) {
                    _registrationStatus.postValue("회원가입에 성공했다.")
```

```
                } else {
                    _registrationStatus.postValue("회원가입에 실패했다.")
                    Log.e("MemberViewModel", "Registration failed: ${response.errorBody()?.st
ring()}")
                }
            } catch (e: Exception) {
                Log.e("MemberViewModel", "Error on member registration: ${e.message}")
                _registrationStatus.postValue("오류 발생: ${e.message}")
            }
        }
    }

    // 이메일 중복 검사를 수행하는 함수
    fun checkEmailDuplicate(email: String, onResult: (Boolean) -> Unit) {
        viewModelScope.launch {
            try {
                val response = withContext(Dispatchers.IO) {
                    memberApiService.checkIdDuplicate(email).execute()
                }
                if (response.isSuccessful) {
                    onResult(true)
                } else {
                    Log.e("MemberViewModel", "Check email duplicate failed: ${response.erro
rBody()?.string()}")
                    onResult(false)
                }
            } catch (e: Exception) {
                Log.e("MemberViewModel", "Error checking email duplicate: ${e.message}")
                onResult(false)
            }
        }
    }

    // 로그인과 관련된 기능
    private val _isLoggedIn = MutableLiveData<Boolean>()
    val isLoggedIn: LiveData<Boolean> = _isLoggedIn
    fun login(email: String, password: String) {
        viewModelScope.launch {
            try {
                // 로그인 정보를 담은 DTO 객체 생성
                val memberLoginDto = MemberLoginDto(email, password)
                // 로그인 API를 호출하고 응답을 받아온다.
                val response = withContext(Dispatchers.IO) {
                    memberApiService.login(memberLoginDto).execute()
                }
```

```
                if (response.isSuccessful && response.body() != null) {
                    val token = response.body()!!.token
                    // SharedPreferences를 사용하여 토큰과 사용자 이메일을 저장한다.
                    val context = getApplication<Application>().applicationContext
                    SharedPreferencesUtils.saveToken(context, token)
                    SharedPreferencesUtils.saveEmail(context, email)
                    _isLoggedIn.postValue(true)
                } else {
                    _isLoggedIn.postValue(false)
                }
            } catch (e: Exception) {
                Log.e("MemberViewModel", "Error on member login: ${e.message}")
                _isLoggedIn.postValue(false)
            }
        }
    }
}
```

- MemberViewModel 클래스
- AndroidViewModel 클래스를 확장하여 Android 애플리케이션 컨텍스트를 포함하는 ViewModel을 생성한다.
- Retrofit을 사용하여 서버와 통신하기 위한 memberApiService를 초기화한다.

- register 함수
- 회원 가입 기능을 처리하는 함수이다.
- Coroutine을 사용하여 비동기적으로 실행되며, 회원 가입 API를 호출하고 응답을 처리한다.
- _registrationStatus LiveData를 통해 회원 가입 상태 메시지를 관찰할 수 있다.

- checkEmailDuplicate 함수
- 이메일 중복 검사를 수행하는 함수이다.
- Coroutine을 사용하여 비동기적으로 실행되며, 이메일 중복 확인 API를 호출하고 응답을 처리한다.

- login 함수
- 로그인 기능을 처리하는 함수이다.
- Coroutine을 사용하여 비동기적으로 실행되며, 로그인 API를 호출하고 응답을 처리한다.
- 로그인에 성공하면 토큰과 사용자 이메일을 SharedPreferences를 사용하여 저장하고, _isLoggedIn LiveData를 통해 로그인 상태를 관찰할 수 있다.

MemberDto

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member

// MemberLoginDto 클래스: 사용자 로그인에 필요한 데이터를 나타내는 데이터 전송 객체 (DTO)
data class MemberLoginDto(
    val email: String,      // 사용자 이메일 주소
    val password: String    // 사용자 비밀번호
)

package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member
```

```
// MemberRegisterDto 클래스: 사용자 회원 가입에 필요한 데이터를 나타내는 데이터 전송 객체 (DTO)
data class MemberRegisterDto(
    val email: String,          // 사용자 이메일 주소
    val password: String,       // 사용자 비밀번호
    val passwordCheck: String, // 비밀번호 확인을 위한 필드 (비밀번호와 일치해야 함)
    val username: String        // 사용자의 사용자 이름
)

package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member

// MemberResponse 클래스: 서버에서 클라이언트로 전달되는 사용자 정보를 나타내는 데이터 전송 객체 (DTO)
data class MemberResponse(
    val email: String,     // 사용자 이메일 주소
    val username: String // 사용자의 사용자 이름
)

package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member

// MemberTokenResponse 클래스: 사용자 로그인 후 서버에서 발급된 토큰과 사용자 이메일을 클라이언트로 전달하는 데이터 전송 객체 (DTO)
data class MemberTokenResponse(
    val email: String,  // 사용자 이메일 주소
    val token: String    // 사용자 인증을 위한 토큰
)
```

RegisterScreen

```
import android.widget.Toast
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.*
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.text.KeyboardActions
import androidx.compose.foundation.text.KeyboardOptions
import androidx.compose.foundation.verticalScroll
import androidx.compose.material3.*
import androidx.compose.runtime.*
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Modifier
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.input.ImeAction
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Member.MemberRegisterDto
```

```
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.R
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Screen
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.MemberViewModel

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun RegisterScreen(viewModel: MemberViewModel, navController: NavController) {
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var passwordCheck by remember { mutableStateOf("") }
    var username by remember { mutableStateOf("") }
    var isCheckingEmail by remember { mutableStateOf(false) }
    var emailDuplicationPassed by remember { mutableStateOf(false) }
    val registrationStatus by viewModel.registrationStatus.observeAsState()
    val scrollState = rememberScrollState()

    val context = LocalContext.current

    // 회원 가입 결과 메시지를 토스트 메시지로 표시하는 부분이다.
    LaunchedEffect(registrationStatus) {
        registrationStatus?.let { status ->
            Toast.makeText(context, status, Toast.LENGTH_SHORT).show()
            if (status == "회원가입에 성공했다.") {
                email = ""
                password = ""
                passwordCheck = ""
                username = ""
            }
        }
    }

    // 회원 가입 화면의 UI를 구성하는 부분이다.
    Column(
        modifier = Modifier
            .verticalScroll(scrollState)
            .padding(16.dp)
            .fillMaxWidth()
    ) {
        // 이미지를 표시한다.
        Image(
            painter = painterResource(id = R.drawable.todayfitcomplete_register),
            contentDescription = "타이틀 사진",
            modifier = Modifier.fillMaxWidth(),
            contentScale = ContentScale.Crop
        )
```

```
Spacer(modifier = Modifier.height(16.dp))

// 회원 가입 제목을 표시한다.
Text("회원가입",
    style = MaterialTheme.typography.headlineMedium,
    modifier = Modifier.fillMaxWidth())

Spacer(modifier = Modifier.height(16.dp))

// 이메일 입력 필드와 중복 확인 버튼을 표시한다.
OutlinedTextField(
    value = email,
    onValueChange = {
        email = it
        emailDuplicationPassed = false
    },
    label = { Text("이메일") },
    keyboardOptions = KeyboardOptions.Default.copy(imeAction = ImeAction.Next),
    trailingIcon = {
        if (isCheckingEmail) {
            CircularProgressIndicator()
        } else {
            Button(onClick = {
                isCheckingEmail = true
                viewModel.checkEmailDuplicate(email) { duplicate ->
                    isCheckingEmail = false
                    emailDuplicationPassed = duplicate
                    val message = if (emailDuplicationPassed) {
                        "이메일 사용이 가능한다."
                    } else {
                        "이메일이 중복된다."
                    }
                    Toast.makeText(context, message, Toast.LENGTH_SHORT).show()
                }
            }) {
                Text("중복 확인")
            }
        }
    },
    modifier = Modifier.fillMaxWidth()
)

Spacer(modifier = Modifier.height(8.dp))
```

```
        // 비밀번호 입력 필드와 비밀번호 확인 입력 필드를 표시한다.
        OutlinedTextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("비밀번호") },
            keyboardOptions = KeyboardOptions.Default.copy(imeAction = ImeAction.Next),
            modifier = Modifier.fillMaxWidth()
        )

        Spacer(modifier = Modifier.height(8.dp))

        OutlinedTextField(
            value = passwordCheck,
            onValueChange = { passwordCheck = it },
            label = { Text("비밀번호 확인") },
            keyboardOptions = KeyboardOptions.Default.copy(imeAction = ImeAction.Next),
            modifier = Modifier.fillMaxWidth()
        )

        Spacer(modifier = Modifier.height(8.dp))

        // 사용자 이름 입력 필드를 표시한다.
        OutlinedTextField(
            value = username,
            onValueChange = { username = it },
            label = { Text("사용자 이름") },
            keyboardOptions = KeyboardOptions.Default.copy(imeAction = ImeAction.Done),
            keyboardActions = KeyboardActions(onDone = {
                viewModel.register(MemberRegisterDto(email, password, passwordCheck, user
name))
            }),
            modifier = Modifier.fillMaxWidth()
        )

        Spacer(modifier = Modifier.height(16.dp))

        // 회원 가입 버튼을 표시한다.
        Button(
            onClick = {
                viewModel.register(MemberRegisterDto(email, password, passwordCheck, user
name))
            },
            enabled = emailDuplicationPassed && email.isNotEmpty() && password.isNotEmp
ty() && passwordCheck.isNotEmpty() && username.isNotEmpty(),
            modifier = Modifier.fillMaxWidth()
```

```
        ) {
            Text("회원가입")
        }

        Spacer(modifier = Modifier.height(16.dp))

        // 로그인 화면으로 이동하는 버튼을 표시한다.
        Button(
            onClick = { navController.navigate(Screen.Login.route) },
            modifier = Modifier.fillMaxWidth()
        ) {
            Text("로그인 하러 가기")
        }
    }
}
```

-RegisterScreen

- 이 함수는 사용자가 회원 가입을 할 수 있는 UI를 제공하며, 회원 가입 요청을 ViewModel을 통해 처리한다.
- 이메일 입력 필드: 사용자가 이메일을 입력할 수 있는 텍스트 필드를 표시하고, 입력된 이메일 값을 email 변수에 저장한다. 중복 확인을 위한 버튼을 제공하며, 중복 확인 중에는 로딩 스피너를 표시한다. 중복 확인 버튼을 누를 때, 입력된 이메일이 중복되는지 확인하고 결과를 email DuplicationPassed 변수에 저장한다.
- 비밀번호 입력 필드: 사용자가 비밀번호를 입력할 수 있는 텍스트 필드를 표시하고, 입력된 비밀번호 값을 password 변수에 저장한다.
- 비밀번호 확인 입력 필드: 사용자가 비밀번호를 확인할 수 있는 텍스트 필드를 표시하고, 입력된 값은 passwordCheck 변수에 저장한다.
- 사용자 이름 입력 필드: 사용자가 자신의 이름을 입력할 수 있는 텍스트 필드를 표시하고, 입력된 값을 username 변수에 저장한다.
- 회원 가입 버튼: 사용자가 입력한 정보를 사용하여 회원 가입을 시도하는 버튼이다.
- 버튼은 입력 유효성 검사를 통과하고 중복 확인이 완료된 경우에만 활성화된다.
- 로딩 스피너: 중복 확인 중 또는 회원 가입 진행 중일 때, 로딩 스피너를 표시하여 사용자에게 진행 중임을 시각적으로 알려준다.
- 화면 이동 :회원 가입이 성공하면 입력 필드들을 초기화하고, 사용자를 로그인 화면으로 이동하도록 할 수 있다.

