

3

Chapter 3. API URL의 설계 & 프로젝트 세팅 (2)

 학습 목표

 잠깐 ! 스터디 인증샷은 찍으셨나요? 

 3주차 주제

 3주차 본문

API란? 

REST API

API Endpoint

HTTP 메소드

RESTful API Endpoint의 설계

회원 가입, 로그인, 탈퇴

 회원 가입은 유저 한 명이 생기는 건데, POST /users/id 아닌가요?

 리소스 간 연관 관계가 있는 경우

 N : M 관계는?

세부적인 API 설계

 회원 가입이란 , 회원 정보를 새로 저장하는 건데, 무슨 정보를 저장해?

Path Variable

Query String

Request Body

Request Header

API 설계 예시

API Endpoint

Request Body

Request Header

Query String

프로젝트 세팅 - Node.js

프로젝트 세팅 - Spring boot

 핵심 키워드

 학습 후기

 스터디 진행 방법

 실습 체크리스트

 실습 인증

 미션

- 👉 미션 기록
- ⚡ 트러블 슈팅
- 🤔 참고 자료

학습 목표

1. RESTful한 API 설계를 익힌다.
2. Node.js, Spring boot의 프로젝트 세팅을 한다.

잠깐 ! 스터디 인증샷은 찍으셨나요?

* 스터디리더께서 대표로 매 주차마다 한 장 남겨주시면 좋겠습니다! 🤖💕
(사진을 저장해서 이미지 임베드를 하셔도 좋고, 복사+붙여넣기해서 넣어주셔도 좋습니다!)

3주차 주제

다들 2주차 미션은 잘 완료 하셨나요? 저 같은 경우는 밑에와 같이 작성을 하였습니다 (꼭 2주차 미션 후에 열람해주세요!) (2주차 상단에 예시로 드린 ERD로 하였고 모두 ERD가 다르니 참고만 해주세요!)

😊 2주차 기본 미션 답안지 및 테스트용 데모데이터

1주차에서 ERD를 설계했었죠, 그 후에 필요한 것은 무엇일까요?

바로 **API를 큰 틀에서 설계를 하는 것**입니다.

API의 세부적인 사항은 처음에 다 정하기는 어렵지만,
데이터베이스와 마찬가지로
큰 틀을 잡아두는 것은 가능합니다.

그리고 API 설계 자체가 크게 변하지 않기도 합니다.

API 설계 시 필요한 것은 아래와 같습니다.

1. **API End point의 설계**
2. **요청 데이터, 응답 데이터의 설계**

그리고 위의 정보를 문서화해서 **프론트 개발자가 API를 사용하기 쉽도록 돕는 문서**를

API 명세서라고 부릅니다.

이번 주는 API 설계를 다루고 프로젝트 세팅을 할 것입니다.



3주차 본문

API란? 🤔

Application Programming Interface (API)

API라는 단어를 많이 들어봤지만, 무슨 뜻인지 정확히 감이 오는 단어는 아닙니다.

그리고 많은 분들이 API를 클라이언트와 서버 간 통신에 필요한 것 정도로만 이해를 하고 있죠.

그러나 **API**는 더 넓은 의미를 가집니다.



Application을 Programming 할 때 사용되는 인터페이스

인터페이스는 일상생활에서 많이 접할 수 있습니다.

UI도 User Interface의 약자이죠.

인터페이스는 간단히 말해서,

어려운 것은 감추고 보다 쉽게 상호작용 할 수 있도록 해주는 것들을 의미합니다.



**API는 애플리케이션을 프로그래밍 할 때,
보다 쉽게 할 수 있도록 해주는 도구들을 의미합니다.**

예시로 Js에서 **console.log**

python에서 **print**

모두 콘솔에 출력을 해주는 함수잖아요?

왜, 어떻게 콘솔에 출력을 해줄까요?

이런 것들을 API라고 부릅니다.

API 라는 단어에 **추상화, 캡슐화의 개념**이 들어간 것도 느껴지죠.

REST API

REST는 Representational State Transfer의 약자로,

HTTP를 기반으로 하는 웹 서비스 아키텍처를 의미하며

HTTP 메소드와 자원을 이용해 서로 간의 통신을 주고받는 방법입니다.

API Endpoint

REST API에서 API Endpoint는 해당 API를 호출하기 위한 HTTP 메소드, 그리고 URL을 포함합니다.

HTTP 메소드

HTTP 메소드는 REST 방식으로 통신 할 때 필요한 작업을 표시하는 방법으로

여러 가지가 있지만 아래의 5가지만 소개를 하겠습니다.

그리고 아래의 5개 메소드는 CRUD(생성, 조회, 갱신, 삭제) 4가지에 대응이 됩니다.

1. GET : 조회
2. POST : 생성
3. PUT : 갱신(전체)
4. PATCH : 갱신(일부)
5. DELETE : 삭제

위의 5개 메소드 중 **POST**는 새로운 자원의 생성도 있지만,
클라이언트가
특정 정보를 서버로 넘기고 그에 대한 처리를 요청 하는 것을
전부 POST로 처리 가능합니다.

RESTful API Endpoint의 설계

이제 RESTful한 API의 설계를 위한 규칙을 알아보시다.

RESTful한 API의 Endpoint는 **아래의 규칙**에 따라 설계가 가능합니다.

1. URI에 동사가 포함이 되어선 안된다.
2. URI에서 단어의 구분이 필요한 경우 -(하이픈)을 이용한다.
3. 자원은 기본적으로 복수형으로 표현한다.
4. 단 하나의 자원을 명시적으로 표현을 하기 위해서는 **/users/id**와 같이 식별 값을 추가로 사용한다.
5. **자원 간 연관 관계가 있을 경우 이를 URI에 표현한다.**

5번은 추가적인 설명이 필요하나 나머지는 쉽게 감이 올 것입니다.

이제 예시를 보면서 REST API 설계를 어떻게 하는지 살펴봅시다.

회원 가입, 로그인, 탈퇴

사실 API URL Endpoint 설계가 **가장 난해한 경우**인데요,
우선 로그인의 경우가 가장 애매합니다.

먼저 로그인의 주체가 되는 대상은 사용자이므로
url 상에 users가 들어가는 것은 쉽게 감이 옵니다.
그런데 그 이후가 난해하죠. 🤔

우선 클라이언트가 로그인에 필요한 정보를 서버로 넘겨 로그인에 대한 처리를 요청하는 것
이니

POST로 설계를 해야겠죠.

그러나 여기서 **POST /users** 이렇게 하면 될까요?

회원 가입 역시 새로운 사용자의 생성이기 때문에 **POST /users**로 표현이 가능합니다.
따라서 회원 가입의 경우가 있기 때문에 로그인을 **POST /users**로 표현하기가 힘들죠.

이런 경우에는 너무 타이트하게 **RESTful** 한 API 설계를 따르려고 하지 않아도 됩니다.



POST /users/login

이런 형식으로 만들어도 괜찮습니다.

