TodoList 3조

이대현, 강형준

목차

개발 방향성

이번 한 학기동안 공부한 내용을 토대로 간단한 TodoList 를 만들어 보는 것이 목표이기에 , 최대한 완성하여 시연을 해보는 것에 의의를 두었습니다.

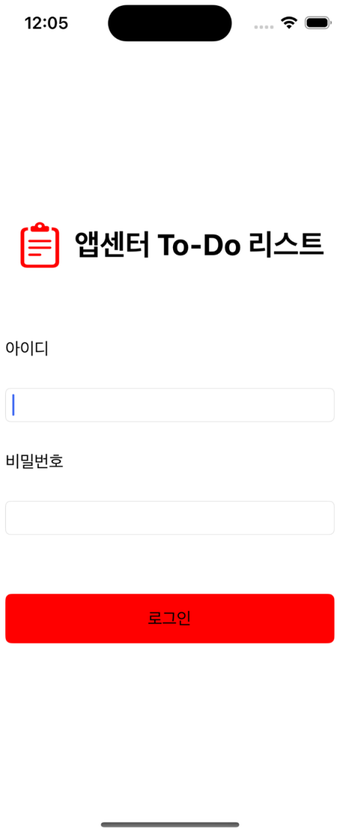
프론트엔드

1. 앱 화면 소개

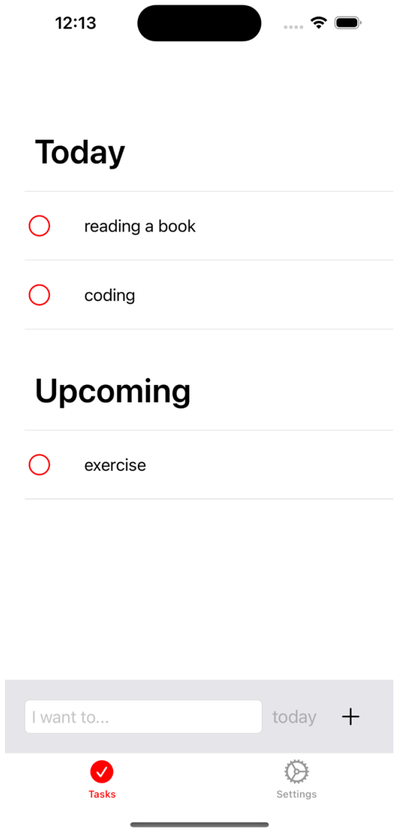
2. 로그인 기능 & JWT 토큰 적용

3. Network 구조 설계

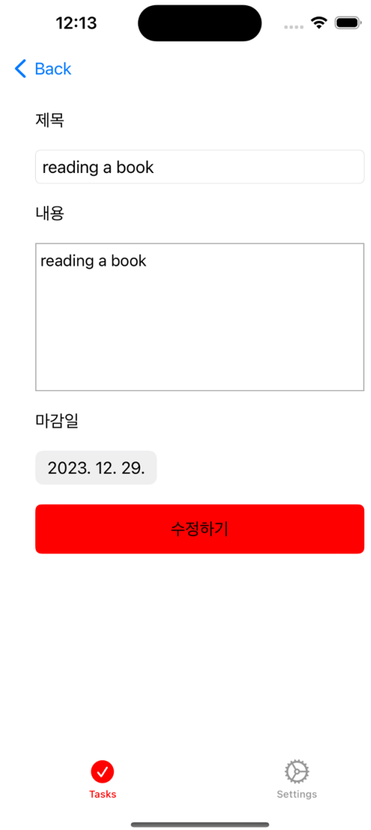
앱 화면 소개



로그인 화면



투두리스트 화면
(메인 화면)



투두 수정 화면
(상세 화면)

로그인 기능 & JWT 토큰 적용

```
if response.statusCode == 200 {
    guard let result = try? response.map(SignInResultDTO.self) else {
        return
    }
    self.saveToken(token: result.token)
    self.moveToMain()
```