LoginScreen

```
import androidx.compose.foundation.Image
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.rememberScrollState
```

```
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.FitnessCenter
import androidx.compose.material3.Button
import androidx.compose.material3.CircularProgressIndicator
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.layout.ContentScale
import androidx.compose.ui.res.painterResource
import androidx.compose.ui.text.font.FontWeight
import androidx.compose.ui.text.input.PasswordVisualTransformation
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.R
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Screen
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.MemberViewModel

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun LoginScreen(viewModel: MemberViewModel, navController: NavController) {
    var email by remember { mutableStateOf("") }
    var password by remember { mutableStateOf("") }
    var isLoading by remember { mutableStateOf(false) }
    var loginError by remember { mutableStateOf(false) }
    val scrollState = rememberScrollState()

    // ViewModel에서 로그인 상태를 관찰하고, 로그인되었을 때 화면을 이동하는 부분이다.
    viewModel.isLoggedIn.observeAsState().value?.let { loggedIn ->
        if (loggedIn) {
            navController.navigate(Screen.BoardList.route) {
                popUpTo(Screen.Login.route) { inclusive = true } // 로그인 화면을 스택에서
제거
            }
        }
```

```
    }

    // 로그인 화면의 UI를 구성하는 부분이다.
    Column(modifier = Modifier
        .verticalScroll(scrollState)
        .padding(16.dp)
        .fillMaxWidth()) {
        // 화면 상단에 로고와 앱 이름을 표시한다.
        Row(verticalAlignment = Alignment.CenterVertically) {
            Icon(Icons.Filled.FitnessCenter, contentDescription = "오운완 로고", Modifier.size(4
0.dp))
            Spacer(modifier = Modifier.width(8.dp))
            Text(text = "오운완", style = MaterialTheme.typography.titleLarge)
            Spacer(modifier = Modifier.width(8.dp))
            Text(
                text = ": 오늘 운동 완료 인증 애플리케이션",
                style = MaterialTheme.typography.bodySmall.copy(fontWeight = FontWeight.B
old),
                color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.7f)
            )
        }
        Spacer(modifier = Modifier.height(10.dp))

        // 이미지를 표시한다.
        Image(
            painter = painterResource(id = R.drawable.todayfitcomplete_login),
            contentDescription = "로그인 사진",
            modifier = Modifier.fillMaxWidth(),
            contentScale = ContentScale.Crop
        )
        Spacer(modifier = Modifier.height(5.dp))

        // 로그인 안내 메시지를 표시한다.
        Text(
            text = "다른 사용자들에게 오늘의 운동을 인증하고 인정받으세요!",
            style = MaterialTheme.typography.bodyMedium,
            color = MaterialTheme.colorScheme.onSurface
        )
        Spacer(modifier = Modifier.height(10.dp))

        // 이메일 입력 필드
        OutlinedTextField(
            value = email,
            onValueChange = { email = it },
            label = { Text("이메일") },
```

```
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(20.dp))

        // 비밀번호 입력 필드
        OutlinedTextField(
            value = password,
            onValueChange = { password = it },
            label = { Text("비밀번호") },
            visualTransformation = PasswordVisualTransformation(),
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(20.dp))

        // 로그인 버튼
        Button(
            onClick = {
                isLoading = true
                viewModel.login(email, password)
                isLoading = false
            },
            enabled = email.isNotEmpty() && password.isNotEmpty() && !isLoading,
            modifier = Modifier.fillMaxWidth()
        ) {
            if (isLoading) {
                CircularProgressIndicator(color = MaterialTheme.colorScheme.onPrimary)
            } else {
                Text("로그인")
            }
        }

        // 로그인 오류 메시지를 표시한다.
        if (loginError) {
            Text(
                text = "아이디와 비밀번호가 일치하지 않다.",
                color = MaterialTheme.colorScheme.error,
                style = MaterialTheme.typography.bodySmall
            )
        }

        Spacer(modifier = Modifier.height(16.dp))

        // 회원가입 화면으로 이동하는 버튼
        Button(
            onClick = { navController.navigate(Screen.Register.route) },
```

```
                modifier = Modifier.fillMaxWidth()
            ) {
                Text("회원가입 하러 가기")
            }
        }
}
```

-LoginScreen

- 이 함수는 사용자가 로그인을 할 수 있는 UI를 제공하며, 로그인 요청을 ViewModel을 통해 처리한다.
- 이미지: Image를 사용하여 로그인 화면의 배경 이미지를 표시한다.
- 로그인 안내 메시지: 사용자에게 로그인 안내 메시지를 텍스트로 표시한다.
- 이메일 입력 필드: 사용자가 이메일 주소를 입력할 수 있는 OutlinedTextField를 표시하고, 입력된 이메일 값을 email 변수에 저장한다.
- 비밀번호 입력 필드: 사용자가 비밀번호를 입력할 수 있는 OutlinedTextField를 표시하고, 입력된 비밀번호 값을 password 변수에 저장한다.
- 비밀번호는 마스킹 처리되어 보이지 않도록 되어 있다.
- 로그인 버튼: 사용자가 이메일과 비밀번호를 입력한 후, 활성화된 Button을 통해 로그인을 시도할 수 있다. 버튼을 클릭하면 viewModel.login(email, password)이 호출되어 로그인을 시도하며, 로딩 중에는 로딩 스피너가 표시된다.
- 로딩 스피너: 로그인 중에는 로딩 스피너가 회전하여 진행 중임을 나타낸다. 로딩이 완료되면 버튼에 "로그인" 텍스트가 다시 표시된다.
- 로그인 오류 메시지: 로그인 시도가 실패하면 "아이디와 비밀번호가 일치하지 않다."라는 오류 메시지가 표시된다.
- 회원가입 버튼: 로그인 화면 하단에는 "회원가입 하러 가기" 버튼이 있으며, 클릭하면 회원가입 화면으로 이동한다.

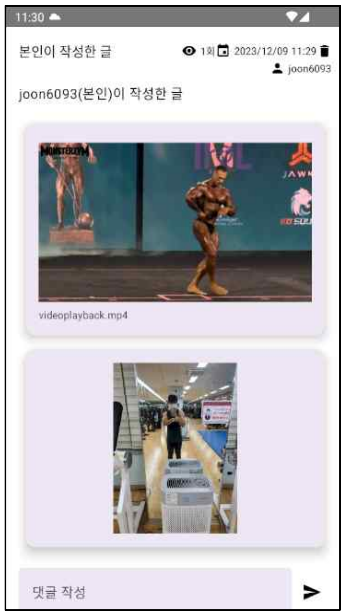
4.1.2 운동 인증 게시글 조회 및 삭제 / 4.1.3 운동 인증 게시글 작성

- 운동 인증 게시글 조회 / 삭제 화면

성공 시나리오

- 사용자는 "운동 인증 게시글 화면"에 접속한다.

- 사용자는 해당 화면에서 원하는 게시글을 확인하고 선택한다. 시스템은 데이터베이스에서 해당 게시글의 정보를 받아온다. 시스템은 받아온 게시글 정보를 사용자에게 표시한다.



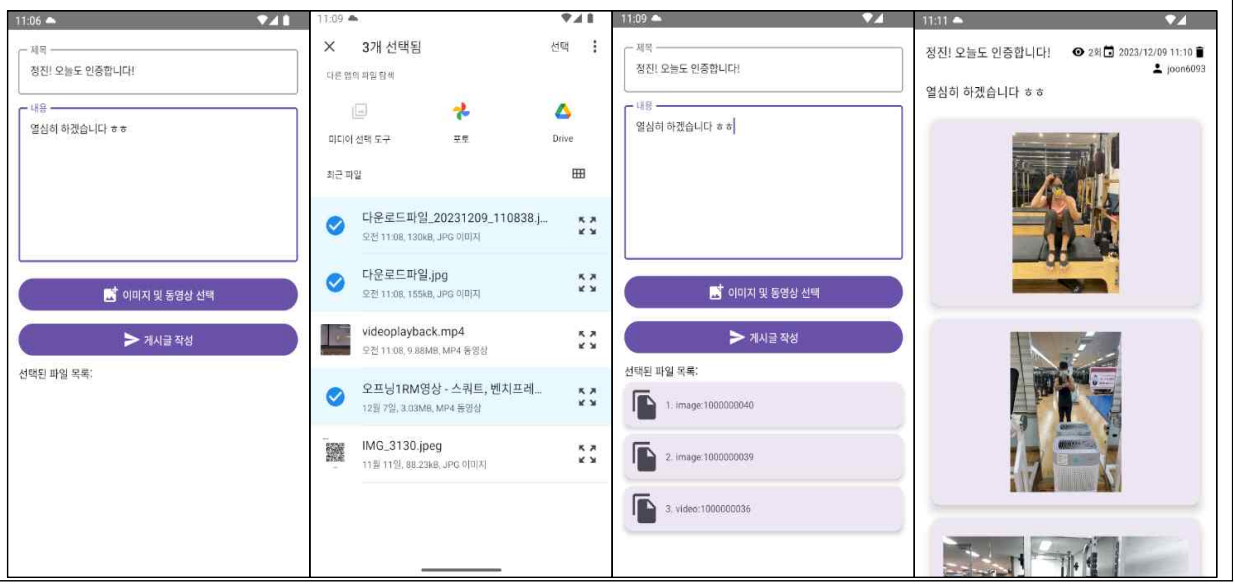
- 사용자는 삭제 버튼을 클릭해 본인이 작성한 글을 삭제한다.

(본인이 작성한 글 삭제 후 없어진 모습) (다른 사람이 작성한 글은 삭제 버튼이 없는 모습)

- 운동 인증 게시글 작성 화면

성공 시나리오
- 사용자는 운동 인증 게시판에 접속한다.

- 사용자는 '게시글 작성' 버튼을 클릭하여 게시글을 작성한다. 게시글 작성 후, 시스템은 게시글 정보를 데이터베이스에 저장한다



- 소스코드

BoardApiService
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ApiService import BoardListPageResponse import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Board.BoardDetailsResponse import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Borad.BoardWriteDto import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Borad.BoardWriteResponse import retrofit2.Call import retrofit2.http.Body import retrofit2.http.DELETE import retrofit2.http.GET

```
import retrofit2.http.Header
import retrofit2.http.POST
import retrofit2.http.Path

interface BoardApiService {
    // 게시판 목록을 가져오는 API 엔드포인트이다.
    @GET("/board/list")
    fun getBoardList(): Call<BoardListPageResponse>

    // 특정 게시글의 상세 정보를 가져오는 API 엔드포인트이다.
    @GET("/board/{boardId}")
    fun getBoardDetails(@Path("boardId") boardId: Long): Call<BoardDetailsResponse>

    // 게시글을 작성하는 API 엔드포인트이다.
    @POST("/board/write")
    fun writePost(@Header("Authorization") token: String?, @Body boardWriteDto:
BoardWriteDto): Call<BoardWriteResponse>

    // 게시글을 삭제하는 API 엔드포인트이다.
    @DELETE("/board/{boardId}/delete")
    fun deleteBoard(@Header("Authorization") token: String?, @Path("boardId") boardId:
Long): Call<Void>
}
```

- BoardApiService 인터페이스
• 게시판과 관련된 API 엔드포인트를 정의한다.
- @GET("/board/list"):
• GET 메서드를 사용하여 게시판 목록을 가져오는 엔드포인트를 정의한다.
- @GET("/board/{boardId}")
• GET 메서드를 사용하여 특정 게시글의 상세 정보를 가져오는 엔드포인트를 정의한다. {boardId}는 동적 경로 변수로, 요청 시 특정 게시글을 식별하기 위해 사용된다.
- @POST("/board/write")
• POST 메서드를 사용하여 게시글을 작성하는 엔드포인트를 정의한다.
• @Header("Authorization") token: String?: 헤더에 인증 토큰을 추가한다. 이를 통해 사용자를 인증한다.
• @Body boardWriteDto: BoardWriteDto: 요청 본문에 게시글 작성을 위한 데이터를 포함하는 DTO 객체를 전달한다.
- @DELETE("/board/{boardId}/delete")
• DELETE 메서드를 사용하여 게시글을 삭제하는 엔드포인트를 정의한다.
• @Header("Authorization") token: String?: 헤더에 인증 토큰을 추가한다. 이를 통해 삭제 권한을 인증한다.
• {boardId}는 동적 경로 변수로, 삭제할 게시글을 식별하기 위해 사용된다.

FileApiService

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ApiService

import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.File.FileUploadResponse
```

```
import okhttp3.MultipartBody
import retrofit2.Call
import retrofit2.http.Header
import retrofit2.http.Multipart
import retrofit2.http.POST
import retrofit2.http.Part
import retrofit2.http.Path