위처럼 HTTP 메소드, 그리고 필요한 자원에 대한 명시를 한 URI를 합쳐 API Endpoint 라고 하며,
웹 서비스를 하는 서버의 도메인 주소가
<https://umc.com>일 경우

POST https://umc.com/users/login 이렇게 요청을 할 경우 로그인이 되는 것이죠.

이제 다음으로 회원 가입과 탈퇴를 설계 해봅시다.

회원 가입의 경우는 **POST /users** 를 해도 됩니다.

허나, 이것 역시 요구 사항에 따라 달라지는 것으로,
기능 요구 사항에서 회원 가입 외에 사용자가 새로 추가가 되는 경우가 있다면
해당 API와 구분을 해야 하겠죠.

위와 같은 경우를 제외하면 **POST /users**로 회원 가입을 명시적으로 표현이 가능합니다.

🤔 **회원 가입은 유저 한 명이 생기는 건데, POST /users/id 아닌가요?**

POST /users/id의 형태로 설계를 하려면, 해당 유저를 식별할 값이 존재해야 하는데

회원 가입 이후에서야 해당 유저가 생성이 되기에, 유저를 식별할 값이 API 호출 시점에는 없죠.

따라서 이런 경우는 POST /users/id와 같은 설계가 불가능합니다.

다음으로 회원 탈퇴를 생각해봅시다.

회원 탈퇴는 러프하게 생각하면 아래처럼 설계가 가능하다고 생각할 수 있습니다.

DELETE /users

그러나 요구 사항이

**회원 탈퇴 시 곧바로 데이터베이스에서 삭제가 아니라, 비활성 계정으로 만들고,
추후 계정 복구를 고려**

한다면 **DELETE로 설계를 하면 안되죠.**

이런 경우 데이터베이스에서 사용자 테이블에 status를 active에서 inactive로 변경하는 것
이기에

일부 수정인 PATCH를 활용해야 합니다.

PATCH /users

그런데 이 경우 **회원 정보 수정 API까지 고려해서 설계를** 해야 합니다.

회원 정보 수정 API 역시 닉네임 등 **일부 정보만 수정하는 것**이기에

PATCH를 사용하며, **PATCH /users** 로 설계가 가능하겠죠.

이런 상황에서 저 같은 경우,

회원 정보 수정은 PATCH /users/id 로 특정 유저의 정보 수정임을 명시적으로 표현하고,

회원 탈퇴는 PATCH /users 로 표현하기도 합니다.

하나 프론트엔드 개발자 분들이 저런 설계를 헛갈려 하실 수도 있어서,

회원 탈퇴의 경우 PATCH /users 뒤에 추가적인 정보를 더해 설계를 할 수도 있습니다.

이렇듯, **API 설계에는 절대적인 정답은 없고**

위에서 나열한 5개의 규칙을 지키되,

요구 사항에 따라

피치 못한 경우는 무조건 저 규칙을 따르지 않아도 됩니다.

결국 프론트엔드 개발자 분들이 우리의 API 설계를 보고

쉽게 무슨 API인지 감을 잡으면 되는 거예요!

👉 리소스 간 연관 관계가 있는 경우

예를 들어 교과목이 한 명의 사용자(교사)가 여러 개의 교과목을 강의 할 수 있다고 하고,

교사와 교과목이 1 : N 관계인 경우

이런 상황에서 교과목의 목록을 조회하는 API를 설계한다고 가정해봅시다.

교과목은 교사와 1 : N 관계를 가지므로, 아래처럼 설계를 할 수 있습니다.

편의를 위해 사용자는 오로지 교사만 있다고 하겠습니다.

🔑 **/users/subjects**

이런 식으로 교사가 여러 개의 과목을 강의하기에 계층 관계는 **교사 다음 교과목**으로 잡고,

uri상에 교사가 더 계층 관계 상 우선이 된다는 것을 표현할 수 있습니다.

만약, 특정 교사의 교과목 목록을 보고 싶다면 어떻게 할 수 있을까요?

🔑 **/users/id/subjects**

이렇게 설계가 가능합니다.

가운데 id는 아래에서 path variable을 설명하면서 다시 설명 하도록 하겠습니다.

만약, 교과목 하나 단건 조회 API를 설계한다면?

이런 경우는



특정 교사의 특정 교과목이 더 의미가 잘 전달이 되는지,
아니면 그저 특정 교과목이 더 의미가 잘 전달이 되는지.
요구 사항에 더 맞는 설계를 선택하면 됩니다.



/users/id/subjects/id

(id 부분은 아래 path variable에서 다시 설명 할 예정입니다. 왜 id를 두 번 쓰는지 의문이 들더라도 조금만 기다려주세요.)

혹은

/users/subjects/id

이렇게 설계가 가능합니다.

🤔 N : M 관계는?

API 설계에서 가장 짜증나는 경우😓입니다.

게시글과 해시태그가 이런 경우죠.

앞선 주차 워크북에서 게시글과 해시태그가 N : M 관계라고 했죠?

이런 경우 계층 관계를 한눈에 보기가 매우 힘들게 됩니다.



/articles/hash-tags가 나올 지

/hash-tags/articles가 나올 지

난감합니다.

이런 경우는 비즈니스 로직상 더 중요한 대상을 계층 관계에서 앞에 두는 방법이 있습니다.

사실 위의 경우는 직감적으로 **해시태그는 부수적인 것이고**,

게시글이 서비스에서 더 중요한 역할을 한다는 것은 쉽게 느낄 수 있죠.

이런 경우는



/articles/hash-tag

이렇게 설계가 가능합니다.

이처럼 API 설계에서도 기획의 요구 사항이 영향을 끼치므로

반드시 PM, 프론트 개발자 분들과 많은 소통을 하여 설계를 하는 것이 좋습니다.

세부적인 API 설계

이제 세부적인 API 설계를 어떻게 할지 생각을 해봅시다.

이제, 아래의 키워드를 API 설계에서 포함해서 고려를 할 것입니다.



1. **path variable**
2. **query string**
3. **request body**
4. **request header**

위의 4개 중 **API endpoint**에 포함이 되는 것은 **path variable** 하나 뿐입니다.

사실 **API Endpoint**는 간략하게 **[이런 역할을 하는 API야!]** 정도만 알려주는 것입니다.

예를 들어

회원 가입을 한다고 합시다.

POST /users 정도로 엔드포인트 설계는 끝이 나겠죠.

그러나 위의 정보 만으로는 부족한 것이 있습니다.

🤔 회원 가입이란 , 회원 정보를 새로 저장하는 건데, 무슨 정보를 저장해?

그렇죠?

엔드 포인트 하나 만으로는 실제 동작 수행에 필요한 정보를 표현할 수 없죠.

이런 경우 바로 위의 4가지를 사용하게 됩니다. (path variable, query string, request body, request header)

우선 **path variable**과 **query string**을 알아보시다.

