

OUR
SOPT

SHOUT OUR PASSION TOGETHER

3차 세미나

— Retrofit의 활용 —

SHOUT OUR PASSION TOGETHER
SOPT

지난 1,2차 세미나에서는...

지난 1,2차 세미나에서는...

UI

×

chjune0205@gmail.com

비밀번호

로그인

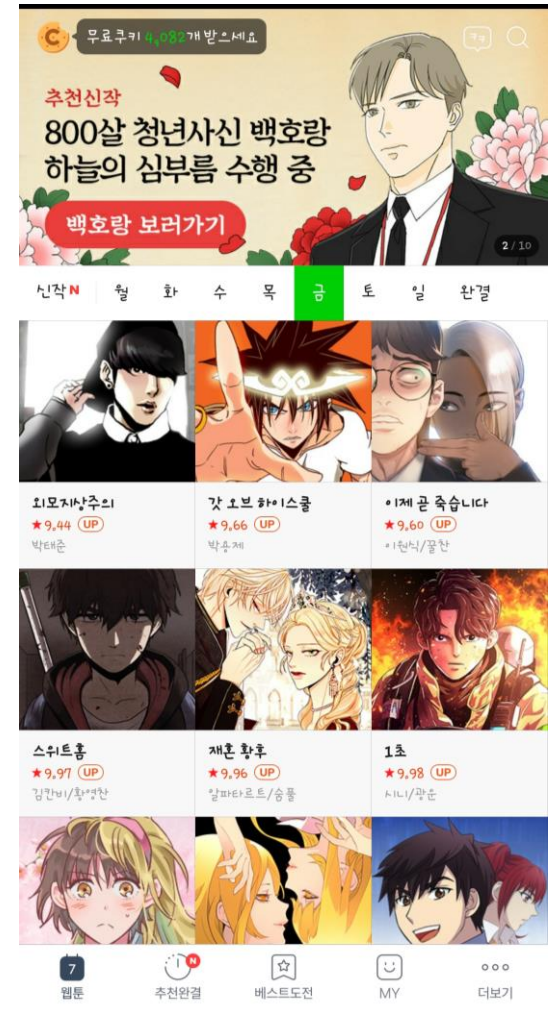
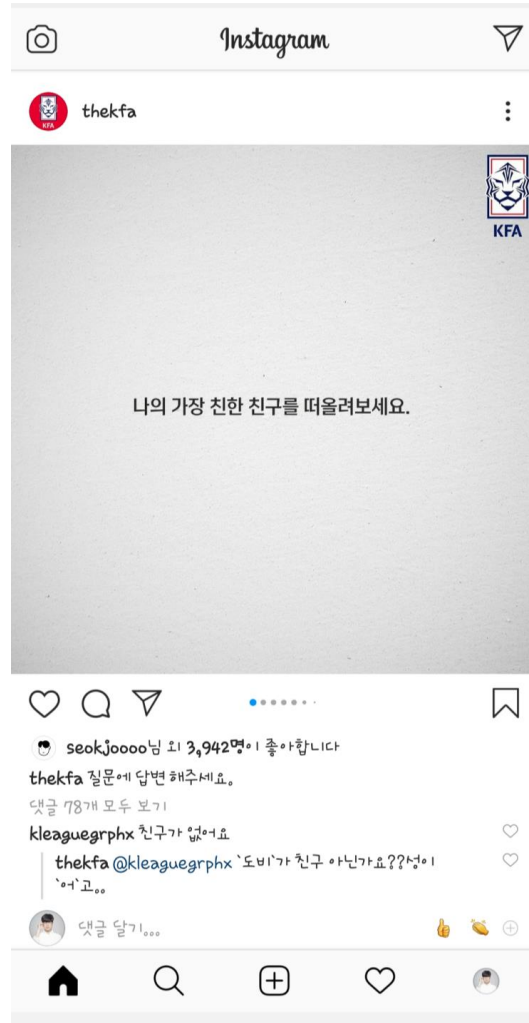
아이디 찾기 | 비밀번호 찾기