interface FileApiService {
    // 멀티파트 요청을 사용하여 파일 업로드 API를 정의한다.
    @Multipart
    @POST("/board/{boardId}/file/upload")
    fun uploadFile(
        // 헤더에 인증 토큰을 추가하여 사용자를 인증한다.
        @Header("Authorization") token: String,
        // 경로 변수 {boardId}를 동적으로 설정하여 업로드할 게시판을 지정한다.
        @Path("boardId") boardId: Long,
        // 멀티파트 요청의 일부로 업로드할 파일을 첨부한다.
        @Part file: MultipartBody.Part
    ): Call<List<FileUploadResponse>>
}
```

- FileApiService 인터페이스
• 파일 업로드와 관련된 API 엔드포인트를 정의한다.
- @Multipart 어노테이션
• 이 API 메서드가 멀티파트(form-data) 요청을 사용한다는 것을 나타낸다. 멀티파트 요청을 통해 파일을 업로드할 수 있다.
- @POST("/board/{boardId}/file/upload")
• POST 메서드를 사용하여 지정된 엔드포인트에 요청을 보낸다. {boardId}는 동적으로 바인딩되는 경로 변수로, 실제로 요청을 보낼 때 해당 변수에 값을 제공해야 한다. 이를 통해 특정 게시판에 파일을 업로드할 수 있다.
- uploadFile 함수
• 파일 업로드를 처리하는 메서드이다.
• @Header("Authorization") token: String: 헤더에 인증 토큰을 추가한다. 이를 통해 서버는 요청을 보내는 사용자를 인증할 수 있다.
• @Path("boardId") boardId: Long: 엔드포인트의 경로 변수 {boardId}를 동적으로 설정한다. 이 변수는 특정 게시판을 나타내며 업로드할 파일을 연결할 게시판을 지정한다.
• @Part file: MultipartBody.Part: 멀티파트 요청의 일부로 업로드할 파일을 나타낸다. MultipartBody.Part 객체를 사용하여 파일을 첨부한다.
- Call<List<FileUploadResponse>>
• Retrofit의 Call 객체를 반환한다. 이것은 비동기적으로 파일 업로드 요청을 수행하고 서버로부터의 응답을 처리할 수 있도록 한다.
• 파일 업로드에 대한 응답은 FileUploadResponse 객체를 리스트로 받아온다.

BoardViewModel

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel
```

```
import android.app.Application
import android.net.Uri
import android.util.Log
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ApiService.BoardApiService
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ApiService.FileApiService
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Board.BoardDetailsResponse
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Borad.BoardListResponse
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Borad.BoardWriteDto
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.SharedPreferencesUtils
import okhttp3.MediaType.Companion.toMediaTypeOrNull
import okhttp3.MultipartBody
import okhttp3.RequestBody.Companion.asRequestBody
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory
import java.io.File
import java.io.FileOutputStream

class BoardViewModel(application: Application) : AndroidViewModel(application) {
    private val SERVER_URL = "https://port-0-todayfitcomplete-1drvf2llomgqfda.sel5.cloudtype.app"
    private val boardApi: BoardApiService
    private val fileApi: FileApiService
    private val _boardList = MutableLiveData<List<BoardListResponse>>()
    data class FileDetail(val file: File, val mimeType: String, val fileName: String)

    init {
        val retrofit = Retrofit.Builder()
            .baseUrl(SERVER_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        boardApi = retrofit.create(BoardApiService::class.java)
        fileApi = retrofit.create(FileApiService::class.java)
        fetchBoardData()
    }

    // 게시판 목록을 LiveData로 관리
    val boardList: LiveData<List<BoardListResponse>>
```

```
        get() = _boardList

    // 게시판 목록을 가져오는 함수
    private fun fetchBoardData() {
        viewModelScope.launch {
            val response = withContext(Dispatchers.IO) {
                boardApi.getBoardList().execute()
            }
            if (response.isSuccessful) {
                _boardList.value = response.body()?.content
            } else {
                Log.e("fetchBoardData()", "Response not successful")
            }
        }
    }

    // 게시글 상세 정보를 LiveData로 관리
    private val _boardDetails = MutableLiveData<BoardDetailsResponse>()
    val boardDetails: LiveData<BoardDetailsResponse> = _boardDetails

    // 특정 게시글의 상세 정보를 가져오는 함수
    fun fetchBoardDetails(boardId: Long) {
        viewModelScope.launch {
            val response = withContext(Dispatchers.IO) {
                boardApi.getBoardDetails(boardId).execute()
            }
            if (response.isSuccessful && response.body() != null) {
                _boardDetails.postValue(response.body())
            }
        }
    }

    // 게시글 작성 및 파일 업로드 함수
    fun writePostWithFiles(title: String, content: String, fileUris: List<Uri>, onPostComplete:
() -> Unit) {
        val token = SharedPreferencesUtils.getToken(getApplication<Application>().application
Context)
        val boardWriteDto = BoardWriteDto(title, content)

        viewModelScope.launch {
            // 파일 정보 생성
            val fileDetails = fileUris.mapNotNull { uri ->
                val fileName = getFileName(uri)
                val mimeType = getApplication<Application>().contentResolver.getType(uri) ?:
"application/octet-stream"
```

```
                val file = uriToFile(uri, fileName)
                FileDetail(file, mimeType, fileName)
            }
            // 게시글 작성 API 호출
            val postResponse = withContext(Dispatchers.IO) {
                boardApi.writePost("Bearer $token", boardWriteDto).execute()
            }
            if (postResponse.isSuccessful) {
                onPostComplete()
                fetchBoardData()
                val createdBoardId = postResponse.body()?.boardId
                fileDetails.forEach { fileDetail ->
                    if (createdBoardId != null) {
                        uploadFile(createdBoardId, fileDetail, token)
                    }
                }
            }
        }
    }

    // Uri를 File 객체로 변환하는 함수
    private fun uriToFile(uri: Uri, fileName: String): File {
        val inputStream = getApplication<Application>().contentResolver.openInputStream(ur
i)
        val file = File(getApplication<Application>().cacheDir, fileName)
        inputStream.use { input ->
            FileOutputStream(file).use { output ->
                input?.copyTo(output)
            }
        }
        return file
    }

    // 파일 업로드 함수
    private suspend fun uploadFile(boardId: Long, fileDetail: FileDetail, token: String?) {
        val requestBody = fileDetail.file.asRequestBody(fileDetail.mimeType.toMediaTypeOrNull
())
        val multipartBody = MultipartBody.Part.createFormData("file", fileDetail.fileName, req
uestBody)

        val response = withContext(Dispatchers.IO) {
            fileApi.uploadFile("Bearer $token", boardId, multipartBody).execute()
        }
    }
```

```
    // Uri에서 파일 이름을 가져오는 함수
    private fun getFileName(uri: Uri): String {
        var result: String? = null
        if (uri.scheme == "content") {
            val cursor = getApplication<Application>().contentResolver.query(uri, null, null, null, null)
            cursor?.use {
                if (it.moveToFirst()) {
                    result = it.getString(it.getColumnIndexOrThrow("_display_name"))
                }
            }
        }
        if (result == null) {
            result = uri.path
            val cut = result?.lastIndexOf('/')
            if (cut != null && cut != -1) {
                result = result?.substring(cut + 1)
            }
        }
        return result ?: "unknown"
    }

    // 게시글 삭제 함수
    fun deleteBoard(boardId: Long) {
        val token = SharedPreferencesUtils.getToken(getApplication<Application>().applicationContext)
        viewModelScope.launch {
            withContext(Dispatchers.IO) {
                boardApi.deleteBoard("Bearer $token", boardId).execute()
            }
            fetchBoardData()
        }
    }
}
```

- BoardViewModel 클래스
- AndroidViewModel을 상속하며 Android 애플리케이션 컨텍스트를 사용하여 ViewModel을 초기화한다.
- Retrofit을 통해 서버와 통신하기 위한 BoardApiService 및 FileApiService 객체를 생성한다.
- 게시글 목록을 저장하기 위한 MutableLiveData인 _boardList를 초기화하고, 게시글 상세 정보를 저장하기 위한 MutableLiveData인 _boardDetails도 초기화한다.

- fetchBoardData 함수
- Coroutine을 사용하여 백그라운드 스레드에서 게시글 목록을 서버에서 가져온다.
- 결과는 _boardList MutableLiveData에 저장된다.

- fetchBoardDetails 함수

- Coroutine을 사용하여 백그라운드 스레드에서 특정 게시글의 상세 정보를 서버에서 가져온다.
- 결과는 _boardDetails MutableLiveData에 저장된다.

- writePostWithFiles 함수
- 게시글을 작성하고 파일을 업로드하는 함수이다.
- 게시글 제목(title), 내용(content), 그리고 업로드할 파일(fileUris) 정보를 파라미터로 받는다.
- 먼저, 파일 정보를 생성하고 게시글 작성 API를 호출한다.
- 게시글 작성이 성공하면 onPostComplete 함수를 호출하여 작성 완료를 처리하고, 게시글 목록을 업데이트한다.
- 그 후, 파일 정보를 순회하면서 각 파일을 업로드한다.

- uriToFile 함수
- Uri를 File 객체로 변환하는 함수이다. 업로드할 파일을 File 객체로 변환하는 데 사용된다.

- uploadFile 함수
- 파일을 업로드하는 함수로, 게시글 작성 시 호출된다.
- 파일 업로드를 위해 Retrofit을 사용하여 서버에 Multipart 요청을 보낸다.

- getFileName 함수
- Uri에서 파일 이름을 가져오는 함수이다. Uri를 통해 얻은 파일의 이름을 파악하는 데 사용된다.

- deleteBoard 함수
- 게시글 삭제를 위한 함수로, 게시글 ID(boardId)를 파라미터로 받는다.
- 토큰을 헤더에 포함하여 게시글 삭제 API를 호출하고, 삭제 후에는 게시글 목록을 업데이트한다.

BoradDto

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Board

import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment.CommentListResponse

// BoardDetailsResponse 클래스: 특정 게시글의 상세 정보를 나타낸다.
data class BoardDetailsResponse(
    val boardId: Long,                      // 게시글 고유 ID
    val title: String,                     // 게시글 제목
    val content: String,                    // 게시글 내용
    val viewCount: Int,                     // 게시글 조회수
    val writerName: String,                 // 작성자 이름
    val createdDate: String,                // 작성일
    val modifiedDate: String,               // 수정일
    val comments: List<CommentListResponse> = listOf(), // 댓글 목록 (기본값은 빈 목록)
    val files: List<BoardDetailsFileResponse> = listOf()  // 파일 목록 (기본값은 빈 목록)
)

// BoardDetailsFileResponse 클래스: 게시글에 첨부된 파일의 정보를 나타낸다.
data class BoardDetailsFileResponse(
    val fileId: Long,                      // 파일 고유 ID
    val originFileName: String,             // 원본 파일 이름
    val fileType: String,                  // 파일 유형
```

```
    val filePath: String                    // 파일 경로
)
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Board.BoardListResponse

// BoardListPageResponse 클래스: 게시글 목록을 페이지로 나누어 응답할 때 사용된다.
data class BoardListPageResponse(
    val content: List<BoardListResponse>,  // 게시글 목록
)
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Board

// BoardListResponse 클래스: 게시글 목록에서 각 게시글의 요약 정보를 나타낸다.
data class BoardListResponse(
    val boardId: Long,                      // 게시글 고유 ID
    val title: String,                     // 게시글 제목
    val content: String,                    // 게시글 내용
    val viewCount: Int,                     // 게시글 조회수
    val createdDate: String,                // 작성일
    val modifiedDate: String,               // 수정일
    val writerName: String                  // 작성자 이름
)
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Board

// BoardWriteDto 클래스: 게시글을 작성할 때 필요한 데이터를 나타낸다.
data class BoardWriteDto(
    val title: String,                     // 게시글 제목
    val content: String                     // 게시글 내용
)
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Board

// BoardWriteResponse 클래스: 게시글 작성 후 서버에서 반환되는 응답 데이터를 나타낸다.
data class BoardWriteResponse(
    val boardId: Long,                      // 생성된 게시글의 고유 ID
    val title: String,                     // 게시글 제목
    val content: String,                    // 게시글 내용
    val writerName: String,                 // 작성자 이름
    val createdDate: String                 // 작성일
)
```

FileDto

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.File


// FileUploadResponse 클래스: 파일 업로드 후 서버에서 반환되는 응답 데이터를 나타낸다.
data class FileUploadResponse(
    val fileId: Long,                // 파일의 고유 ID
    val originFileName: String,    // 업로드된 파일의 원본 파일 이름
    val filePath: String,           // 업로드된 파일의 서버 상의 저장 경로
    val fileType: String             // 파일의 유형 (예: 이미지, 동영상 등)
)
```

BoardListScreen

```
import androidx.compose.foundation.clickable
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Box
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.PaddingValues
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxSize
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.lazy.LazyColumn
import androidx.compose.foundation.lazy.items
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Add
import androidx.compose.material.icons.filled.Event
import androidx.compose.material.icons.filled.FitnessCenter
import androidx.compose.material.icons.filled.Person
import androidx.compose.material.icons.filled.Visibility
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.FloatingActionButton
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
```

```
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Board.BoardListResponse
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.BoardViewModel

@Composable
fun BoardListScreen(viewModel: BoardViewModel, navController: NavController) {
    // ViewModel에서 게시판 목록을 가져온다.
    val boardList by viewModel.boardList.observeAsState(initial = emptyList())

    // 화면 전체를 채우는 Box를 만듭니다.
    Box(modifier = Modifier.fillMaxSize()) {
        Column(modifier = Modifier.padding(8.dp)) {
            // 화면 상단에 앱 로고와 제목을 표시한다.
            Row(verticalAlignment = Alignment.CenterVertically) {
                Icon(Icons.Filled.FitnessCenter, contentDescription = "오운완 로고", Modifier.si
ze(40.dp))
                Spacer(modifier = Modifier.width(8.dp))
                Text(text = "오늘의 오운완", style = MaterialTheme.typography.titleLarge)
                Spacer(modifier = Modifier.width(8.dp))
            }

            // LazyColumn을 사용하여 게시판 목록을 스크롤 가능한 리스트로 표시한다.
            LazyColumn(
                verticalArrangement = Arrangement.spacedBy(5.dp),
                contentPadding = PaddingValues(horizontal = 8.dp)
            ) {
                items(boardList) { board ->
                    // 각 게시물 항목을 표시하는 BoardListItem을 호출한다.
                    BoardListItem(board = board) {
                        // 게시물 항목을 클릭하면 해당 게시물의 상세 화면으로 이동한다.
                        navController.navigate("boardDetail/${board.boardId}")
                    }
                }
            }
        }

        // 화면 하단에 게시글 작성 버튼(FloatingActionButton)을 표시한다.
        FloatingActionButton(
            onClick = { navController.navigate("boardWrite") },
            content = { Icon(Icons.Filled.Add, "게시글 작성") },
            modifier = Modifier
                .align(Alignment.BottomEnd)
                .padding(16.dp)
        )
    }
}
```

```
@Composable
fun BoardListItem(board: BoardListResponse, onClick: () -> Unit) {
    // 각 게시물 항목을 Card로 감싸서 표시한다.
    Card(
        modifier = Modifier
            .padding(8.dp)
            .fillMaxWidth()
            .clickable(onClick = onClick), // 항목을 클릭하면 onClick 이벤트가 발생한다.
        elevation = CardDefaults.cardElevation(8.dp)
    ) {
        Column(modifier = Modifier.padding(16.dp)) {
            // 게시물 제목과 조회수, 작성일을 표시한다.
            Row(
                horizontalArrangement = Arrangement.SpaceBetween,
                verticalAlignment = Alignment.CenterVertically,
                modifier = Modifier.fillMaxWidth()
            ) {
                Text(text = board.title, style = MaterialTheme.typography.titleMedium)
                Row(verticalAlignment = Alignment.CenterVertically) {
                    Icon(Icons.Filled.Visibility, contentDescription = "조회수", Modifier.size(18.
dp))
                    Spacer(modifier = Modifier.width(4.dp))
                    Text(text = "${board.viewCount}회", style = MaterialTheme.typography.bo
dySmall)
                    Spacer(modifier = Modifier.width(8.dp))
                    Icon(Icons.Filled.Event, contentDescription = "작성일", Modifier.size(18.d
p))
                    Spacer(modifier = Modifier.width(4.dp))
                    Text(text = board.createdDate.substringBeforeLast(':'), style = MaterialTh
eme.typography.bodySmall)
                }
            }
            // 작성자 정보를 표시한다.
            Row(
                modifier = Modifier.fillMaxWidth(),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.End
            ) {
                Icon(Icons.Filled.Person, contentDescription = "작성자", Modifier.size(18.dp))
                Spacer(modifier = Modifier.width(4.dp))
                Text(text = board.writerName, style = MaterialTheme.typography.bodySmall)
            }
            Spacer(modifier = Modifier.height(8.dp))
            // 게시물 내용을 표시한다.
```

```
            Text(text = board.content, style = MaterialTheme.typography.bodyLarge)
        }
    }
}
```