Path Variable

이런 경우가 있을 수 있겠죠.

| 게시글 하나 상세 조회 API

위의 경우 곧바로 단건 조회, 즉 GET에 단건 조회라는 것은 빠르게 캐치가 가능합니다.

여기서 어떻게 특정 게시글임을 서버에게 전달이 가능할까요?

다시 말해, 서버에게 어떻게 원하는 게시글을 식별할 수 있는 데이터를 넘길 수 있을까요?

이런 경우 **path variable**을 사용합니다.

| 단 하나, 특정 대상을 지목할 때

/users/articles/id 처럼 id에 게시글의 식별 값을 넣어서 전달하게 됩니다.

따라서 실제로는 아래와 같겠죠.

GET https://umc.com/users/articles/4 ← 저 4에 주목!

여기서 저 4가 데이터베이스 상에서 게시글의 기본키라는 것을 캐치 한다면 잘 이해한 것입니다.

따라서

/users/articles/id에서 id는 진짜 문자 그대로 id가 아니라 **게시글의 식별 값을 의미**하죠.

따라서 **실제 API 명세서에서는 아래처럼 표현**합니다.



GET /users/articles/{**articleId**}

위 처럼 { }(중괄호)로 감싸는 부분은 path variable을 의미하고,
그냥 id 보다는 articleId처럼
더 명확히 표시해주는 것이 더 좋습니다.

그리고 **단어 구분을 -를 사용해서** 한다고 했으니



GET /users/articles/{**article-id**}

이렇게 해도 됩니다. 그런데 꼭 RESTful 설계를 타이트하게 따르지 않아도 되기에

articleId 이렇게 표현을 해도 됩니다.



GET /users/articles/{**articleId**}

Query String

쿼리 스트링은 언제 사용이 되냐,

| **게시글 중 이름에 umc가 포함된 게시글들을 조회하려고 할 때**

이런 경우, 1개가 될 수도 있고 여러 개가 될 수도 있잖아요?

따라서 단 하나만 조회하는 path variable을 사용하는 것은 아니고,

query string을 활용하는 것입니다.

쿼리 스트링은 보통 **검색 조회** 때 사용이 되며 따라서 **GET 요청에서 사용**이 됩니다.



GET /users/articles?**name=umc**

위처럼 **name**이 **umc**라는 것을 전달 할 수 있고,

만약 **전달 할 정보가 여러 개**일 경우 아래와 같이



GET /users/articles?**name=umc&owner=ddol**

이런 식으로 &를 통해 쿼리 스트링으로 전달을 하려는 값을 여러 개 연결 할 수 있습니다.

! 주의할 점은, 쿼리 스트링은 **API 엔드 포인트에 포함**이 되지 않기에
엔드 포인트 자체는 **GET /users/articles** 이렇게 설계를 해야 합니다.

Request Body

POST 방식의 경우는 어떻게 서버로 데이터를 전달할까요?

POST는 조회가 아닌, 생성을 위해 사용하기도 하기에 url에 해당 정보들을 노출하는 것은
굉장히

위험할 수 있습니다.

따라서 **url에 노출이 되지 않고 request body에 해당 데이터를 담을 수 있으며,**
보통

json의 형태 혹은 **form-data** 형태로 보내게 됩니다.

form-data 형태의 전송은 11주차 AWS S3에서 더 자세히 다룹니다.

예를 들어 회원 가입 시 아래의 정보가 필요하다고 합시다.

1. 이름
2. 전화번호
3. 닉네임

이럴 경우 request body에 위의 정보를 json으로 담아서 서버로 전송하게 됩니다.

```
{
  "name" : "최용욱",
  "phoneNum" : "010-1111-2222",
  "nickName" : "ddol",
}
```

Request Header

request header는 서버와 전송 시 메타데이터,
즉, **전송에 관련된 기타 정보들이 담기는 부분**입니다.

body에 담기는 데이터의 형식이 json인지, 아니면 form-data인지를 담기도 하고
혹은 데이터를 담을 수 도 있습니다.

대표적으로 로그인이 되었다는 것을 알려주기 위한 토큰을 헤더에 담기도 합니다!

API 설계 예시

예를 들어 닉네임 변경(**사용자 정보 변경**) API에 대한 설계를 한다고 할 경우,
아래와 같은 정보를 명시할 수 있습니다.

API Endpoint

PATCH /users/{userId}

Request Body

```
{  
  "nickname" : "ddol"  
}
```

Request Header

Authorization : accessToken (String)


accessToken은 로그인 된 사용자가 **나 로그인 된 상태야!** 하고 알려주는 것으로
보통 Authorization이라는 키에 대한 값으로 헤더에 담아서 서버로 보내주게 됩니다.

Query String

필요 없음

API 응답에 대한 것은 추후 8주차에서 다루게 됩니다.

프로젝트 세팅 - Node.js

 Chapter 3. API URL의 설계 & 프로젝트 세팅 - Node.js

프로젝트 세팅 - Spring boot

▼ 프로젝트 세팅 - Spring boot

Spring boot의 경우, 0주차 실습에서 나오듯이 **UMC는 java 17, spring boot 3.3.3을 사용할 예정입니다.**

이제 Spring boot 프로젝트를 생성해주는 고마운 친구를 사용해봅시다.

<https://start.spring.io/>

아래처럼 우선 **Gradle**을 선택하고 Kotlin은 자바를 할 것이라 선택하지 않고
버전은 **3.3.3 버전 중 안정된 버전을 선택**하고

Group, Artifact는 본인이 원하는 것을 입력하고

Jar, Java 17을 선택해주세요

Project

- ☒ Gradle - Groovy
☐ Gradle - Kotlin ☐ Maven

Language

- ☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

- ☐ 3.4.0 (SNAPSHOT) ☐ 3.4.0 (M2) ☐ 3.3.4 (SNAPSHOT) ☒ 3.3.3
☐ 3.2.10 (SNAPSHOT) ☐ 3.2.9

Project Metadata

Group	<u>umc</u>
Artifact	<u>spring</u>
Name	<u>spring</u>
Description	<u>for umc spring</u>
Package name	<u>umc.spring</u>
Packaging	<input checked="" type="radio"/> Jar <input type="radio"/> War
Java	<input type="radio"/> 22 <input type="radio"/> 21 <input checked="" type="radio"/> 17

이후 우측의 Dependencies를 아래 사진처럼 해주세요.

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Lombok

DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

MySQL Driver

SQL

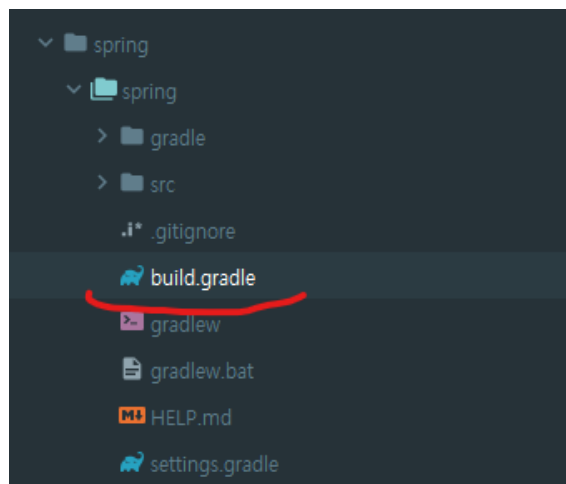
MySQL JDBC driver.

Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

이제 IntelliJ 접속 후 open을 통해 spring initializer로 생성한 프로젝트를 찾아
아래 사진처럼 build.gradle을 클릭 후 open을 해주세요
이후 잠시 설치가 될 때 까지 기다려주세요용 (っ ^_^)っ



인텔리제이 사용법은 스스로 익히세요 (´_`)*:☆

이제 설치가 완료되면 실행을 눌러보세요

아마 안될거예요 😊

안되는 이유는 mysql과 연결 접속 정보가 필요한데 아직 그걸 적어주지 않아서 그렇습니다.

그래서 아래 사진처럼 mysql 의존성은 주석 처리를 해주세요.

주석 후 우측 상단에 나오는 코끼리 모양을 눌러주세요.

```
dependencies {  
    // implementation 'org.springframework.boot:spring-boot-starter-data-jpa'  
    implementation 'org.springframework.boot:spring-boot-starter-web'  
    compileOnly 'org.projectlombok:lombok'  
    // runtimeOnly 'com.mysql:mysql-connector-j'  
    annotationProcessor 'org.projectlombok:lombok'  
    testImplementation 'org.springframework.boot:spring-boot-starter-test'  
}  
  
tasks.named('test') {  
    useJUnitPlatform()  
}
```

이제 실행을 하면 실행이 됩니다!

Spring project는 다음주 워크북부터 차근차근 디벨롭을 해봅시다.

핵심 키워드



주요 내용들에 대해 조사해보고, 자신만의 생각을 통해 정리해보세요!

레퍼런스를 참고하여 정의, 속성, 장단점 등을 적어주셔도 됩니다.

조사는 공식 홈페이지

Best, 블로그(최신 날짜) **Not Bad**

이번 주차는 키워드가 딱히 없습니다!

원하실 경우 Restful API에 대해 한번 더 살펴보면 좋을 것 같습니다.

학습 후기

- 이번 주차 워크북을 해결해보면서 어땠는지 회고해봅시다.
- 핵심 키워드에 대해 완벽하게 이해했는지? 혹시 이해가 안 되는 부분은 뭐였는지?