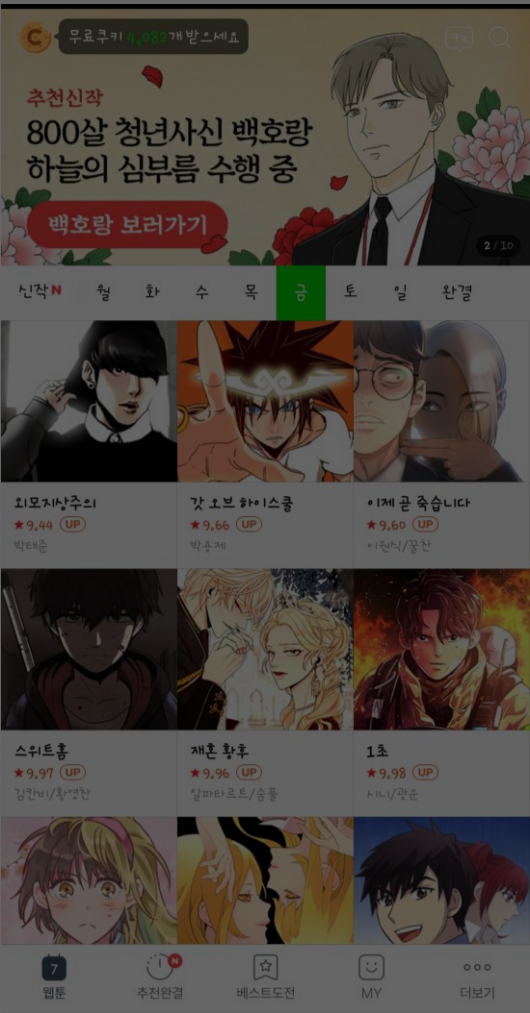
페이스북으로 로그인

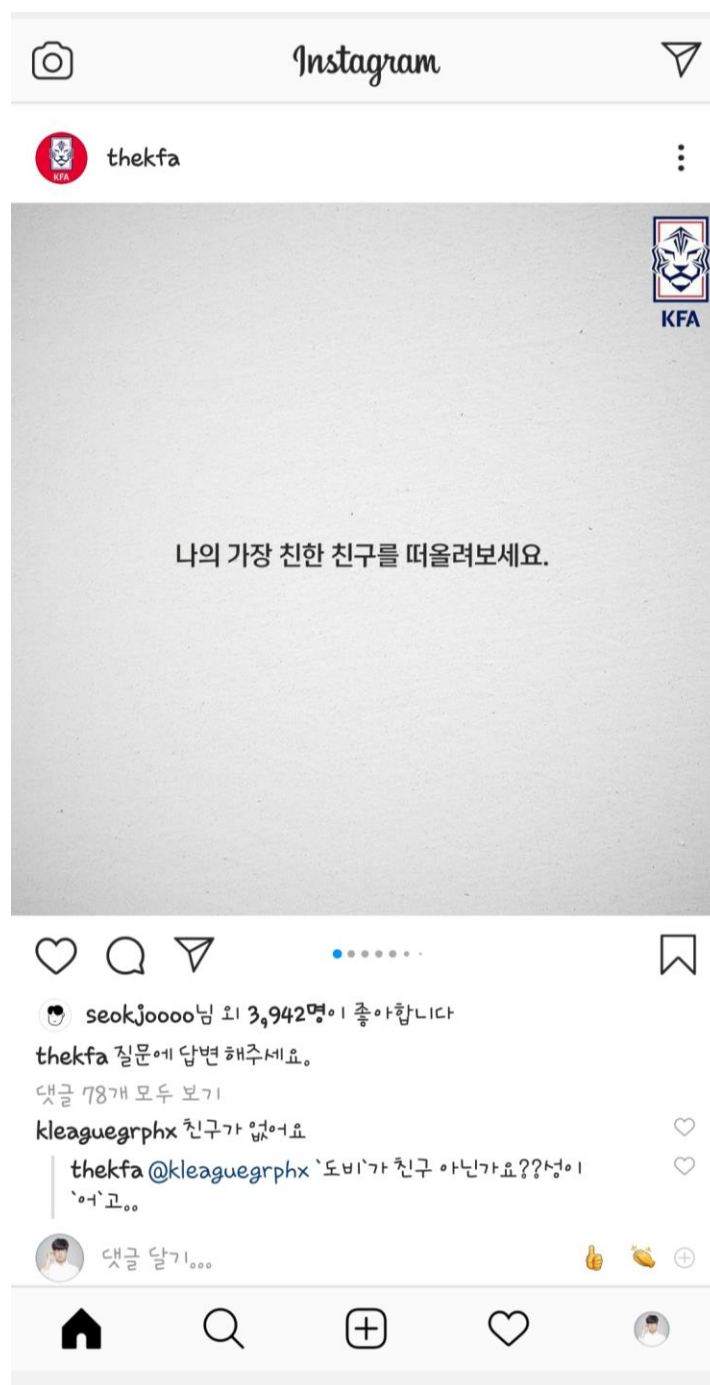