- BoardListScreen
- 이 함수는 게시판 목록 화면을 정의한다.
- boardList는 ViewModel에서 가져온 게시판 목록을 저장하고 화면에 표시하는 데 사용된다.
- 화면에는 앱 로고와 제목이 표시되며, 게시판 목록은 스크롤 가능한 리스트로 표시된다.
- 각 게시물 항목은 BoardListItem으로 표시되며 클릭하면 해당 게시물의 상세 정보 화면으로 이동한다.
- 화면 하단에는 "게시글 작성" 버튼이 표시되며 클릭하면 게시글 작성 화면으로 이동한다.

- BoardListItem
- 이 함수는 게시물 항목을 표시하는 카드 형태의 UI를 정의한다.
- 각 게시물의 제목, 조회수, 작성일 및 작성자 정보를 표시한다.
- 게시물 항목은 클릭 가능하며 클릭하면 해당 게시물의 상세 정보 화면으로 이동한다.

BoardDetailScreen

```
import android.widget.VideoView
import androidx.compose.foundation.layout.Arrangement
import androidx.compose.foundation.layout.Column
import androidx.compose.foundation.layout.Row
import androidx.compose.foundation.layout.Spacer
import androidx.compose.foundation.layout.fillMaxWidth
import androidx.compose.foundation.layout.height
import androidx.compose.foundation.layout.padding
import androidx.compose.foundation.layout.size
import androidx.compose.foundation.layout.width
import androidx.compose.foundation.rememberScrollState
import androidx.compose.foundation.shape.RoundedCornerShape
import androidx.compose.foundation.verticalScroll
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.Delete
import androidx.compose.material.icons.filled.Event
import androidx.compose.material.icons.filled.Person
import androidx.compose.material.icons.filled.Send
import androidx.compose.material.icons.filled.Visibility
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.Divider
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.IconButton
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.Text
import androidx.compose.material3.TextField
import androidx.compose.runtime.Composable
```

```
import androidx.compose.runtime.LaunchedEffect
import androidx.compose.runtime.getValue
import androidx.compose.runtime.livedata.observeAsState
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.draw.alpha
import androidx.compose.ui.draw.clip
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import androidx.compose.ui.viewinterop.AndroidView
import androidx.navigation.NavController
import coil.compose.AsyncImage
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment.CommentDto
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.SharedPreferencesUtils
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.BoardViewModel
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.CommentViewModel

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun BoardDetailScreen(boardViewModel: BoardViewModel, commentViewModel: CommentView
Model, boardId: Long, navController: NavController) {
    // 새로 작성한 댓글 내용을 저장하는 변수
    var newCommentText by remember { mutableStateOf("") }
    // 게시물 상세 정보 및 댓글 목록을 가져옴
    val boardDetails by boardViewModel.boardDetails.observeAsState()
    val comments by commentViewModel.commentsLiveData.observeAsState()
    val context = LocalContext.current
    // 현재 사용자의 이메일 주소를 가져옴
    val currentUserEmail = SharedPreferencesUtils.getEmail(context)
    // 스크롤 상태를 저장하는 변수
    val scrollState = rememberScrollState()

    // 화면이 처음 열릴 때 게시물 상세 정보와 댓글 목록을 가져옴
    LaunchedEffect(boardId) {
        boardViewModel.fetchBoardDetails(boardId)
        commentViewModel.fetchComments(boardId)
    }

    // 게시물 상세 정보가 있을 경우 아래의 화면을 그립니다.
    boardDetails?.let { details ->
        Column(
            modifier = Modifier
```

```
                .verticalScroll(scrollState)
                .padding(16.dp)
        ) {
            // 게시물 제목과 작성자 정보 표시
            Row(
                horizontalArrangement = Arrangement.SpaceBetween,
                verticalAlignment = Alignment.CenterVertically,
                modifier = Modifier.fillMaxWidth()
            ) {
                Text(text = details.title, style = MaterialTheme.typography.titleMedium)
                Row(verticalAlignment = Alignment.CenterVertically) {
                    // 조회수와 작성일 정보 표시
                    Icon(Icons.Filled.Visibility, contentDescription = "조회수", Modifier.size(18.
dp))
                    Spacer(modifier = Modifier.width(4.dp))
                    Text(text = "${details.viewCount}회", style = MaterialTheme.typography.bo
dySmall)
                    Icon(Icons.Filled.Event, contentDescription = "작성일", Modifier.size(18.d
p))
                    Spacer(modifier = Modifier.width(4.dp))
                    Text(
                        text = details.createdDate.substringBeforeLast(':'),
                        style = MaterialTheme.typography.bodySmall
                    )
                    // 현재 사용자가 게시물 작성자인 경우 삭제 버튼 표시
                    if (details.writerName == currentUserEmail) {
                        IconButton(
                            onClick = {
                                boardViewModel.deleteBoard(boardId)
                                navController.popBackStack() // 게시글 삭제 후 이전 화면으로
돌아가기
                            },
                            modifier = Modifier.size(18.dp)
                        ) {
                            Icon(Icons.Filled.Delete, contentDescription = "게시글 삭제")
                        }
                    }
                }
            }
            // 작성자 정보 표시
            Row(
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.End,
                modifier = Modifier.fillMaxWidth()
            ) {
```

```
                Icon(Icons.Filled.Person, contentDescription = "작성자", Modifier.size(18.dp))
                Spacer(modifier = Modifier.width(4.dp))
                Text(text = details.writerName, style = MaterialTheme.typography.bodySmall)
            }
            Spacer(modifier = Modifier.height(8.dp))
            // 게시물 내용 표시
            Text(text = details.content, style = MaterialTheme.typography.bodyLarge)
            Spacer(modifier = Modifier.height(16.dp))

            // 첨부 파일 표시
            details.files.forEach { file ->
                Card(
                    modifier = Modifier
                        .padding(8.dp)
                        .fillMaxWidth(),
                    elevation = CardDefaults.cardElevation(8.dp)
                ) {
                    Column(modifier = Modifier.padding(16.dp)) {
                        when {
                            // 이미지 파일인 경우
                            file.fileType.startsWith("image") -> {
                                AsyncImage(
                                    model = "https://port-0-todayfitcomplete-1drvf2llomgqfda.sel5.cloudtype.app/files/${file.filePath}",
                                    contentDescription = "첨부 이미지",
                                    modifier = Modifier
                                        .height(200.dp)
                                        .fillMaxWidth()
                                        .clip(RoundedCornerShape(10.dp))
                                )
                            }

                            // 비디오 파일인 경우
                            file.fileType.startsWith("video") -> {
                                AndroidView(
                                    factory = { context ->
                                        VideoView(context).apply {
                                            // 비디오 파일의 URL을 설정하고 클릭 시 재생 또는 일시 정지
                                            setVideoPath("https://port-0-todayfitcomplete-1drvf2llomgqfda.sel5.cloudtype.app/files/${file.filePath}")
                                            setOnClickListener {
                                                if (!isPlaying) {
                                                    start() // 클릭 시 비디오 재생
                                                } else {
```

```
                                        pause() // 이미 재생 중일 경우 일시 정지
                                    }
                                }
                            }
                        },
                        modifier = Modifier
                            .height(200.dp)
                            .fillMaxWidth()
                    )
                    Text(
                        text = file.originFileName,
                        style = MaterialTheme.typography.bodySmall
                    )
                }
            }
        }
    }
}
Spacer(modifier = Modifier.height(16.dp))

// 댓글 목록 표시
comments?.forEach { comment ->
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(vertical = 4.dp)
    ) {
        Row(
            verticalAlignment = Alignment.CenterVertically,
            horizontalArrangement = Arrangement.SpaceBetween,
            modifier = Modifier.fillMaxWidth()
        ) {
            Column(modifier = Modifier.weight(1f)) {
                Row(
                    modifier = Modifier.fillMaxWidth(),
                    verticalAlignment = Alignment.CenterVertically,
                    horizontalArrangement = Arrangement.SpaceBetween
                ) {
                    Row(verticalAlignment = Alignment.CenterVertically) {
                        Icon(
                            Icons.Filled.Person,
                            contentDescription = "작성자",
                            Modifier.size(16.dp)
                        )
                        Spacer(modifier = Modifier.width(4.dp))
```

```
                                    Text(
                                        text = comment.commentWriterName,
                                        style = MaterialTheme.typography.bodySmall
                                    )
                                }
                                Row(verticalAlignment = Alignment.CenterVertically) {
                                    Icon(
                                        Icons.Filled.Event,
                                        contentDescription = "작성일",
                                        Modifier.size(16.dp)
                                    )
                                    Spacer(modifier = Modifier.width(4.dp))
                                    Text(
                                        text = comment.createdDate.substringBeforeLast(':'),
                                        style = MaterialTheme.typography.bodySmall
                                    )
                                }
                            }
                            Row(
                                modifier = Modifier.fillMaxWidth(),
                                verticalAlignment = Alignment.CenterVertically,
                                horizontalArrangement = Arrangement.SpaceBetween
                            ) {
                                Row(verticalAlignment = Alignment.CenterVertically) {
                                    Text(
                                        text = comment.content,
                                        style = MaterialTheme.typography.bodyMedium
                                    )
                                }
                                Row(verticalAlignment = Alignment.CenterVertically) {
                                    // 댓글 작성자인 경우 삭제 버튼 표시
                                    IconButton(
                                        onClick = {
                                            if (comment.commentWriterName == currentUse
rEmail) {
                                                commentViewModel.deleteComment(commen
t.commentId)
                                                commentViewModel.fetchComments(boardId)
                                            }
                                        },
                                        enabled = comment.commentWriterName == current
UserEmail,
                                        modifier = Modifier.alpha(if (comment.commentWrite
rName == currentUserEmail) 1f else 0f)
                                    ) {
```

```
                                        Icon(Icons.Filled.Delete, contentDescription = "삭제")
                                    }
                                }
                            }
                        }
                    }
                    // 댓글 사이에 구분선 추가
                    Divider(color = MaterialTheme.colorScheme.onSurface.copy(alpha = 0.2
f))
                }
            }

            // 댓글 작성 입력란 및 게시 버튼
            Row(
                modifier = Modifier.fillMaxWidth(),
                verticalAlignment = Alignment.CenterVertically,
                horizontalArrangement = Arrangement.SpaceBetween
            ) {
                TextField(
                    value = newCommentText,
                    onValueChange = { newCommentText = it },
                    label = { Text("댓글 작성") },
                    modifier = Modifier.weight(1f)
                )
                IconButton(onClick = {
                    commentViewModel.postComment(boardId, CommentDto(newCommentTex
t))
                    newCommentText = ""
                }) {
                    Icon(Icons.Filled.Send, contentDescription = "게시")
                }
            }
        }
    }
}
```

- BoardDetailScreen

- 이 함수는 게시물의 상세 정보와 댓글을 표시하는 화면을 정의한다.
- newCommentText는 새로 작성된 댓글 내용을 저장하고 업데이트하는 데 사용된다.
- boardDetails 및 comments는 ViewModel에서 가져온 게시물 상세 정보와 댓글 목록을 저장하고 화면에 표시하는 데 사용된다.
- 게시물 내용, 작성자 정보, 첨부 파일, 댓글 목록 등이 화면에 표시된다.
- 사용자가 댓글을 작성하고 삭제할 수 있는 입력란 및 버튼이 제공된다.
- AsyncImage: 이미지 파일을 비동기적으로 로드하고 표시하는 데 사용된다.
- AndroidView: 비디오 파일을 표시하고 제어하는 데 사용된다. 게시물에 첨부된 비디오 파일을

표시하고 사용자가 재생 및 일시 정지할 수 있다.