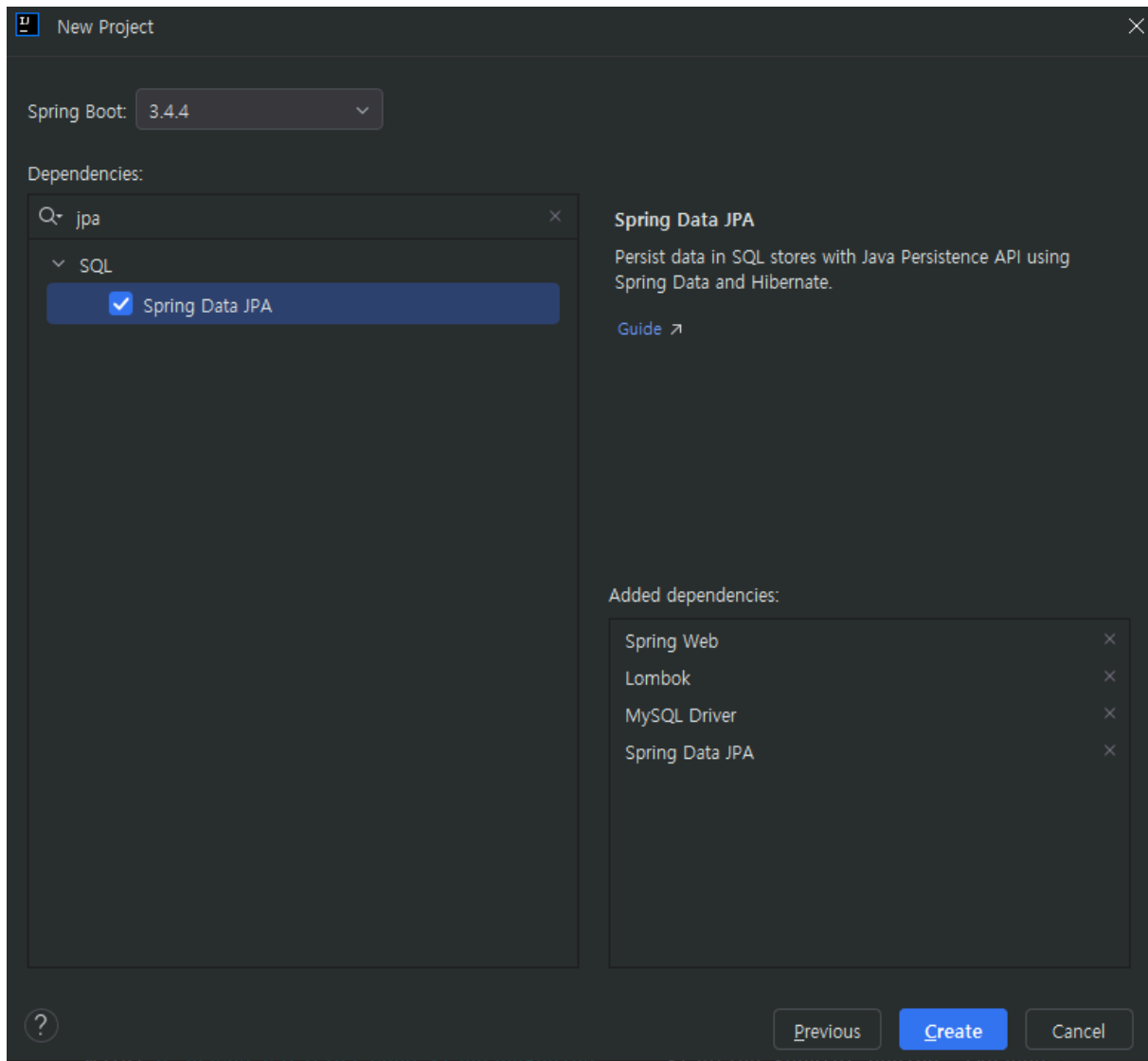


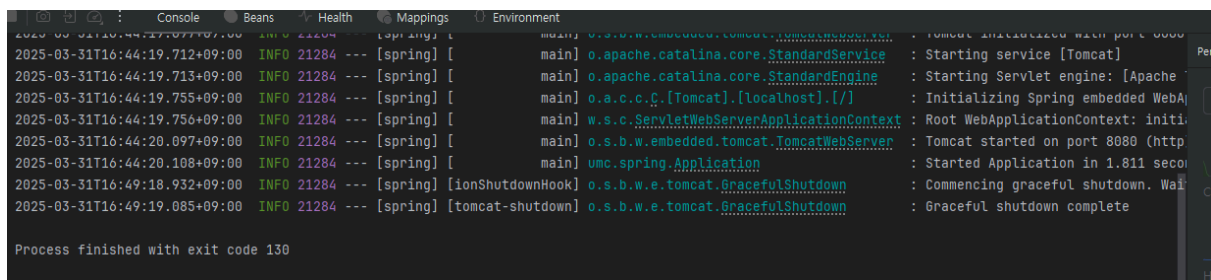
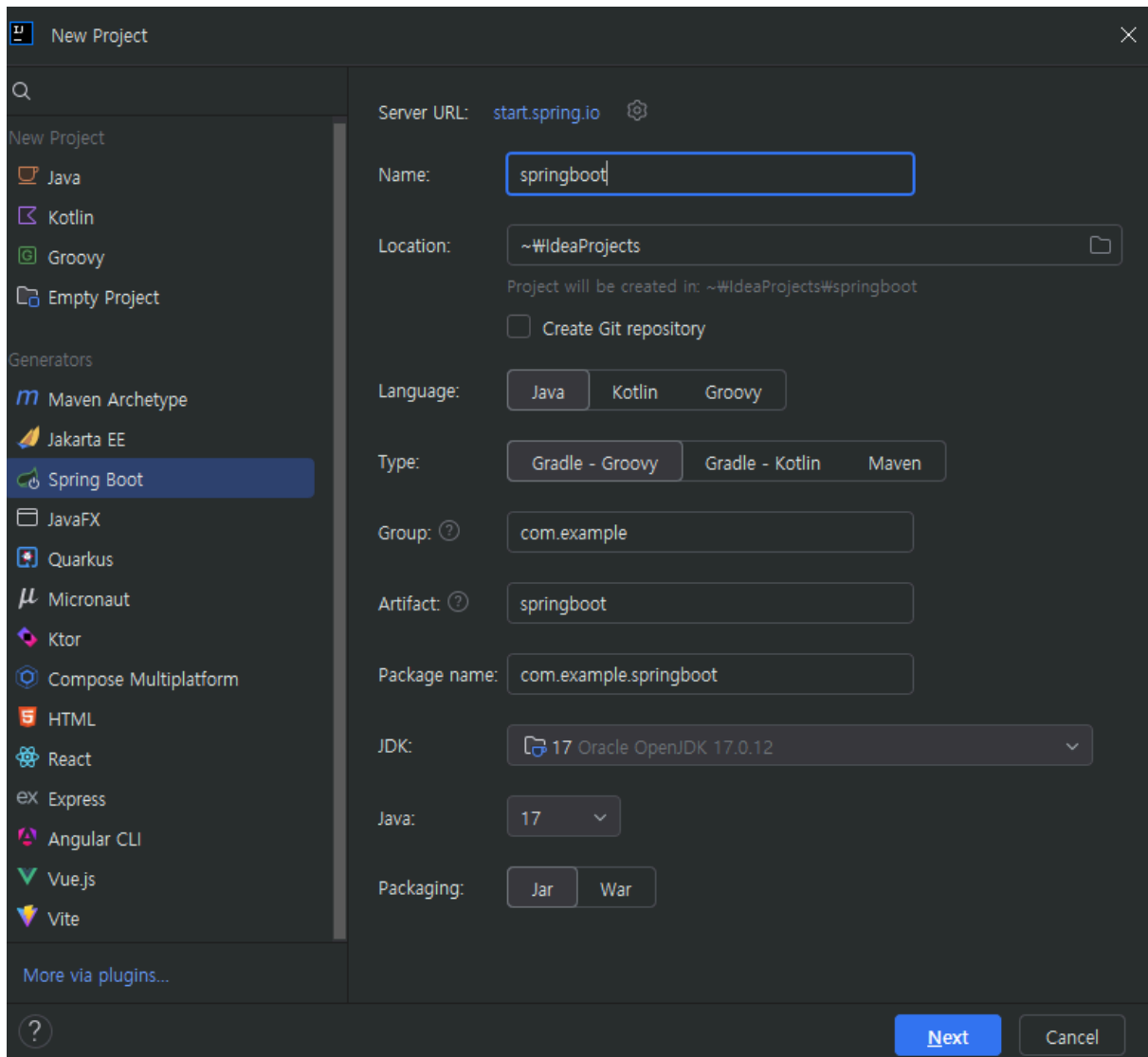
스터디 진행 방법

1. 스터디를 진행하기 전, 워크북 내용들을 모두 채우고 스터디에서는 서로 모르는 내용들을 공유해주세요.
2. 미션은 워크북 내용들을 모두 완료하고 나서 스터디 전/후로 진행해보세요.
3. 다음주 스터디를 진행하기 전, 지난주 미션을 서로 공유해서 상호 피드백을 진행하시면 됩니다.

실습 체크리스트

실습 인증





```
0  java {
1      }
2
3  configurations {
4      compileOnly {
5          extendsFrom annotationProcessor
6      }
7  }
8
9  repositories {
10     mavenCentral()
11 }
12
13 dependencies {
14     //implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
15     implementation 'org.springframework.boot:spring-boot-starter-web'
16     compileOnly 'org.projectlombok:lombok'
17     //runtimeOnly 'com.mysql:mysql-connector-j'
18     annotationProcessor 'org.projectlombok:lombok'
19     testImplementation 'org.springframework.boot:spring-boot-starter-test'
20     testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
21 }
22
23 tasks.named('test') { Task it ->
24     useJUnitPlatform()
25 }
```

🔥 미션

1주차 때 제공된 IA, WF를 참고하여 (아래 **미션 참고 자료**를 보셔도 됩니다!)