네이버로 로그인

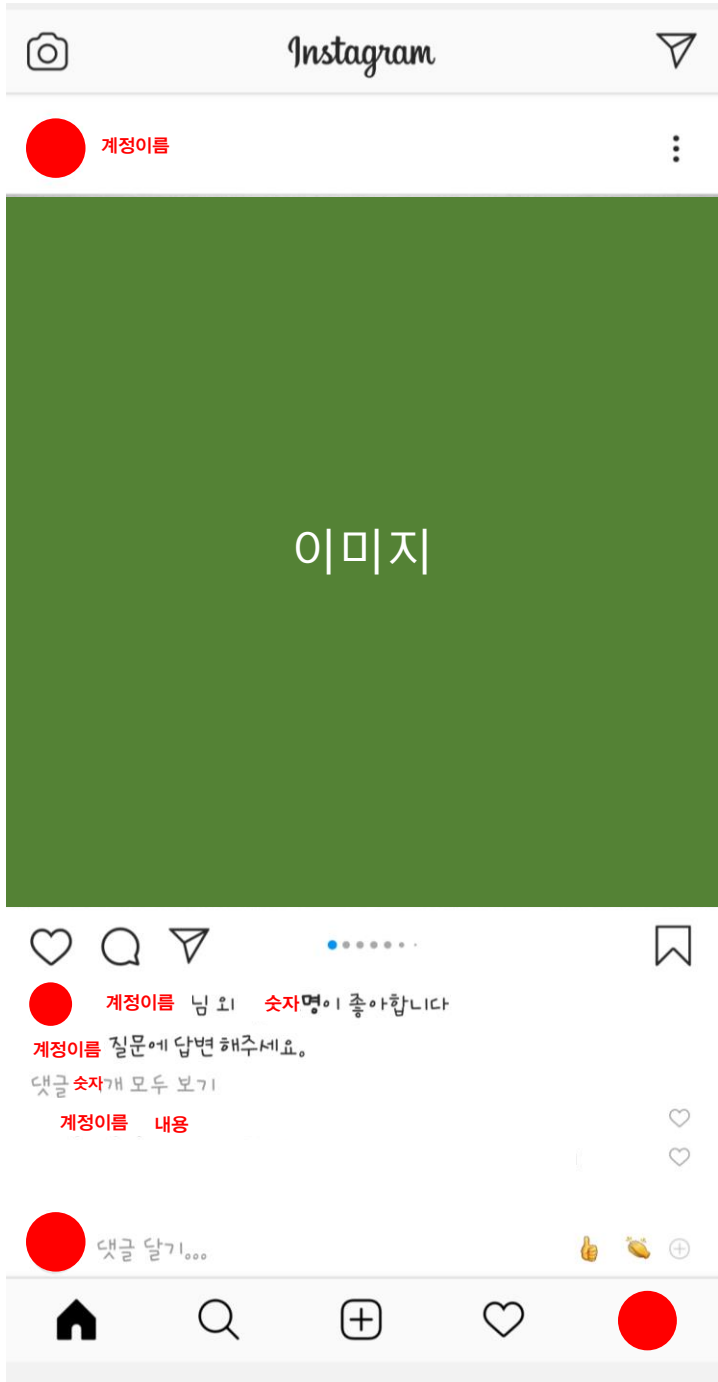
Apple로 로그인