로그인 성공시

```
var headers: [String : String]? {
    return [
        "Content-type": "application/json",
        "JWT" : (UIApplication.shared.delegate as? AppDelegate)?.TOKEN ?? ""
    ]
}
```

요청 헤더

Network 구조 설계

- 네트워크 계층 추상화
  - 재사용성을 높이고, 가독성을 높임
  - 개발자는 오직 request, response만 신경 쓰도록 함
- Moya Framework

```
enum TodoAPI {
    // Todo 정보 API
    case getDetail(_ id: Int)
    case updateTodo(_ id: Int, _ params: TodoRequestDTO)
    case createTodo(_ params: TodoRequestDTO)
    case updateCheck(_ id: Int, _ completed: Bool)
    case deleteTodo(_ id: Int)
    // 유저 정보 API
    case getList
    case login(_ id: String, _ password: String)
}
```

요청 api 리스트

```
var path: String {
    switch self {
    case .getList:
        return "/users/search"
    case .createTodo(_):
        return "/todos"
    case .updateCheck(_, _):
        return "/todos/check"
    case .deleteTodo(let id),
        .getDetail(let id),
        .updateTodo(let id, _):
        return "/todos/\(id)"
    case .login(_, _):
        return "/users/sign-in"
    }
}
```

요청 url

```
var task: Moya.Task {
    switch self {
    case .getList:
        return .requestPlain
    case .createTodo(let params):
        return .requestJSONEncodable(params)
    case .updateCheck(let id, let completed):
        let params: [String: Any] = [
            "id": id,
            "completed": completed
        ]
        return .requestParameters(parameters: params, encoding:
            URLEncoding.queryString)
```

request params

Network 구조 설계

- 네트워크 계층 추상화
  - 재사용성을 높이고, 가독성을 높임
  - 개발자는 오직 request, response만 신경 쓰도록 함
- Moya Framework

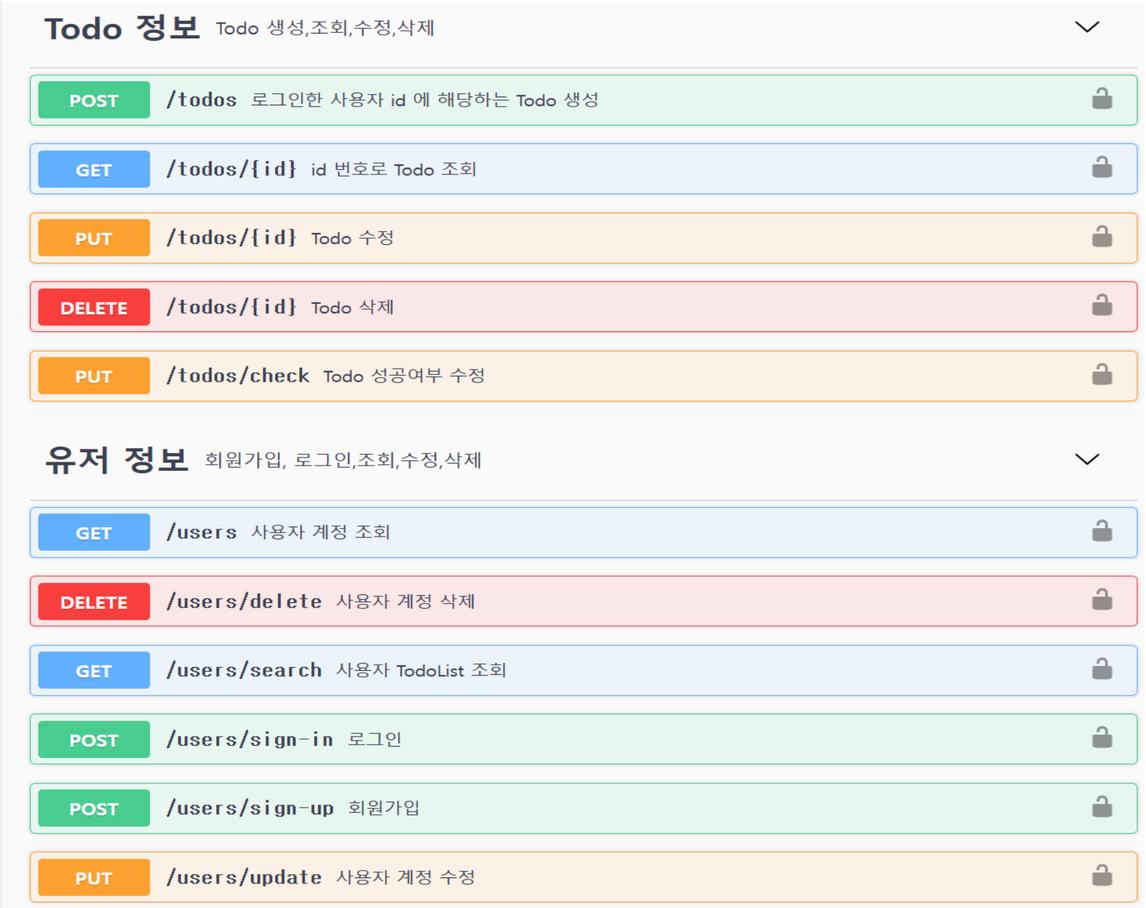
```
provider.request(.createTodo(requestBody)) { result in
    switch result {
    case let .success(response):
        print(String(data: response.data, encoding: .utf8))
        if let result = try? response.map(TodoResponseDTO.self) {
            let task = Todo(id: String(result.id),
                            name: result.title,
                            isCompleted: result.completed,
                            deadLine: result.deadLineDate)
            if section == .Today {
                self.todayTaskData.append(task)
            } else {
                self.upcomingTaskData.append(task)
            }
            self.loadTableView()
        }
```