▼ 미션 참고 자료



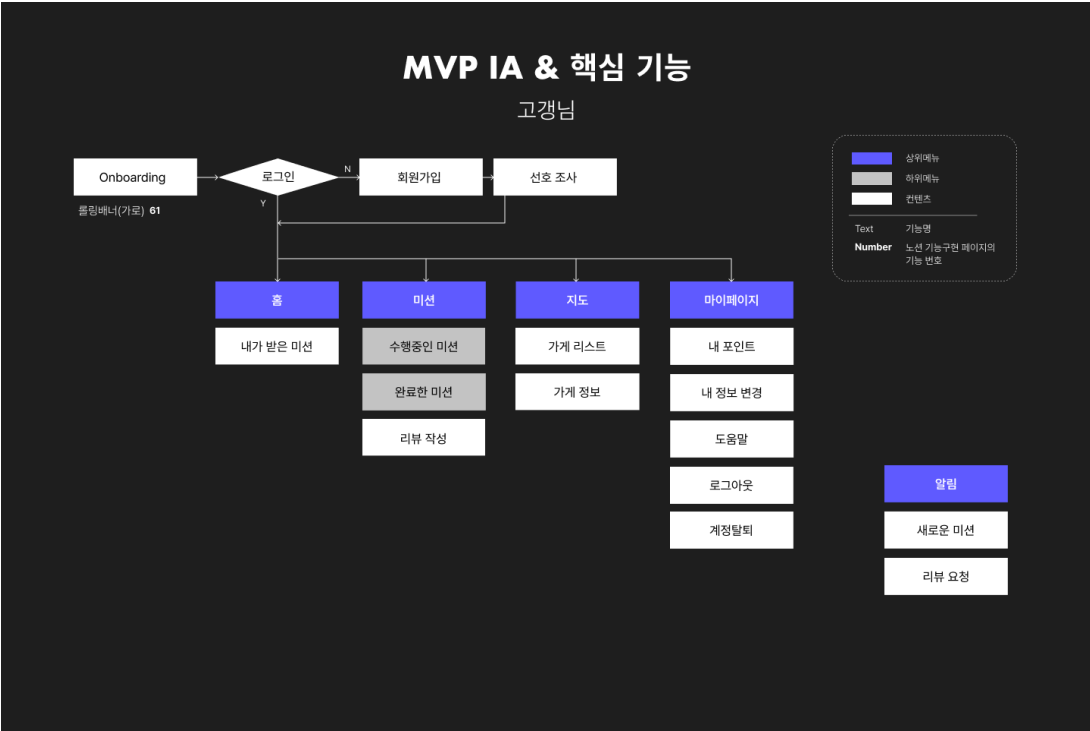
! 해당 자료는 챌린저분들의 미션을 위해 Plan 파트에서 제공해준 자료로, 미션 외의 용도로 사용하는 것 및 유출을 절대 금합니다.

!

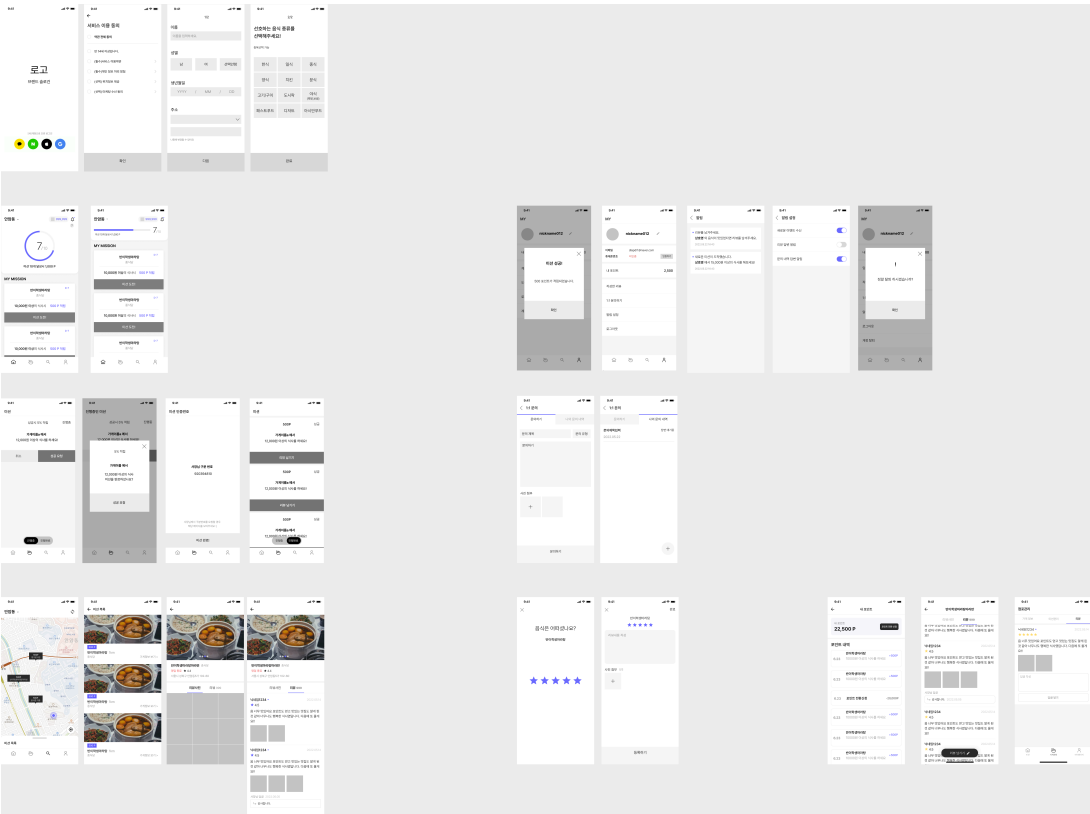
! 아래 IA&WF 사진 및 for_UMC.fig 파일 및 파일 속 내용의 저작권은 모두 7th UMC Plan 파트장 아크(박승민)에게 있음을 밝힙니다. **!**

▼ IA&WF 사진 파일

아래 사진 외에도 첨부드린 사진에 해당하는 피그마 파일이 존재하니 사진을 통해 보기 어려우시다면 피그마 파일을 이용해주세요!



IA



WF

for_UMC.fig



홈 화면, 마이 페이지 리뷰 작성, 미션 목록 조회(진행중, 진행 완료), 미션 성공 누르기,
회원 가입 하기(소셜 로그인 고려 X)

위의 기능을 구현하는데 **필요한 API들을 설계**하여



API Endpoint, Request Body, Request Header, query String, Path variable

이 포함된 간단한 명세서를 만들기!

실제 명세서는 Spring Boot 8주차 및 Node.js 9주차, Swagger 설정에서 더 자세히 다룹니다.

< 시니어 미션 >



시니어 미션



미션 기록



미션 기록의 경우, 아래 미션 기록 토글 속에 작성하시거나, 페이지를 새로 생성하여 해당 페이지에 기록하여도 좋습니다!

하지만, 결과물만 올리는 것이 아닌, **중간 과정 모두 기록하셔야 한다는 점!** 잊지 말아주세요.