혹시, 배달의민족이 처음이신가요? 회원가입

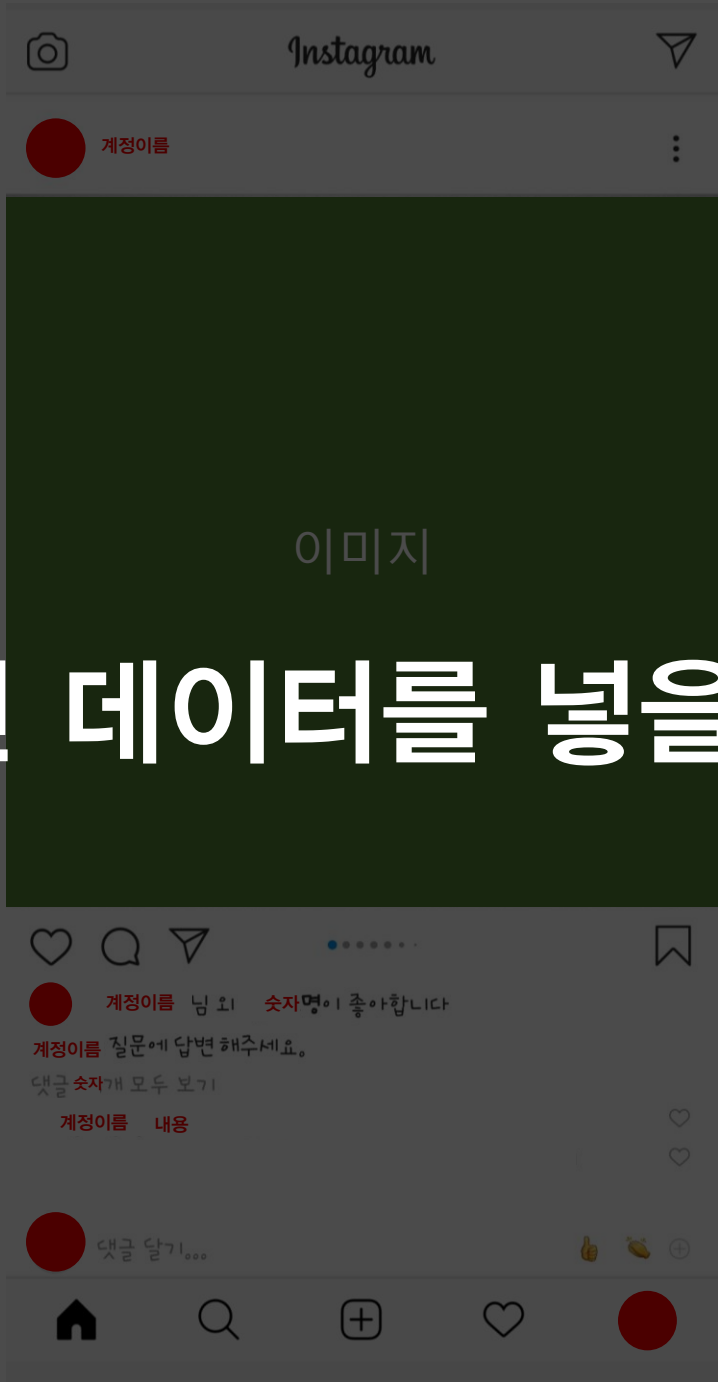