request & response

백엔드

1. 기본적인 CRUD
2. 레벨 업 기능
3. 예외처리 & 유효성 검사
4. Spring Security와 JWT 적용

기본적인 CRUD



200

Response body


```
[
  {
    "id": 1,
    "title": "나의 하루 루틴",
    "content": "1.독서하기, 2.영화보기",
    "completed": false,
    "deadLine": "2023-12-31T23:59:59",
    "createdDate": "2023-12-28T20:08:30",
    "modifiedDate": "2023-12-28T20:08:30"
  },
  {
    "id": 2,
    "title": "오늘 할일",
    "content": "1.쓰레기 버리기, 2.치킨 먹기",
    "completed": false,
    "deadLine": "2023-12-30T00:00:00",
    "createdDate": "2023-12-28T20:09:12",
    "modifiedDate": "2023-12-28T20:09:12"
  }
]
```


1. 기본적으로 유저와 Todo 에 대해 조회, 생성, 수정, 삭제 구현

1. 추가적으로 사용자 조회에서 해당 사용자가 작성한 Todo를 전부 가져올 수 있게 구현

레벨 업 기능

- 사용자의 할일 성공 여부에 따른 레벨업 기능

- 할일 수행에 대해 소소한 재미를 제공

레벨 업 기능

1. 초기 유저 정보


```
{
  "id": 4,
  "name": "홍길순",
  "email": "stronghj@naver.com",
  "level": 0
}
```


2. 해당 유저의 할일 생성


```
{
  "id": 20,
  "title": "나의 하루 루틴",
  "content": "1.독서하기, 2.영화보기",
  "completed": false,
  "deadLine": "2023-12-31T23:59:59",
  "createdDate": "2023-12-28T18:47:20.851587",
  "modifiedDate": "2023-12-28T18:47:20.851587"
}
```


3. 할일 성공 체크


```
{
  "id": 20,
  "title": "나의 하루 루틴",
  "content": "1.독서하기, 2.영화보기",
  "completed": true,
  "deadLine": "2023-12-31T23:59:59",
  "createdDate": "2023-12-28T18:47:20",
  "modifiedDate": "2023-12-28T18:47:20"
}
```