▼ 미션 기록

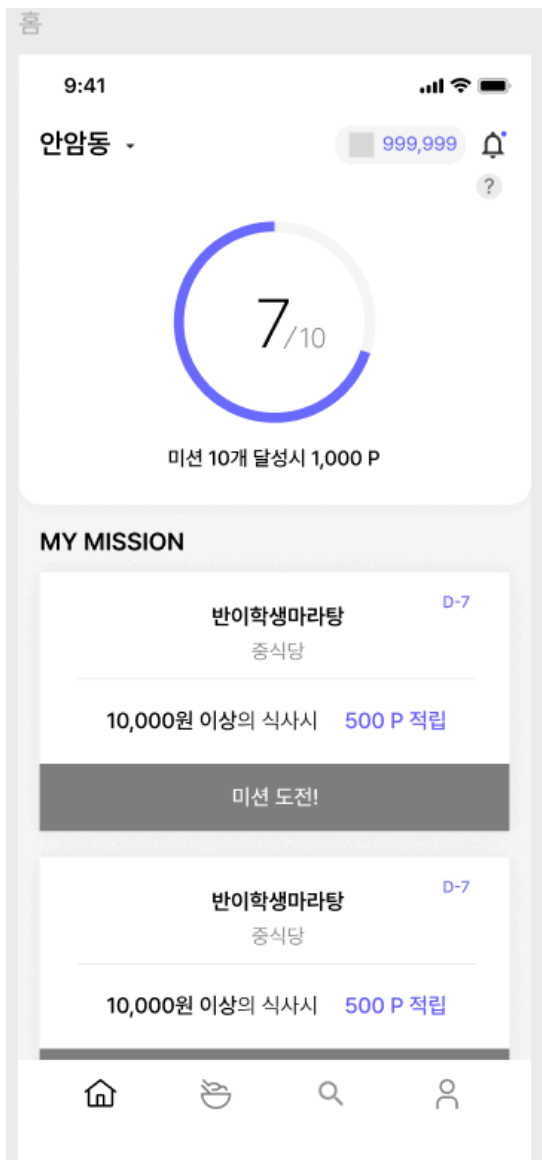
api에 대해 워크북을 참고하여 공부를 진행했지만 막상 하려고 하니 어떻게 해야하는지 잘 몰라 api명세서 작성 가이드를 참고하려고 **인터넷을 참고**하였습니다. 링크는 아래 달아두겠습니다.

<https://cobinding.tistory.com/165>

1. 홈화면
2. 마이페이지 리뷰 작성
3. 미션 목록 조회(진행중, 진행완료)
4. 미션 성공 누르기
5. 회원 가입하기(소셜 로그인 고려 X)

미션을 총 5가지의 문제로 나눌 수 있으며, 먼저 각 문제는 어떤 메서드를 사용해야하는지 부터 생각해보았습니다.

1. 홈 화면



현재 홈 화면을 보면 MY MISSION 섹션 아래로 2개의 **내가 받은 미션들을 확인할** 수가 있습니다.

각 미션을 자세히 살펴보면

"반이학생마라탕" = 이름

"D-7" = 남은 일 수

"500 P" = 적립 포인트

"미션 도전!" = 버튼

을 확인할 수 있으며, 이 화면에서 필요한 데이터는 미션ID, 이름, 남은 일수, 적립 포인트, 미션 상태 등등의 필드가 필요한 사실을 알 수 있습니다.

또한 상단 "미션 10개 달성 시 1,000 P"라는

부분을 참고하여

1. 사용자가 달성한 미션 수
2. 적립포인트

또한 포함시켜야한다고 생각하였습니다.

1. GET : 조회
2. POST : 생성
3. PUT : 갱신(전체)
4. PATCH : 갱신(일부)
5. DELETE : 삭제

중 **GET** 메서드로 사용자의 미션 목록을 조회하는 API가 필요하다고 생각하였습니다.

API 설계

```
GET/api/users/{user-id}/missions?status=before-starting&type=member
```

= 사용자 미션 현황 및 목록

--

2. 마이페이지 리뷰 작성



문제의 의도를 잘 이해못하여 곰곰히 생각해본 저의 생각에 따르면...ㅠ

리뷰작성이 아닌 마이페이지 리뷰 작성이라고 적혀있어 .fig파일의 WF를 봤지만

리뷰와 관련해서 있는 사진이 좌측의 사진이 끝이라 "마이페이지"리뷰작성은 마이페이지에서 내가 적은 가게의 리뷰를 보는 기능이라고 생각하여 그렇게 작성하도록 하겠습니다.

좌측 사진의 리뷰를 확인해보면

1. "닉네임1234" = 닉네임
2. "반이학생마라탕마라반" = 가게이름
3. "★ 4.5" = 별점
4. "음 너무 맛있어요~~~..." = 내용
5. "■" = 사진

등등의 정보를 알 수 있습니다.

리뷰 작성 API는 **POST** 메서드로 구현했으며,

요청 body에는 사용자의 ID, 가게ID, 별점, 내용, 이미지 등을 포함해야한다고 생각합니다.

2주차 과제의 데이터베이스 스키마를 생각해보면 리뷰테이블에

1. user_id
2. store_id
3. rating

4. content
5. picture
6. created_at과 updated_at

필드가 필요할 것이고, 이미지는 여러개 올릴 수 있으니까 배열로 처리해야 합니다.

API 설계

POST/api/users/{user-id}/reviews

Request Body

```
{
  "id" = "int",
  "store_id" = "int",
  "rating" = "int",
  "content" = "string",
  "picture" = "url[]" // 여기서 약간의 혼동이 있었습니다. 그 이유는 아래에 기재했습니다.
}
```

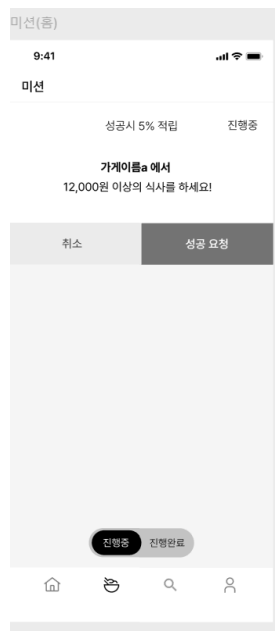
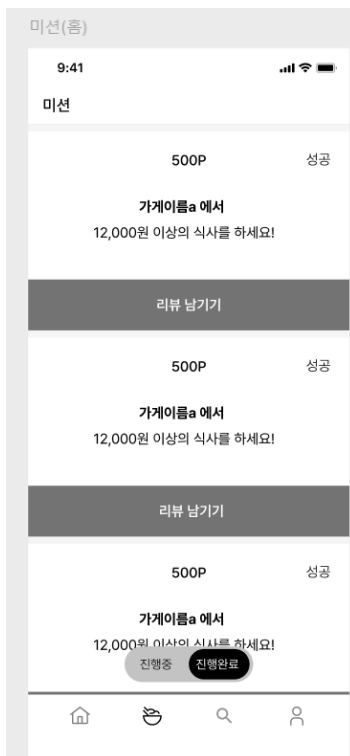
처음 "picture" = "url"로 적었지만 이미지를 여러개 올릴 수 있다는 생각에 array로 사용하려고

"picture" = "array"

+ "picture": "url[]" 라고 적었지만 이 코드의 문제는 "url[]"는 단순히 문자열로 인식된다는 문제점이 있으며, 실제로 사진 URL을 여러개 넣을 수 없다는 사실을 알게 되었습니다.

배열 사용은 맞는 []문법은 필수인 건 맞지만 url[]표기는 금지라는 사실을 알게 되어 "picture" = "url[]"로 기재하게 되었습니다.

3. 미션 목록 조회 (진행 중, 진행 완료)



이 사진들을 보면 알 수 있듯이 진행 중, 진행완료 두 가지의 미션 목록을 조회할 수 있습니다.

“조회”의 영역이기 때문에 GET 메소드를 사용하는 것이 맞다고 판단했으며,

1. 진행 중
2. 진행완료

로 2가지로 나누어

api endpoint를 적어보겠습니다.