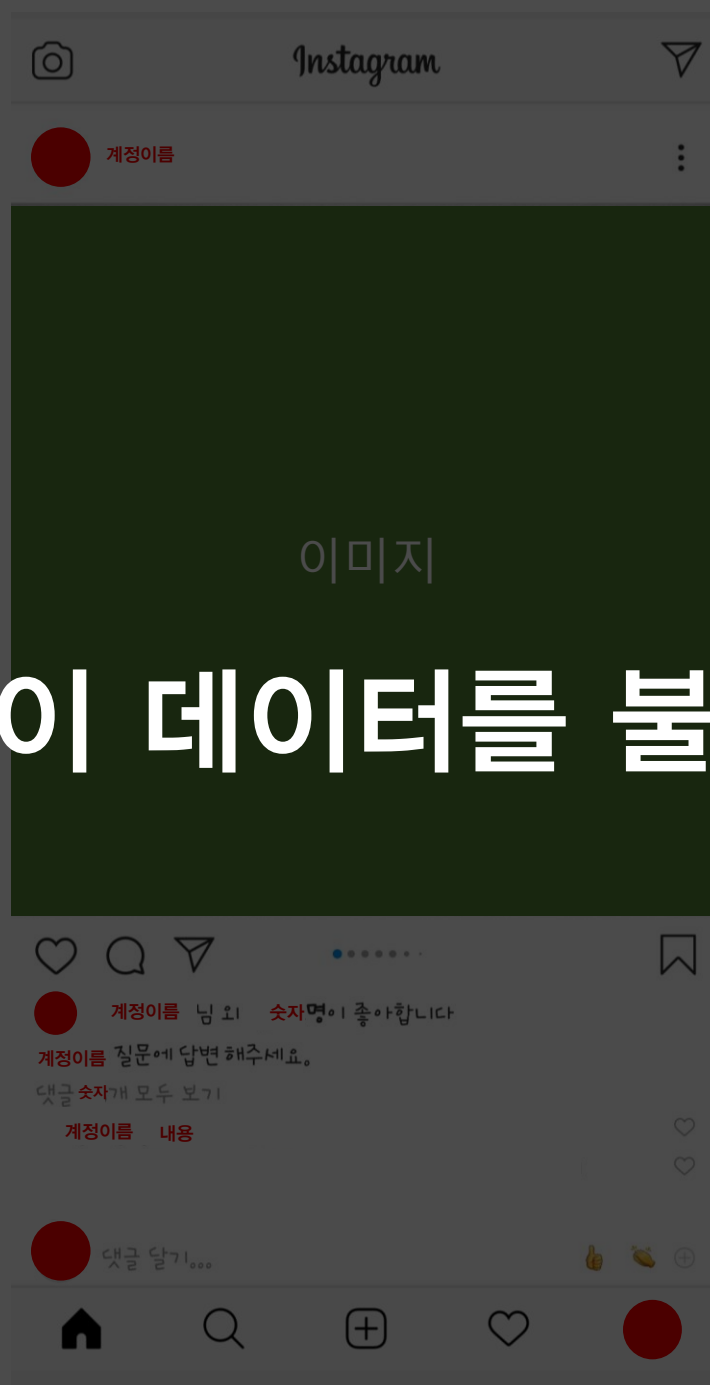




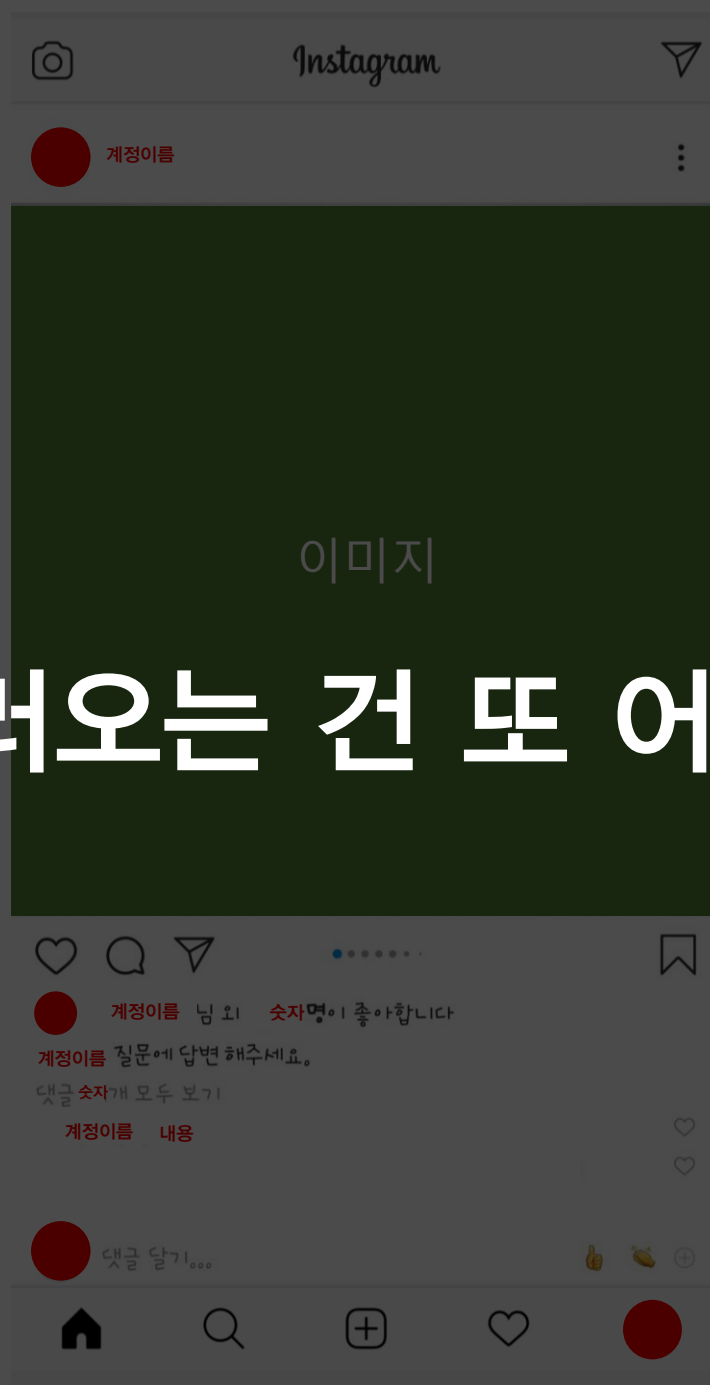
어떤 데이터를 넣을까?