4. 유저의 레벨 상승


```
{
  "id": 4,
  "name": "홍길순",
  "email": "stronghj@naver.com",
  "level": 1
}
```

예외처리 & 유효성 검사

- 잘못된 이메일 형식

```
{
  "email": "이메일",
  "name": "홍길순",
  "password": "123456789k",
  "uid": "wldbs4746"
}
```

- 에러 메시지

400 Error:
Undocumented

Response body

```
{
  "details": "이메일 형식을 맞춰주세요",
  "message": "MethodArgumentNotValidException 이 발생하였습니다."
}
```

- 잘못된 이름 형식

```
{
  "email": "stronghj@naver.com",
  "name": "@@@@@@",
  "password": "123456789k",
  "uid": "wldbs4746"
}
```

- 에러 메시지

400 Error:
Undocumented

Response body

```
{
  "details": "이름은 한글로 2자 이상 입력해야합니다.",
  "message": "MethodArgumentNotValidException 이 발생하였습니다."
}
```

예외처리 & 유효성 검사

- 잘못된 userId 형식& 중복 userId 형식

```
{
  "email": "stronghj@naver.com",
  "name": "홍길순",
  "password": "123456789k",
  "uid": "w1"
}
```

```
{
  "email": "stronghj@naver.com",
  "name": "홍길순",
  "password": "123456789k",
  "uid": "wldbs4746"
}
```

- 에러 메시지

400 Error:
Undocumented

Response body

```
{
  "details": "크기가 4에서 10 사이여야 합니다",
  "message": "MethodArgumentNotValidException 이 발생하였습니다."
}
```

400 Error:
Undocumented

Response body

```
{
  "details": "중복되는 UserId입니다.",
  "message": "DuplicateKeyException 발생!!"
}
```

- 잘못된 password 형식

```
{
  "email": "stronghj@naver.com",
  "name": "홍길순",
  "password": "123456789",
  "uid": "wldbs4746"
}
```

- 에러 메시지

400 Error:
Undocumented

Response body

```
{
  "details": "영어문자 + 숫자 조합으로 8~20 자 사이로 입력해주세요.",
  "message": "MethodArgumentNotValidException 이 발생하였습니다."
}
```

Spring Security와 JWT 적용

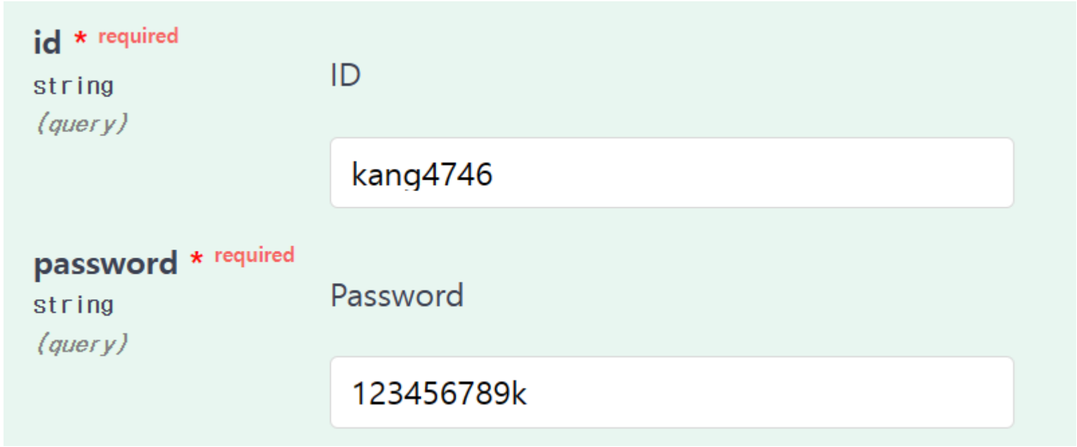
- 권한 없이 리소스에 접근하는 경우

```
[INFO ] [http-nio-5152-exec-2] com.example.mytodolist.config.security.JwtAuthenticationFilter [doFilterInternal] token 값 유효성 체크 시작
[INFO ] [http-nio-5152-exec-2] com.example.mytodolist.config.security.CustomAuthenticationEntryPoint [commence] 인증에 실패하였습니다 : Full authenticat
```