이번엔 json파일의 예시도 이해를 돕기위해 적어보았습니다.

API 설계

1. 미션 목록 조회 (진행 중 상태일 때)

GET/api/users/{user-id}/missions?status=doing

Request Body

```
{
  "id": "int",
  "title": "string",
  "status": "string", // 진행 중 string도 가능하며 0과 1로 표현하는 int도 가능
  "reward": "int",
  "created_at": "string"
}
```

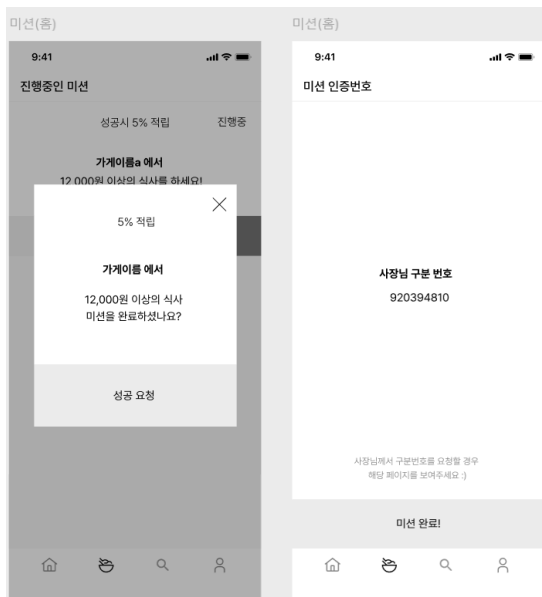
2. 미션 목록 조회 (진행 완료 상태일 때)

```
GET/api/users/{user-id}/missions?status=complete
```

Request Body

```
{
  "id": "int",
  "title": "string",
  "status": "string", // 진행 중 string도 가능하며 0과 1로 표현하는 int도 가능
  "reward": "int",
  "created_at": "string"
}
```

4. 미션 성공 누르기



또한 미션 인증번호가 있는 걸로 보아 인증 관련 로직도 구현해야함을 알 수 있습니다.

왼쪽의 사진을 보면 '성공 요청' 버튼과 '미션 완료!' 문자가 포함되어있고, 이를 통해 사용자가 미션 성공 시 서버에 성공 요청을 보내고, 해당 미션의 상태를 변경해야함을 알 수 있습니다.

미션 성공 API는 POST와 PATCH 둘 중 하나의 메서드를 사용해야하며, 요청시 미션 ID를 받아 해당 미션의 상태를 complete로 변경하는 과정이 필요합니다.

더 나아가 변경된 데이터를 저장하기 위해 JSON파일을 업데이트 하는 과정이 포함되어야합니다.


```
PATCH/api/users/{user-id}/missions/{mission-id}
```

Request Body

```
{  
  "auth_number": "int", // 사진에 있는 번호  
  "complete_reason": "string" // "완료"같은 string이나 0과 1인 int 표현도 가능  
}
```

+a PATCH를 통한 인증번호 성공 api endpoint도 구현해보았습니다.

```
PATCH /api/users/{user-id}/missions/{mission-id}  
Headers: Content-Type: application/json  
Body: { "auth_number": "int" } // 사진 인증번호 예시
```

+b 엔드포인트 구조는 봤을 때 이해가 된다는 선에서 최대한 간결한게 좋다고 하여 {user-id}가

미션 데이터에 이미 포함되어 있다면 중복되게 되므로 간단하게 표현하게 된다면

```
PATCH/api/missions/{mission-id}
```

으로 표현 할 수도 있겠다고 생각하여 추가로 기재해보았습니다.

5. 회원가입하기 (소셜 로그인 고려 X)

회원가입1

9:41 1/2

이름

이름을 입력하세요.

성별

남 여 선택안함

생년월일

YYYY / MM / DD

주소

나중에 변경할 수 있어요

다음

회원 가입 역시 새로운 사용자의 생성이기 때문에 `POST /users`로 표현이 가능합니다. 따라서 **회원 가입**의 경우가 있기 때문에 로그인을 `POST /users`로 표현하기가 힘들죠.

이런 경우에는 너무 타이트하게 RESTful 한 API 설계를 따르려고 하지 않아도 됩니다.

👉 `POST /users/login`

이런 형식으로 만들어도 괜찮습니다.

위 워크북에서 배운대로 `POST/user`으로 표현이 가능하지만 로그인을 `POST/users`로 표현하기 힘들어

너무 타이트하게 RESTful한 API 설계를 따르려고 하지 않고

`POST/users/login`

으로 표현해도 된다고 함으로써

5번문제 회원가입 외에도 저의 알파한($\pi.\pi$) 지식으로

로그인 API까지 구현해보도록 하겠습니다.

API 설계

1. 회원가입 API

`POST/users`

Request Body

```
{
  "id" = "int"
  "name" = "string"
  "gender" = "int" // 남자,여자 문자열 string도 가능하며 0과 1로 표현하는 int도
  "birthday" = "string"
```

```
"adress" = "string"
}
```

2. 로그인 API (소셜(?) 이메일 로그인으로 가정)

POST/users/login

Request Body

```
{
  "email" = "string"
  "password" = "string"
}
```

- 로그인 API는 js의 bcrypt 사용이나 비밀번호는 해시 처리하여 저장한다던지, 로그인 시 JWT 토큰을 발급하는 방식등 많은 방식 등등 다른 방법들로 보안 처리가 필수긴 하지만 아직 지식이 없어 배제하도록 하겠습니다
-
-


트러블 슈팅



실습하면서 생긴 문제들에 대해서, **이슈 - 문제 - 해결** 순서로 작성해주세요.



스스로 해결하기 어렵다면? 스터디원들에게 도움을 요청하거나 **너디너리의 지식 IN 채널에 질문**해보세요!

▼  이슈 작성 예시 (이슈가 생기면 아래를 복사해서 No.1, No.2, No3 ... 으로 작성해서 트러블 슈팅을 꼭 해보세요!)

이슈

 앱 실행 중에 노래 다음 버튼을 누르니까 앱이 종료되었다.

문제

👉 노래클래스의 데이터리스트의 Size를 넘어서 NullPointerException이 발생하여 앱이 종료된 것이었다.

해결

👉 노래 다음 버튼을 눌렀을 때 데이터리스트의 Size를 검사해 Size보다 넘어가려고 하면 다음으로 넘어가는 메서드를 실행시키지 않고, 첫 노래로 돌아가게끔 해결

참고레퍼런스

- 링크

▼ ⚡ 이슈 No.1

이슈

👉 [트러블이 생긴 상태 작성]

문제

👉 [어떤 이유로 해당 이슈가 일어났는지 작성]

해결

👉 [해결 방법 작성]

참고레퍼런스

- [문제 해결 시 참고한 링크]



🤔 참고 자료

Copyright © 2023 최용욱(똥이) All rights reserved.

Copyright © 2024, 2025 김준환(제이미) All rights reserved.