- 댓글 목록: 댓글 목록을 반복하며 각 댓글의 작성자, 작성일, 내용을 표시한다. 작성자는 댓글을 작성한 사용자를 나타낸다. 작성일은 댓글이 작성된 날짜 및 시간을 표시한다. 사용자가 자신의 댓글을 삭제할 수 있는 버튼도 제공된다.
- TextField 및 IconButton: 사용자가 새로운 댓글을 작성하고 게시하는 데 사용된다. TextField는 사용자가 댓글을 입력하는 데 사용되며, IconButton을 클릭하여 댓글을 게시할 수 있다.

BoardWriteScreen

```
import android.net.Uri
import androidx.activity.compose.rememberLauncherForActivityResult
import androidx.activity.result.contract.ActivityResultContracts
import androidx.compose.foundation.layout.*
import androidx.compose.material.icons.Icons
import androidx.compose.material.icons.filled.AddPhotoAlternate
import androidx.compose.material.icons.filled.FileCopy
import androidx.compose.material.icons.filled.Send
import androidx.compose.material3.Button
import androidx.compose.material3.Card
import androidx.compose.material3.CardDefaults
import androidx.compose.material3.ExperimentalMaterial3Api
import androidx.compose.material3.Icon
import androidx.compose.material3.MaterialTheme
import androidx.compose.material3.OutlinedTextField
import androidx.compose.material3.Text
import androidx.compose.runtime.Composable
import androidx.compose.runtime.getValue
import androidx.compose.runtime.mutableStateOf
import androidx.compose.runtime.remember
import androidx.compose.runtime.setValue
import androidx.compose.ui.Alignment
import androidx.compose.ui.Modifier
import androidx.compose.ui.platform.LocalContext
import androidx.compose.ui.unit.dp
import androidx.navigation.NavController
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.BoardViewModel

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun BoardWriteScreen(viewModel: BoardViewModel, navController: NavController) {
    // 게시물 제목과 내용을 저장하는 변수
    var title by remember { mutableStateOf("") }
    var content by remember { mutableStateOf("") }
    // 선택된 파일 URI 목록을 저장하는 변수
    var selectedFileUris by remember { mutableStateOf<List<Uri>>(emptyList()) }

    val context = LocalContext.current
```

```
    // 갤러리 열기 액티비티를 실행하는 런처를 초기화
    val galleryLauncher = rememberLauncherForActivityResult(
        contract = ActivityResultContracts.GetMultipleContents()
    ) { uris: List<Uri>? ->
        uris?.let {
            selectedFileUris = it
        }
    }

    Column(modifier = Modifier.padding(16.dp)) {
        // 제목 입력 필드
        OutlinedTextField(
            value = title,
            onValueChange = { title = it },
            label = { Text("제목") },
            modifier = Modifier.fillMaxWidth()
        )
        Spacer(modifier = Modifier.height(8.dp))

        // 내용 입력 필드 (다중 라인)
        OutlinedTextField(
            value = content,
            onValueChange = { content = it },
            label = { Text("내용") },
            modifier = Modifier
                .fillMaxWidth()
                .height(200.dp)
        )
        Spacer(modifier = Modifier.height(16.dp))

        // 갤러리 열기 버튼
        Button(
            onClick = { galleryLauncher.launch("*/*") },
            modifier = Modifier.fillMaxWidth()
        ) {
            Icon(Icons.Default.AddPhotoAlternate, contentDescription = null)
            Spacer(modifier = Modifier.width(4.dp))
            Text("이미지 및 동영상 선택")
        }
        Spacer(modifier = Modifier.height(8.dp))

        // 게시글 작성 버튼
        Button(
            onClick = {
                // 제목, 내용, 선택된 파일 URI 목록을 사용하여 게시글 작성 요청
```

```
                viewModel.writePostWithFiles(title, content, selectedFileUris) {
                    navController.popBackStack()
                }
            },
            modifier = Modifier.fillMaxWidth()
        ) {
            Icon(Icons.Default.Send, contentDescription = null)
            Spacer(modifier = Modifier.width(4.dp))
            Text("게시글 작성")
        }

        Spacer(modifier = Modifier.height(8.dp))

        // 선택된 파일 목록 표시
        Text("선택된 파일 목록:", style = MaterialTheme.typography.bodyMedium)
        selectedFileUris.forEachIndexed { index, uri ->
            Card(
                modifier = Modifier
                    .padding(vertical = 4.dp)
                    .fillMaxWidth(),
                elevation = CardDefaults.cardElevation(4.dp)
            ) {
                Row(
                    modifier = Modifier.padding(8.dp),
                    verticalAlignment = Alignment.CenterVertically
                ) {
                    // 파일 아이콘
                    Icon(
                        imageVector = Icons.Default.FileCopy,
                        contentDescription = "파일",
                        modifier = Modifier.size(40.dp)
                    )
                    Spacer(modifier = Modifier.width(8.dp))
                    // 파일 정보
                    Column {
                        Text(
                            text = "${index + 1}. ${uri.lastPathSegment}",
                            style = MaterialTheme.typography.bodySmall
                        )
                    }
                }
            }
        }
    }
}
```

- BoardWriteScreen

- 이 함수는 앱에서 게시글을 작성할 수 있는 기능을 제공한다.
- title 및 content 변수: 게시글의 제목과 내용을 저장하는 변수이다.
- selectedFileUris 변수: 선택한 파일(이미지 또는 동영상)의 URI를 저장하는 리스트이다.
- galleryLauncher: 갤러리에서 이미지 또는 동영상을 선택하는 액티비티를 실행하기 위한 런처이다. ActivityResultContracts.GetMultipleContents()를 사용하여 갤러리 열기 액티비티의 결과를 처리한다.
- Column: 화면을 세로로 배치하는 컬럼이다. 제목, 내용 입력 필드, 파일 선택 버튼, 게시글 작성 버튼 및 선택된 파일 목록을 표시하는 부분을 포함한다.
- OutlinedTextField: 제목과 내용을 입력하는 데 사용되는 텍스트 필드이다.
- Button: 버튼을 클릭하면 해당 동작을 수행하는 버튼이다. "이미지 및 동영상 선택" 버튼은 갤러리를 열어 파일을 선택하도록 한다. "게시글 작성" 버튼은 게시글을 작성하고 서버에 업로드한다.
- Text: 선택된 파일 목록을 표시하는 텍스트이다. 선택된 파일은 카드 레이아웃으로 표시되며, 파일 이름과 파일 아이콘을 포함한다.

4.1.4 운동 인증 게시글 댓글 작성 및 조회 및 삭제

- 운동 인증 게시글 댓글 작성 및 조회 및 삭제

성공시나리오

- 사용자는 운동 인증 게시글에 접속한다. 시스템은 운동 인증 게시글 정보 및 댓글 목록을 제공한다.

- 사용자는 해당 게시글에 댓글을 입력한다.

- 사용자는 '댓글 작성' 버튼을 클릭하여 댓글을 작성한다. 댓글 작성 후, 시스템은 게시글 정보를 데이터베이스에 저장한다. 사용자는 본인이 작성한 댓글을 포함한 댓글 목록을 확인할 수 있다.

- 사용자는 본인이 작성한 댓글을 삭제할 수 있다.

- 소스코드

CommentApiService

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ApiService

import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment.CommentDto
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment.CommentListPageResponse
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment.CommentListResponse
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.DELETE
import retrofit2.http.GET
import retrofit2.http.Header
import retrofit2.http.POST
import retrofit2.http.Path

interface CommentApiService {
    // 새로운 댓글을 게시판에 작성하는 HTTP POST 요청을 정의한다.
    @POST("/board/{boardId}/comment/write")
    fun postComment(
        @Path("boardId") boardId: Long,            // 게시판의 고유 식별자
        @Header("Authorization") token: String?, // JWT 토큰 (사용자 인증)
        @Body commentDto: CommentDto               // 댓글 데이터
    ): Call<CommentListResponse>

    // 게시판의 댓글 목록을 가져오는 HTTP GET 요청을 정의한다.
    @GET("/board/{boardId}/comment/list")
    fun getComments(@Path("boardId") boardId: Long): Call<CommentListPageResponse>

    // 게시판의 특정 댓글을 삭제하는 HTTP DELETE 요청을 정의한다.
    @DELETE("/board/{boardId}/comment/delete/{commentId}")
    fun deleteComment(
        @Path("commentId") commentId: Long,       // 삭제할 댓글의 고유 식별자
```

```
        @Header("Authorization") token: String?  // JWT 토큰 (사용자 인증)
    ): Call<Void>
}
```

- postComment 함수
- HTTP POST 요청을 통해 게시판에 새로운 댓글을 작성하는 역할을 한다.
- @POST("/board/{boardId}/comment/write")은 요청의 HTTP 메서드와 엔드포인트를 정의한다.
- @Path("boardId") boardId: Long는 엔드포인트 URL의 경로에 동적으로 값을 넣어주기 위한 변수이다. boardId는 게시판의 고유 식별자이다.
- @Header("Authorization") token: String?은 HTTP 요청 헤더에 JWT 토큰을 포함시킨다. 토큰은 사용자 인증에 사용된다.
- @Body commentDto: CommentDto는 HTTP 요청 본문에 들어갈 데이터를 정의한다. CommentDto는 게시판 댓글의 내용을 포함하는 데이터 클래스이다.
- 함수의 반환 값은 Call<CommentListResponse>로, Retrofit의 Call 객체를 반환하여 비동기 HTTP 요청을 수행한다.

- getComments 함수
- HTTP GET 요청을 통해 게시판의 댓글 목록을 가져오는 역할을 한다.
- @GET("/board/{boardId}/comment/list")는 요청의 HTTP 메서드와 엔드포인트를 정의한다.
- @Path("boardId") boardId: Long는 엔드포인트 URL의 경로에 동적으로 값을 넣어주기 위한 변수로, 게시판의 고유 식별자를 나타낸다.
- 함수의 반환 값은 Call<CommentListPageResponse>로, Retrofit의 Call 객체를 반환하여 비동기 HTTP 요청을 수행한다.

- deleteComment 함수
- HTTP DELETE 요청을 통해 게시판의 특정 댓글을 삭제하는 역할을 한다.
- @DELETE("/board/{boardId}/comment/delete/{commentId}")는 요청의 HTTP 메서드와 엔드포인트를 정의한다.
- @Path("commentId") commentId: Long는 엔드포인트 URL의 경로에 동적으로 값을 넣어주기 위한 변수로, 삭제할 댓글의 고유 식별자를 나타낸다.
- @Header("Authorization") token: String?은 HTTP 요청 헤더에 JWT 토큰을 포함시킨다. 토큰은 사용자 인증에 사용된다.
- 함수의 반환 값은 Call<Void>로, Retrofit의 Call 객체를 반환하여 비동기 HTTP 요청을 수행한다.

CommentViewModel

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel


import android.app.Application
import android.util.Log
import androidx.lifecycle.AndroidViewModel
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.viewModelScope
import kotlinx.coroutines.Dispatchers
import kotlinx.coroutines.launch
import kotlinx.coroutines.withContext
```

```
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ApiService.CommentApiService
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment.CommentDto
import  kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment.CommentListRespo
nse
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.SharedPreferencesUtils
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class CommentViewModel(application: Application) : AndroidViewModel(application) {
    // 서버의 기본 URL
    private val SERVER_URL = "https://port-0-todayfitcomplete-1drvf2llomgqfda.sel5.cloudty
pe.app"

    // Retrofit을 사용하여 API 서비스를 생성
    private val apiService: CommentApiService

    init {
        val retrofit = Retrofit.Builder()
            .baseUrl(SERVER_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build()

        apiService = retrofit.create(CommentApiService::class.java)
    }

    // 게시판의 댓글 목록을 저장하는 MutableLiveData
    private val _comments = MutableLiveData<List<CommentListResponse>>()
    val commentsLiveData: LiveData<List<CommentListResponse>>
        get() = _comments

    // 게시판의 댓글 목록을 가져오는 함수
    fun fetchComments(boardId: Long) {
        viewModelScope.launch {
            try {
                val response = withContext(Dispatchers.IO) {
                    apiService.getComments(boardId).execute()
                }
                if (response.isSuccessful) {
                    _comments.value = response.body()?.content
                } else {
                    Log.e("fetchComments()", "Response not successful: ${response.errorBody
()?.string()}")
                }
            } catch (e: Exception) {
                Log.e("fetchComments()", "Error fetching comments: ${e.message}")
```

```
            }
        }
    }

    // 댓글 작성 결과를 저장하는 MutableLiveData
    private val _commentResponse = MutableLiveData<CommentListResponse?>()
    val commentResponseLiveData: LiveData<CommentListResponse?>
        get() = _commentResponse

    // 댓글을 게시판에 작성하는 함수
    fun postComment(boardId: Long, commentDto: CommentDto) {
        viewModelScope.launch {
            val token = SharedPreferencesUtils.getToken(getApplication())

            // API 서비스를 사용하여 댓글 작성 요청을 실행
            val response = withContext(Dispatchers.IO) {
                apiService.postComment(boardId, "Bearer $token", commentDto).execute()
            }

            if (response.isSuccessful) {
                _commentResponse.postValue(response.body())
                fetchComments(boardId) // 댓글 작성 후 댓글 목록 다시 가져오기
            }
        }
    }

    // 댓글을 삭제하는 함수
    fun deleteComment(commentId: Long) {
        viewModelScope.launch {
            val token = SharedPreferencesUtils.getToken(getApplication())

            // API 서비스를 사용하여 댓글 삭제 요청을 실행
            val response = withContext(Dispatchers.IO) {
                apiService.deleteComment(commentId, "Bearer $token").execute()
            }
        }
    }
}
```

- CommentViewModel

- 이 ViewModel은 댓글 관련 기능을 제공하고 서버와 통신하는 데 사용된다.
- SERVER_URL: 서버의 기본 URL을 저장한다.
- apiService: Retrofit을 사용하여 서버 API와 통신하기 위한 인터페이스를 생성한다.
- _comments: MutableLiveData를 사용하여 게시판의 댓글 목록을 저장하고 화면에 표시될 때 업데이트된다.
- fetchComments(boardId: Long): 서버에서 특정 게시판의 댓글 목록을 가져와 _comments에

업데이트한다.