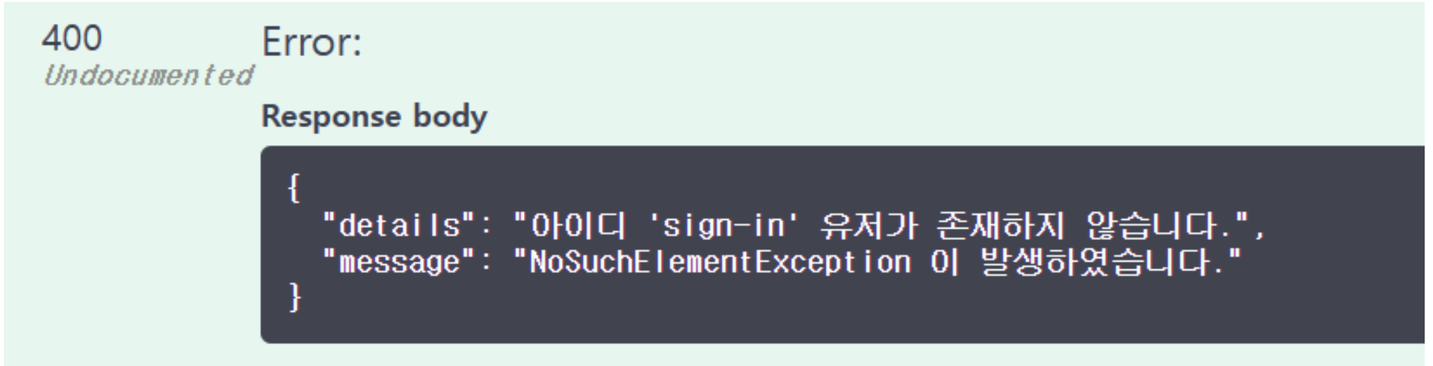
Code	Details
401	Error: **Response body** `"권한이 존재하지않아 인증에 실패하였습니다."`

Spring Security와 JWT 적용

- 로그인 인증 실패



- 에러 메시지



Spring Security와 JWT 적용

- 회원가입한 유저의 userId, password로 로그인

Name	Description
id * required string *(query)*	ID wldbs4746
password * required string *(query)*	Password 123456789k

Response body


```
{
  "id": 1,
  "uid": "wldbs4746",
  "name": "홍길순",
  "email": "stronghj@naver.com",
  "level": 0,
  "token": "eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3bGRiczQ3NDYiLCJyb2xlcyI6WyJST0xFX1VTRVIiXSwiaWF0IjoxNzAzNzYwMzE4LCJleHAiOjE3MDM3NjM5MTh9.MHHoQ31X499oeX7o18FqsEh3YvSWQ5DD2GoY6Y3gKIM"
}
```


- 해당 유저 정보를 기반으로 해당 유저의 JWT 생성

Spring Security와 JWT 적용

- 생성한 토큰을 이용하여 리소스에 접근

Curl

```
curl -X GET "http://hyeongjun.na2ru2.me/users" -H "accept: */*" -H "JWT:
eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ3bGRiczQ3NDYiLCJyb2xlcyI6WyJST0xFX1VTRVIiXSwiaWF0IjoxNzAzNzYxNDU2LCJleHAiOjE3
MDM3NjUwNTZ9.0QuvABI48ML9h-moiVEpgvQR-jc9148Yx6DXDCF2p_s"
```

200

Response body


```
{
  "id": 1,
  "name": "홍길순",
  "email": "stronghj@naver.com",
  "level": 0
}
```

프로젝트 하면서 배운점

- 백엔드와 프론트 엔드의 소통
- API 명세서를 더 구체적으로 작성
- 경험을 통한 자신감