어디서 이 데이터를 불러오지?



데이터 불러오는 건 또 어떻게 할까?



이번 세미나에서는

deSigner
develop
Oper
Planner
sopT

서버와 데이터를 주고 받는 것

00 구체적 목표

01 Server와 Client에 대한 이해

02 Json

03 동기와 비동기

04 Restful API

05 Retrofit의 활용

00 구체적 목표

클라이언트?



역할

서버?



클라이언트?



역할

데이터 전달?

서버?



클라이언트?



역할

데이터 전달?

동기? 비동기?

서버?



클라이언트?



역할

데이터 전달?

동기? 비동기?

Retrofit2 API

서버?

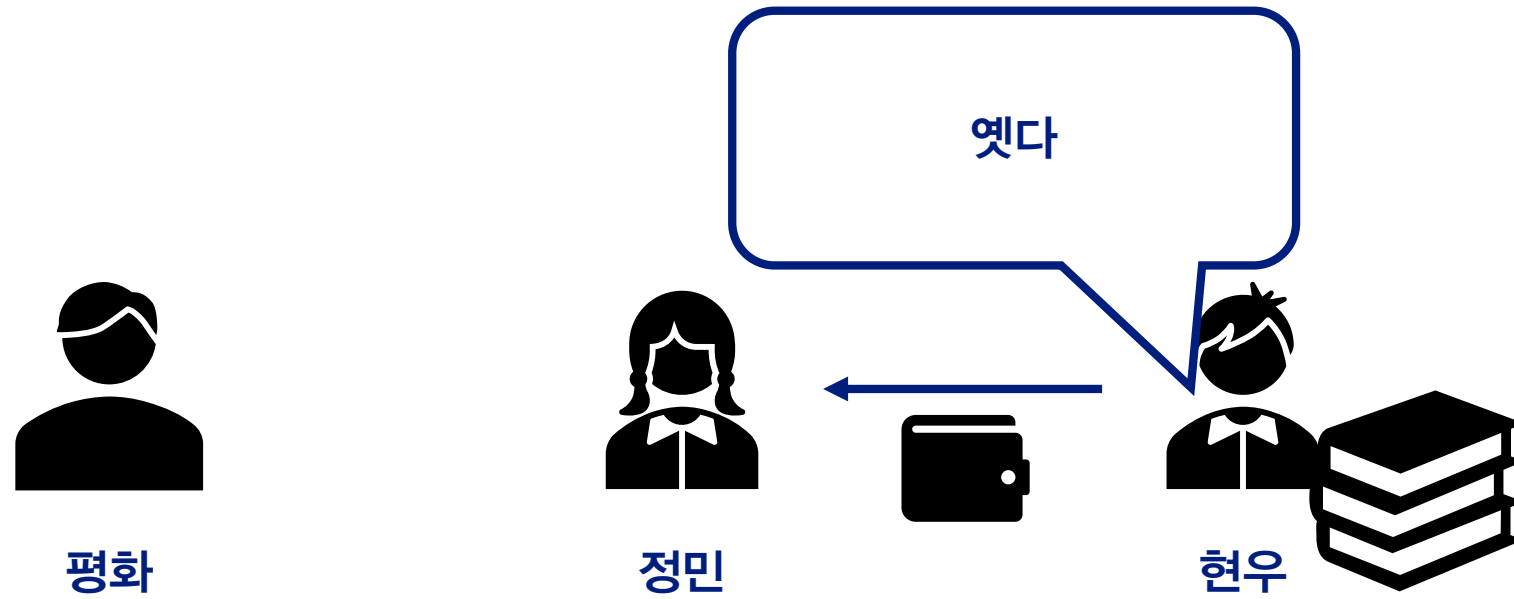


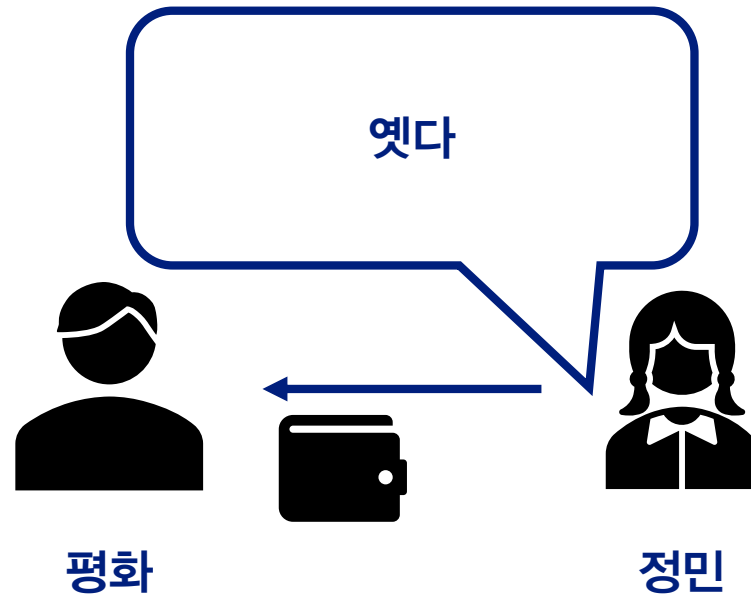
01 Server와 Client에 대한 이해

먼저 큰 틀부터 볼게요!





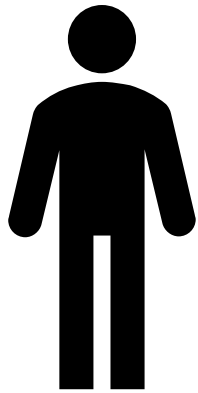






Server와 Client에 대한 이해

deSigner
develop
Oper
Planner
sopT



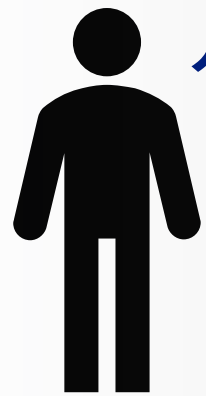
사용자



Client

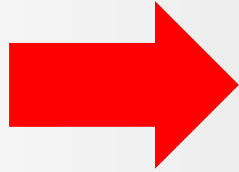


Server



사용자

사용자의 Action



로그인 시도

회원가입

웹툰 사이트

쇼핑...

Client



Server