- _commentResponse: MutableLiveData를 사용하여 댓글 작성 결과를 저장하고, 댓글을 작성하거나 삭제한 후에 업데이트된다.
- postComment(boardId: Long, commentDto: CommentDto): 게시판에 댓글을 작성하는 함수로, 댓글 작성 결과를 _commentResponse에 업데이트하고, 작성한 댓글을 서버에 업로드한다.
- deleteComment(commentId: Long): 서버에서 특정 댓글을 삭제하는 함수이다. 댓글을 삭제한 후에 서버로부터 응답을 받는다.

CommentDto

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment

// CommentDto: 댓글 작성 시 필요한 데이터를 담는 데이터 클래스
data class CommentDto(
    val content: String // 댓글 내용을 저장하는 속성
)

package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment

// CommentListPageResponse: 서버에서 받아온 여러 댓글 목록 페이지 데이터를 담는 데이터 클래스
data class CommentListPageResponse(
    val content: List<CommentListResponse> // 댓글 목록 페이지에 있는 댓글 목록을 저장하는 리스트
)

package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.Dto.Comment

// CommentListResponse: 서버에서 받아온 개별 댓글 데이터를 담는 데이터 클래스
data class CommentListResponse(
    val commentId: Long,            // 댓글 고유 식별자
    val content: String,            // 댓글 내용
    val commentWriterName: String, // 댓글 작성자 이름
    val createdDate: String,        // 댓글 작성 일자
    val modifiedDate: String        // 댓글 수정 일자
)
```

기타 소스코드

MainActivity

```
import BoardDetailScreen
import BoardListScreen
import BoardWriteScreen
import LoginScreen
import RegisterScreen
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
```

```
import androidx.activity.viewModels
import androidx.navigation.compose.NavHost
import androidx.navigation.compose.composable
import androidx.navigation.compose.rememberNavController
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.BoardViewModel
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.CommentViewModel
import kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend.ViewModel.MemberViewModel

// 앱 화면의 다양한 섹션을 식별하기 위한 열거형 Screen 정의
enum class Screen(val route: String) {
    Login("login"),
    Register("register"),
    BoardList("boardList"),
    BoardDetail("boardDetail/{boardId}")
}

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        // ViewModels 초기화
        val memberViewModel: MemberViewModel by viewModels()
        val boardViewModel: BoardViewModel by viewModels()
        val commentViewModel: CommentViewModel by viewModels()

        // 액티비티 내용 설정
        setContent {
            // NavController 생성 및 초기화
            val navController = rememberNavController()

            // NavHost를 사용하여 앱 내비게이션을 정의
            NavHost(navController = navController, startDestination = Screen.Login.name) {
                composable(Screen.Login.route) {
                    // 로그인 화면 표시
                    LoginScreen(memberViewModel, navController)
                }
                composable(Screen.Register.route) {
                    // 회원가입 화면 표시
                    RegisterScreen(memberViewModel, navController)
                }
                composable(Screen.BoardList.route) {
                    // 게시글 목록 화면 표시
                    BoardListScreen(boardViewModel, navController)
                }
                composable(Screen.BoardDetail.route) { backStackEntry ->
```

```
                    // 게시글 상세 화면 표시
                    val boardId = backStackEntry.arguments?.getString("boardId")?.toLongOr
Null()
                    if (boardId != null) {
                        BoardDetailScreen(
                            boardViewModel = boardViewModel,
                            commentViewModel = commentViewModel,
                            boardId = boardId,
                            navController = navController
                        )
                    }
                }
                composable("boardWrite") {
                    // 게시글 작성 화면 표시
                    BoardWriteScreen(boardViewModel, navController)
                }
            }
        }
    }
}
```

-Screen 열거형 정의

- Screen 열거형은 앱의 다양한 화면을 구분하기 위한 열거형이다. 각 화면에 대한 라우트(route)를 정의하고 있다.

- MainActivity 클래스 정의

- ComponentActivity를 상속받아서 앱의 메인 액티비티를 정의한다.
- onCreate 메서드에서 액티비티가 생성될 때 초기화 작업을 수행한다.

- ViewModels 초기화

- MemberViewModel, BoardViewModel, CommentViewModel의 인스턴스를 초기화하고 by viewModels()를 사용하여 ViewModel을 가져온다. 이렇게 생성한 ViewModel 인스턴스는 화면 간 데이터 공유 및 비즈니스 로직을 처리하는 데 사용된다.

- setContent { ... } 블록

- setContent 함수를 사용하여 액티비티의 화면을 설정한다.

- NavHost를 사용한 앱 내비게이션 설정

- NavHost는 Compose Navigation을 설정하는 역할을 한다.
- startDestination 속성은 앱을 실행했을 때 처음으로 표시될 화면을 지정한다.

-composable { ... } 블록 (화면 정의)

- 각 화면을 composable 함수로 정의한다. 이 함수는 NavHost 내에서 해당 화면을 나타내며, 화면의 라우트(route)에 따라 다른 컴포넌트(화면)를 표시한다.
- 예를 들어, Screen.Login.route에 해당하는 화면은 LoginScreen으로 표시되고, Screen.BoardList.route에 해당하는 화면은 BoardListScreen으로 표시된다.
- 화면 간 데이터 공유 및 이동은 navController를 사용하여 처리한다.

NullOnEmptyConverterFactory

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend


import okhttp3.ResponseBody
```

```
import retrofit2.Converter
import retrofit2.Retrofit
import java.lang.reflect.Type

// Retrofit의 Converter.Factory 클래스를 상속하는 NullOnEmptyConverterFactory 클래스 정의
class NullOnEmptyConverterFactory : Converter.Factory() {

    // responseBodyConverter 메서드 재정의
    override fun responseBodyConverter(
        type: Type,
        annotations: Array<out Annotation>,
        retrofit: Retrofit
    ): Converter<ResponseBody, *>? {

        // 이전의 responseBodyConverter를 가져오기 위해 Retrofit의 nextResponseBodyConver
ter 메서드 사용
        val nextResponseBodyConverter =
            retrofit.nextResponseBodyConverter<Any>(this, type, annotations)

        // Converter<ResponseBody, Any>을 반환
        return Converter<ResponseBody, Any> { responseBody ->

            // HTTP 응답의 내용 길이가 0이 아닌 경우
            if (responseBody.contentLength() != 0L) {
                // 이전 컨버터를 호출하여 변환하고 결과 반환
                nextResponseBodyConverter.convert(responseBody)
            } else {
                // 내용이 비어있을 경우 null 반환
                null
            }
        }
    }
}
```

- NullOnEmptyConverterFactory

- 이 컨버터는 서버 응답의 내용이 비어있을 때 null을 반환하도록 동작하도록 구현되어 있다.
- NullOnEmptyConverterFactory 클래스: Retrofit의 Converter.Factory를 상속받아 컨버터를 정의하는 클래스이다.
- responseBodyConverter 메서드 재정의: Retrofit이 HTTP 응답의 내용을 변환하는데 사용하는 메서드를 재정의한다.
- nextResponseBodyConverter 가져오기: retrofit.nextResponseBodyConverter 메서드를 사용하여 이전에 등록된 다른 컨버터를 참조하는 nextResponseBodyConverter를 가져온다.
- Converter<ResponseBody, Any> 반환: 이 메서드는 Converter<ResponseBody, Any>를 반환한다. 이 컨버터는 HTTP 응답의 바디를 Any 형식으로 변환한다.
- HTTP 응답 바디의 길이 확인: it.contentLength() != 0L를 사용하여 HTTP 응답의 내용 길이가 0이 아닌 경우에만 다음 컨버터인 nextResponseBodyConverter를 호출하여 변환한다.

- 이전 컨버터를 사용한 변환: HTTP 응답 내용이 비어있지 않으면 이전의 responseBodyConverter를 사용하여 실제 변환 작업을 수행한다.
- HTTP 응답 바디가 비어있을 경우 null 반환: 만약 HTTP 응답의 내용이 비어있을 경우 null을 반환하며, 이로써 빈 응답을 처리할 수 있게 된다.

SharedPreferencesUtils

```
package kr.ac.kumoh.ce.s20190633.todayfitcomplete_frontend

import android.content.Context

// SharedPreferences를 사용하여 데이터를 저장하고 검색하는 유틸리티 클래스
object SharedPreferencesUtils {
    // SharedPreferences 파일에 대한 고유한 키
    private const val PREFERENCES_FILE_KEY = "kr.ac.kumoh.ce.s20190633.todayfitcomplete
_frontend.preferences"

    // 사용자 토큰을 저장하기 위한 키
    private const val TOKEN_KEY = "token"

    // 사용자 이메일을 저장하기 위한 키
    private const val EMAIL_KEY = "email"

    // 사용자 토큰을 저장하는 함수
    fun saveToken(context: Context, token: String) {
        // SharedPreferences 파일을 가져옴
        val sharedPref = context.getSharedPreferences(PREFERENCES_FILE_KEY, Context.MO
DE_PRIVATE)
        // SharedPreferences 편집기를 얻어 토큰을 저장
        with(sharedPref.edit()) {
            putString(TOKEN_KEY, token)
            apply() // 변경사항을 즉시 저장
        }
    }

    // 사용자 토큰을 검색하는 함수
    fun getToken(context: Context): String? {
        val sharedPref = context.getSharedPreferences(PREFERENCES_FILE_KEY, Context.MO
DE_PRIVATE)
        // 저장된 토큰을 반환하며, 없으면 null을 반환
        return sharedPref.getString(TOKEN_KEY, null)
    }

    // 사용자 이메일을 저장하는 함수
    fun saveEmail(context: Context, email: String) {
        val sharedPreferences = context.getSharedPreferences(PREFERENCES_FILE_KEY, Co
```

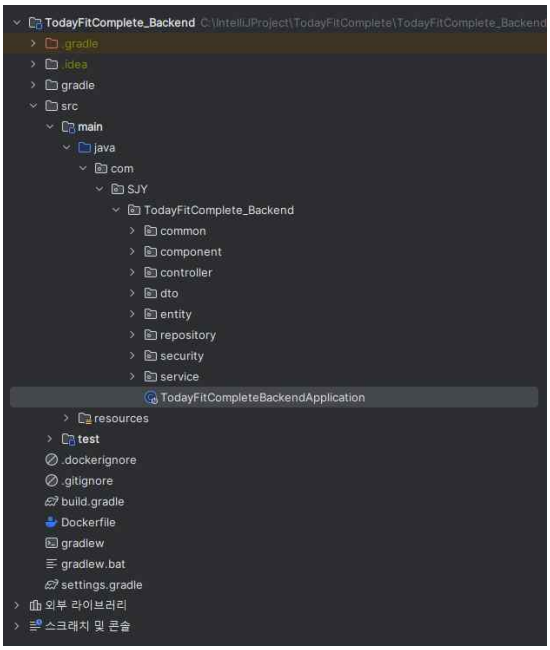
```
ntext.MODE_PRIVATE)
        // SharedPreferences 편집기를 얻어 이메일을 저장
        with(sharedPreferences.edit()) {
            putString(EMAIL_KEY, email)
            apply() // 변경사항을 즉시 저장
        }
    }

    // 사용자 이메일을 검색하는 함수
    fun getEmail(context: Context): String? {
        val sharedPreferences = context.getSharedPreferences(PREFERENCES_FILE_KEY, Co
ntext.MODE_PRIVATE)
        // 저장된 이메일을 반환하며, 없으면 null을 반환
        return sharedPreferences.getString(EMAIL_KEY, null)
    }
}
```

- SharedPreferencesUtils

- 이 클래스는 사용자 토큰 및 이메일과 같은 중요한 데이터를 저장하고 가져오는데 사용된다.
- SharedPreferencesUtils 객체는 싱글톤 패턴으로 구현되어 있으므로 앱 내에서 하나의 인스턴스만 사용된다.
- PREFERENCES_FILE_KEY: SharedPreferences에 사용할 파일 키로, 앱 내에서 데이터를 저장하는 데 사용된다.
- TOKEN_KEY: 사용자 토큰을 저장하고 검색하는 데 사용된다.
- EMAIL_KEY: 사용자 이메일을 저장하고 검색하는 데 사용된다.
- saveToken(context: Context, token: String): 사용자 토큰을 SharedPreferences에 저장하는 함수이다. 주어진 token을 지정된 context의 SharedPreferences에 저장한다.
- getToken(context: Context): String?: SharedPreferences에서 사용자 토큰을 검색하는 함수이다. 사용자 토큰을 반환하며, 저장된 값이 없는 경우 null을 반환한다.
- saveEmail(context: Context, email: String): 사용자 이메일을 SharedPreferences에 저장하는 함수이다. 주어진 email을 지정된 context의 SharedPreferences에 저장한다.
- getEmail(context: Context): String?: SharedPreferences에서 사용자 이메일을 검색하는 함수이다. 사용자 이메일을 반환하며, 저장된 값이 없는 경우 null을 반환한다.

4.2 백엔드 (스프링)

스프링을 이용한 백엔드 개발에 대해 간략하게 서술한다.



수업 시간에 다루지 않은 내용이므로, 본 보고서에서는 프론트엔드(안드로이드)와 통신하는 Controller 계층에 대해서만 집중적으로 다루겠다.

Controller 계층은 주로 클라이언트 요청을 받고, 적절한 서비스 로직을 호출하여 결과를 반환하는 역할을 한다. 이를 통해 프론트엔드(안드로이드)와 백엔드(스프링) 간의 효율적인 데이터 교환 및 처리가 가능한다.

MemberController

```
package com.SJY.TodayFitComplete_Backend.controller;

import com.SJY.TodayFitComplete_Backend.dto.request.member.MemberLoginDto;
import com.SJY.TodayFitComplete_Backend.dto.request.member.MemberRegisterDto;
import com.SJY.TodayFitComplete_Backend.dto.request.member.MemberUpdateDto;
import com.SJY.TodayFitComplete_Backend.dto.response.member.MemberResponse;
import com.SJY.TodayFitComplete_Backend.dto.response.member.MemberTokenResponse;
import com.SJY.TodayFitComplete_Backend.entity.Member;
import com.SJY.TodayFitComplete_Backend.service.MemberService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

import java.util.Map;

@RestController
@RequestMapping("/user")
@RequiredArgsConstructor
public class MemberController {
```

```
    private final MemberService memberService;

    /**
     * 회원의 이메일 중복을 검사한다.
     *
     * @param email 검사할 이메일
     * @return 검사 결과
     */
    @GetMapping("/checkId")
    public ResponseEntity<?> checkIdDuplicate(@RequestParam String email) {
        memberService.checkIdDuplicate(email);
        return ResponseEntity.ok().build();
    }

    /**
     * 새로운 회원을 등록한다.
     *
     * @param memberRegisterDTO 회원 등록 정보
     * @return 등록된 회원 정보
     */
    @PostMapping("/register")
    public ResponseEntity<MemberResponse> register(@RequestBody MemberRegisterDto
memberRegisterDTO) {
        MemberResponse registeredMember = memberService.register(memberRegisterDTO);
        return ResponseEntity.status(HttpStatus.CREATED).body(registeredMember);
    }

    /**
     * 회원 로그인을 처리한다.
     *
     * @param memberLoginDTO 로그인 정보
     * @return 로그인 성공 시 토큰 및 사용자 정보
     */
    @PostMapping("/login")
    public ResponseEntity<MemberTokenResponse> login(@RequestBody MemberLoginDto
memberLoginDTO) {
        MemberTokenResponse loginResult = memberService.login(memberLoginDTO);
        return ResponseEntity.ok().header(loginResult.getToken()).body(loginResult);
    }
}
```

- MemberController

- 이 컨트롤러는 사용자 회원가입, 로그인 및 이메일 중복 검사와 같은 회원 관련 기능을 처리한다.
- @RestController 및 @RequestMapping("/user"): 이 클래스는 RESTful API를 처리하는 컨트롤러임을 나타내며, /user 엔드포인트를 기본 경로로 정의한다.

- 생성자 주입 (@RequiredArgsConstructor): MemberController 클래스는 MemberService를 의존성 주입(DI)을 통해 사용한다. Lombok의 @RequiredArgsConstructor 어노테이션을 사용하여 생성자를 자동으로 생성한다.
- checkIdDuplicate 메서드: GET 요청을 처리하며, 클라이언트가 제공한 이메일 중복 여부를 확인한다. 중복되지 않는 경우 200 OK 응답을 반환한다.
- register 메서드: POST 요청을 처리하며, 새로운 회원을 등록한다. 클라이언트가 제공한 회원 정보를 받아 회원을 등록하고, 등록된 회원 정보와 201 Created 상태 코드를 응답으로 반환한다.
- login 메서드: POST 요청을 처리하며, 회원 로그인을 처리한다. 클라이언트가 제공한 로그인 정보를 받아 로그인을 시도하고, 성공하면 회원 토큰과 사용자 정보를 응답으로 반환한다.

BoardController

```
package com.SJY.TodayFitComplete_Backend.controller;

import com.SJY.TodayFitComplete_Backend.dto.request.board.BoardUpdateDto;
import com.SJY.TodayFitComplete_Backend.dto.request.board.BoardWriteDto;
import com.SJY.TodayFitComplete_Backend.dto.request.board.SearchData;
import com.SJY.TodayFitComplete_Backend.dto.response.board.BoardDetailsResponse;
import com.SJY.TodayFitComplete_Backend.dto.response.board.BoardListResponse;
import com.SJY.TodayFitComplete_Backend.dto.response.board.BoardWriteResponse;
import com.SJY.TodayFitComplete_Backend.entity.Member;
import com.SJY.TodayFitComplete_Backend.service.BoardService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.web.PageableDefault;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/board")
@RequiredArgsConstructor
@Slf4j
public class BoardController {

    private final BoardService boardService;

    /**
     * 게시글 목록을 페이지네이션으로 반환한다.
     *
```

```
     * @param pageable 페이징 정보 (크기, 정렬 등)
     * @return 페이지네이션된 게시글 목록
     */
    @GetMapping("/list")
    public ResponseEntity<Page<BoardListResponse>> boardList(
            @PageableDefault(size = 10, sort = "id", direction = Sort.Direction.DESC)
Pageable pageable) {
        Page<BoardListResponse> listDTO = boardService.getAllBoards(pageable);
        return ResponseEntity.status(HttpStatus.OK).body(listDTO);
    }



    /**
     * 새로운 게시글을 작성한다.
     *
     * @param boardDTO 게시글 작성 데이터
     * @param member 인증된 사용자 정보
     * @return 생성된 게시글 정보
     */
    @PostMapping("/write")
    public ResponseEntity<BoardWriteResponse> write(
            @RequestBody BoardWriteDto boardDTO,
            @AuthenticationPrincipal Member member) {
        BoardWriteResponse saveBoardDTO = boardService.write(boardDTO, member);
        return ResponseEntity.status(HttpStatus.CREATED).body(saveBoardDTO);
    }

    /**
     * 특정 게시글의 상세 정보를 반환한다.
     *
     * @param boardId 게시글 ID
     * @return 게시글 상세 정보
     */
    @GetMapping("/{boardId}")
    public ResponseEntity<BoardDetailsResponse> detail(@PathVariable("boardId") Long
boardId) {
        BoardDetailsResponse findBoardDTO = boardService.detail(boardId);
        return ResponseEntity.status(HttpStatus.OK).body(findBoardDTO);
    }

    /**
     * 특정 게시글을 삭제한다.
     *
     * @param boardId 삭제할 게시글 ID
     * @return 삭제된 게시글 ID
```

```
     */
    @DeleteMapping("/{boardId}/delete")
    public ResponseEntity<Long> delete(@PathVariable Long boardId) {
        boardService.delete(boardId);
        return ResponseEntity.status(HttpStatus.OK).build();
    }
}
```

- BoardController

- 이 컨트롤러는 게시글 관련 기능을 처리한다.
- @RestController 및 @RequestMapping("/board"): 이 클래스는 RESTful API를 처리하는 컨트롤러임을 나타내며, /board 엔드포인트를 기본 경로로 정의한다.
- 생성자 주입 (@RequiredArgsConstructor): BoardController 클래스는 BoardService를 의존성 주입(DI)을 통해 사용한다. Lombok의 @RequiredArgsConstructor 어노테이션을 사용하여 생성자를 자동으로 생성한다.
- boardList 메서드: GET 요청을 처리하며, 게시글 목록을 페이지네이션으로 반환한다. 페이지 크기 및 정렬 방식을 포함하는 pageable 매개변수를 받아와 게시글 목록을 조회하고, 200 OK 응답과 함께 페이지네이션된 게시글 목록을 반환한다.
- write 메서드: POST 요청을 처리하며, 새로운 게시글을 작성한다. 클라이언트가 제공한 게시글 작성 데이터를 받아와 게시글을 작성하고, 201 Created 상태 코드와 함께 생성된 게시글 정보를 응답으로 반환한다. 또한, 현재 인증된 사용자 정보(@AuthenticationPrincipal Member member)를 사용하여 게시글 작성자를 지정한다.
- detail 메서드: GET 요청을 처리하며, 특정 게시글의 상세 정보를 반환한다. 클라이언트가 요청한 게시글 ID(boardId)를 받아와 해당 게시글의 상세 정보를 조회하고, 200 OK 응답과 함께 게시글 상세 정보를 반환한다.
- delete 메서드: DELETE 요청을 처리하며, 특정 게시글을 삭제한다. 클라이언트가 요청한 게시글 ID(boardId)를 받아와 해당 게시글을 삭제하고, 200 OK 응답을 반환한다.

FileController

```
package com.SJY.TodayFitComplete_Backend.controller;

import com.SJY.TodayFitComplete_Backend.dto.response.file.FileDownloadResponse;
import com.SJY.TodayFitComplete_Backend.dto.response.file.FileUploadResponse;
import com.SJY.TodayFitComplete_Backend.service.FileService;
import lombok.RequiredArgsConstructor;
import org.springframework.core.io.ByteArrayResource;
import org.springframework.core.io.Resource;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
```

```
import java.util.List;

@RestController
@RequestMapping("/board/{boardId}/file")
@RequiredArgsConstructor
public class FileController {
    private final FileService fileService;

    /**
     * 특정 게시글에 파일을 업로드한다.
     *
     * @param boardId 게시글 ID
     * @param files 업로드할 파일 목록
     * @return 업로드된 파일 정보 목록
     * @throws IOException 파일 업로드 과정에서 발생하는 예외
     */
    @PostMapping("/upload")
    public ResponseEntity<List<FileUploadResponse>> upload(@PathVariable Long boardId,
                                                           @RequestParam("file") List<Multipa
rtFile> files) throws IOException {
        List<FileUploadResponse> uploadedFiles = fileService.upload(boardId, files);
        return ResponseEntity.status(HttpStatus.CREATED).body(uploadedFiles);
    }

}
```

- FileController

- 이 컨트롤러는 특정 게시글과 관련된 파일 업로드 기능을 처리한다.
- @RestController 및 @RequestMapping("/board/{boardId}/file"): 이 클래스는 RESTful API를 처리하는 컨트롤러임을 나타내며, /board/{boardId}/file 엔드포인트를 기본 경로로 정의한다. {boardId}는 동적으로 게시글 ID를 받아오는 부분으로, 실제 요청에서 게시글의 ID 값을 지정한다.
- 생성자 주입 (@RequiredArgsConstructor): FileController 클래스는 FileService를 의존성 주입(DI)을 통해 사용한다. Lombok의 @RequiredArgsConstructor 어노테이션을 사용하여 생성자를 자동으로 생성한다.
- upload 메서드: POST 요청을 처리하며, 특정 게시글에 파일을 업로드한다. 클라이언트가 요청한 게시글 ID(boardId)와 업로드할 파일 목록(files)을 받아와 해당 게시글에 파일을 업로드한다. 업로드된 파일의 정보(FileUploadResponse) 목록을 받아와 201 Created 상태 코드와 함께 업로드된 파일 정보 목록을 응답으로 반환한다. 이때, 업로드된 파일의 정보는 클라이언트에서 파일을 다운로드하거나 표시할 때 사용된다.

CommentController

```
package com.SJY.TodayFitComplete_Backend.controller;

import com.SJY.TodayFitComplete_Backend.dto.request.comment.CommentDto;
```

```
import com.SJY.TodayFitComplete_Backend.dto.response.comment.CommentResponse;
import com.SJY.TodayFitComplete_Backend.entity.Member;
import com.SJY.TodayFitComplete_Backend.service.CommentService;
import lombok.RequiredArgsConstructor;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;
import org.springframework.data.web.PageableDefault;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/board/{boardId}/comment")
@RequiredArgsConstructor
public class CommentController {

    private final CommentService commentService;

    /**
     * 특정 게시글에 대한 모든 댓글을 페이지네이션 형태로 반환한다.
     *
     * @param boardId 게시글 ID
     * @param pageable 페이징 정보
     * @return 페이지네이션된 댓글 목록
     */
    @GetMapping("/list")
    public ResponseEntity<Page<CommentResponse>> commentList(
            @PathVariable Long boardId,
            @PageableDefault(size = 5, sort = "id", direction = Sort.Direction.DESC) Pageable
pageable) {

        Page<CommentResponse> commentList = commentService.getAllComments(pageable,
boardId);
        return ResponseEntity.status(HttpStatus.OK).body(commentList);
    }

    /**
     * 새로운 댓글을 작성한다.
     *
     * @param member 인증된 사용자 정보
     * @param boardId 게시글 ID
     * @param commentDto 댓글 작성 데이터
     * @return 생성된 댓글 정보
```

```
     */
    @PostMapping("/write")
    public ResponseEntity<CommentResponse> write(
            @AuthenticationPrincipal Member member,
            @PathVariable Long boardId,
            @RequestBody CommentDto commentDto) {

        CommentResponse saveCommentDTO = commentService.write(boardId, member, commentDto);
        return ResponseEntity.status(HttpStatus.CREATED).body(saveCommentDTO);
    }

    /**
     * 특정 댓글을 삭제한다.
     *
     * @param commentId 삭제할 댓글 ID
     * @return 삭제된 댓글 ID
     */
    @DeleteMapping("/delete/{commentId}")
    public ResponseEntity<Long> delete(@PathVariable Long commentId) {

        commentService.delete(commentId);
        return ResponseEntity.status(HttpStatus.OK).build();
    }
}
```

- CommentController

- 이 컨트롤러는 특정 게시글에 대한 댓글 관련 기능을 처리한다.
  @RestController 및 @RequestMapping("/board/{boardId}/comment"): 이 클래스는 RESTful API를 처리하는 컨트롤러임을 나타내며, /board/{boardId}/comment 엔드포인트를 기본 경로로 정의한다. {boardId}는 동적으로 게시글 ID를 받아오는 부분으로, 실제 요청에서 게시글의 ID 값을 지정한다.
- 생성자 주입 (@RequiredArgsConstructor): CommentController 클래스는 CommentService를 의존성 주입(DI)을 통해 사용한다. Lombok의 @RequiredArgsConstructor 어노테이션을 사용하여 생성자를 자동으로 생성한다.
- commentList 메서드: GET 요청을 처리하며, 특정 게시글에 대한 모든 댓글을 페이지네이션 형태로 반환한다. 게시글 ID(boardId)와 페이징 정보(pageable)를 받아와 해당 게시글에 대한 페이지네이션된 댓글 목록을 응답으로 반환한다.
- write 메서드: POST 요청을 처리하며, 새로운 댓글을 작성한다. 인증된 사용자 정보(member), 게시글 ID(boardId), 그리고 댓글 작성 데이터(commentDto)를 받아와 해당 게시글에 댓글을 작성하고, 생성된 댓글 정보를 응답으로 반환한다.
- delete 메서드: DELETE 요청을 처리하며, 특정 댓글을 삭제한다. 삭제할 댓글의 ID(commentId)를 받아와 해당 댓글을 삭제하고, 삭제된 댓글의 ID를 응답으로 반환한다.

DockerFile

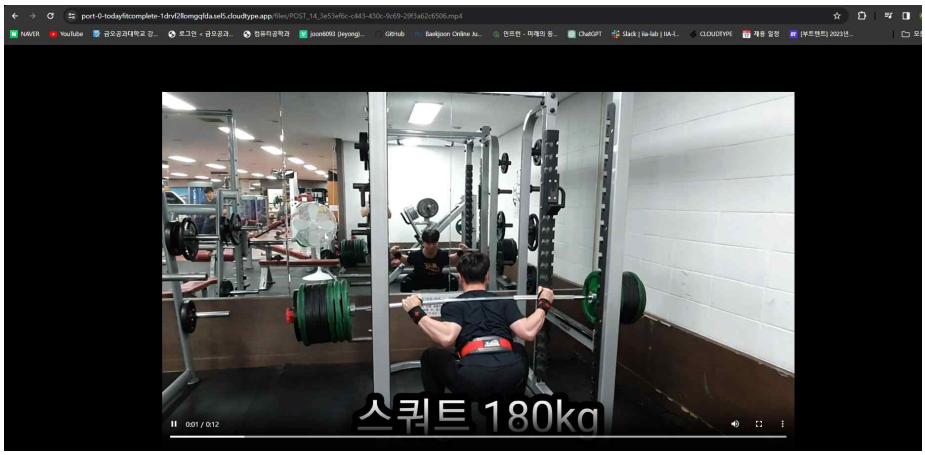
```
# Jar 파일 빌드
FROM eclipse-temurin:17 as builder

# 소스 코드와Gradle Wrapper 복사
COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src

# Gradle Wrapper 실행 권한 부여 및 애플리케이션 빌드
RUN chmod +x ./gradlew && ./gradlew bootJar

# Jar 실행
FROM eclipse-temurin:17-jre as runtime

# 사용자 및 그룹 생성
RUN addgroup --system --gid 1000 worker && \
adduser --system --uid 1000 --ingroup worker --disabled-password worker

# 파일 저장을 위한 디렉토리 생성 및 권한 설정
RUN mkdir -p /app/files && \
chown worker:worker /app/files

# 빌드된Jar 파일 복사
COPY --from=builder /build/libs/*.jar app.jar

# 사용자 전환
USER worker:worker

# 환경변수 설정
ENV PROFILE ${PROFILE}

# 포트8080 열기
EXPOSE 8080

# 애플리케이션 실행
ENTRYPOINT ["java", "-Dspring.profiles.active=${PROFILE}", "-jar", "/app.jar"]
```

- 클라우드 타입 DockerFile을 이용한 배포

클라우드 타입 서버를 사용하여 Spring 프로젝트를 배포할 때, 자동으로 생성되는 DockerFile을 사용하지 않고, 이미지와 동영상을 저장할 수 있는 사용자 정의 DockerFile을 생성하는 방법에 대해 설명하겠다.

```java
package com.SJY.TodayFitComplete_Backend.common;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.EnableWebMvc;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

 joon6093
@Configuration
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer {

    2개 사용 위치  joon6093
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler( ...pathPatterns: "/files/**")
                .addResourceLocations("file:/app/files/");
    }
}
```

해당 코드를 이용해서 폴더에 저장된 이미지 또는 동영상을 제공해준다.

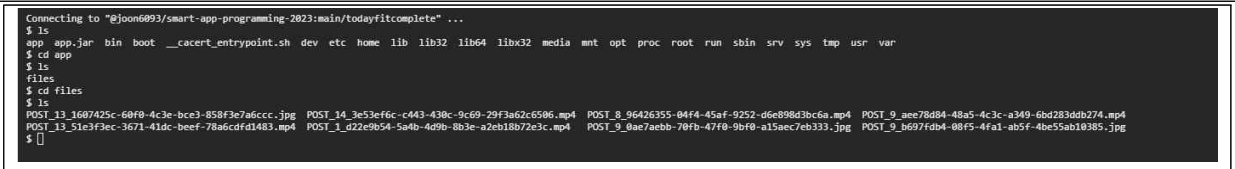


해당 기능을 사용하기위해 클라우드 타입 서버에서 이미지와 동영상을 저장할 경로 지정이 필요하다. Spring 프로젝트 자체를 배포할 경우 기본 DockerFile 자동으로 사용 되고 해당 경우에는 이미지와 동영상을 저장할 경로를 설정할 수 없으므로, 사용자가 직접 DockerFile을 작성해야 한다. 이를 위해 DockerFile에서 파일 저장을 위한 디렉토리를 생성하고 권한을 설정하는 명령어를 포함시킨다. 예를 들어, /app/files 디렉토리를 생성하고 이 디렉토리의 소유권을 worker 사용자에게 할당한다. 이는 다음과 같은 명령어로 이루어진다.

```
# 파일 저장을 위한 디렉토리 생성 및 권한 설정
RUN mkdir -p /app/files && \
chown worker:worker /app/files
```

사용자 정의 DockerFile을 작성한 후, Spring 프로젝트를 이 DockerFile을 이용하여 배포한다. 이렇게 하면 클라우드 타입에서 자동으로 생성하는 DockerFile 대신 사용자가 정의한 DockerFile이 사용된다.

배포 후, 클라우드 타입 서버의 터미널을 통해 /app/files 디렉토리를 확인한다. 이 디렉토리에는 사용자가 업로드한 이미지와 동영상이 저장되어 있는 것을 볼 수 있다.



이 방법을 사용하면 클라우드 타입 서버에서 자동으로 생성되는 DockerFile의 제한을 넘어서, 필요한 파일 저장 경로와 권한을 사용자가 직접 설정할 수 있게 된다. 이로 인해 보다 유연한 파일 관리와 배포가 가능해진다.

5. 결 론

- 개발 내용 및 실험 결과 요약

- 개발된 안드로이드 애플리케이션은 사용자 친화적인 인터페이스로, 운동 인증 및 커뮤니티 상호작용을 중심으로 설계되었다.
- 백엔드는 Spring Framework를 사용하여 구현되었으며, RESTful API를 통해 프론트엔드와 통신한다.
- MariaDB를 데이터베이스 관리 시스템으로 사용하였으며, 사용자 데이터, 운동 인증 기록 등의 정보를 관리한다.
- 전체 시스템은 클라우드 인프라를 활용하여 배포되었다. 이는 확장성, 유연성 및 접근성을 크게 향상시킨다.
- 특히, 백엔드의 경우 Dockerfile을 이용한 컨테이너화를 통해 배포되었으며, 이는 배포 과정을 표준화하고, 클라우드 환경에서의 실행을 보다 효율적으로 만들었다.

- 향후 개선 과제

- 사용자 경험 향상: 사용자 인터페이스와 사용자 경험을 지속적으로 개선하여 더 직관적이고 효율적인 사용자 상호작용을 제공한다.
- 플랫폼 확장: 앱의 접근성 및 호환성을 높이기 위해 다양한 기기 및 운영 체제에서도 사용할 수 있도록 플랫폼을 확장한다.

- 기술적, 사회적, 경제적 파급 효과 및 기대 효과

- 기술적 효과: 디지털 헬스케어 및 소셜 네트워킹 기술의 결합을 통해 사용자가 기술을 통해 건강한 생활을 유지할 수 있는 새로운 방법을 제공한다.
- 사회적 효과: 커뮤니티 기반의 운동 인증 시스템은 사람들이 건강한 생활 방식을 채택하고 유지하는 데 도움을 주며, 사회적 연결감 및 긍정적인 상호작용을 증진시킨다.
- 경제적 효과: 건강한 생활 습관을 통해 장기적으로 의료 비용 절감 및 생산성 향상에 기여할 수 있다. 또한, 앱의 확장 및 개선을 통한 추가적인 수익 창출 가능성도 존재한다.

6. 자체 분석과 평가

- 수행 과정과 결과에 대한 개인적인 생각 및 평가

이번 프로젝트를 수행하며 프론트엔드 개발, 백엔드 시스템 구축 및 데이터베이스 관리에 대한 깊은 이해와 경험을 쌓았다. 특히, Android 앱 개발과 Spring 기반 백엔드의 연동, 그리고 MariaDB 데이터베이스의 효율적 사용에 대한 중요성을 깨달았다. 클라우드 환경에서의 배포 과정은 도전적이었지만, 이를 통해 시스템의 확장성과 유지 보수성에 대한 중요한 경험을 얻었다.

프로젝트 결과에 대해 매우 만족하고 있다. 비록 시간의 제약으로 인해 모든 기능을 완벽하게 구현하지는 못했지만, 기본적인 기능과 구조는 성공적으로 구축했다. 특히 백엔드의 Dockerfile을 활용한 배포 방식은 효율적이었으며, 이를 통해 더욱 안정적인 서비스 제공이 가능했다.

- 과제를 수행하면서 어려웠던 점과 그 문제를 해결한 방법

프로젝트 수행 중 가장 큰 도전은 백엔드 개발 및 클라우드 환경에서의 데이터베이스 연동이었다. Spring Framework와 MariaDB의 연동에서 발생하는 다양한 문제들을 해결하기 위해 공식 문서, 온라인 커뮤니티를 참고하여서 해결했다.

또한, Docker를 사용한 백엔드 배포 과정에서 발생한 문제들을 해결하기 위해 Docker 공식 문서와 관련 커뮤니티에서 제공하는 다양한 자료를 참고했다. 이러한 경험들은 실무적인 문제 해결 능력을 키우는 데 큰 도움이 되었다.

이러한 과정을 통해 얻은 경험과 지식은 앞으로의 개발 경력에 큰 자산이 될 것이며, 향후 프로젝트에서도 이러한 학습을 바탕으로 더 나은 결과를 도출할 수 있을 것으로 기대한다.

7. 부록

- 개발 일정

- 2023/12/01 백엔드 구현 시작
- 2023/12/04 백엔드 구현 완료
- 2023/12/05 백엔드 클라우드 타입 배포 및 프론트엔드 구현 시작
- 2023/12/07 프론트 엔드 구현 완료
- 2023/12/09 보고서 초안 작성
- 2023/12/21 보고서 기술 완료