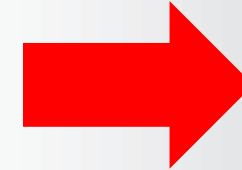
사용자



Client



데이터 요청



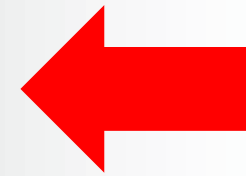
Server



사용자



Client



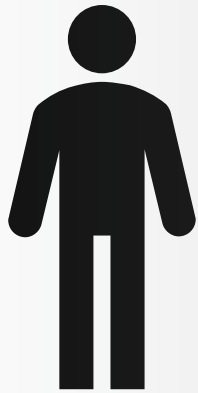
데이터 응답



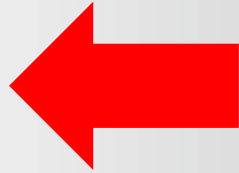
Server



데이터를 사용자에게 보여줌



사용자



Client



Server

조금 더 자세히 봅시다!



사용자

사용자의 Action

로그인 시도, 회원가입, 웹툰 사이트, 쇼핑...



- 사용자의 클릭 이벤트 감지 및 해당 로직 처리
- 사용자에게 화면을 그려줌
- 로딩 화면을 보여줌
- 받은 데이터를 사용자가 보기 좋게 화면에 적절히 그려줌



Client





Client



서버에게 데이터 요청 / 전달



ex) 로그인을 위해 유저가 입력한 아이디, 비밀번호 데이터 전달

ex2) 인스타 새로운 소식 받아오도록 요청



Server



Client



서버로부터 응답 / 에러를 받음



ex) 로그인 성공 / 실패 여부

ex2) 인스타 피드 데이터



Server



Client



받은 데이터를 화면에 그려줌



사용자

ex) 로그인에 성공했습니다! 토스트

ex2) 인스타 새로운 사진 등등



이 과정에서...

서버와 클라이언트는 어떤 언어로 대화를 나눌까?

02 Json



Client



서버와 데이터 요청 및 수신



어떻게 이뤄지는가??!!

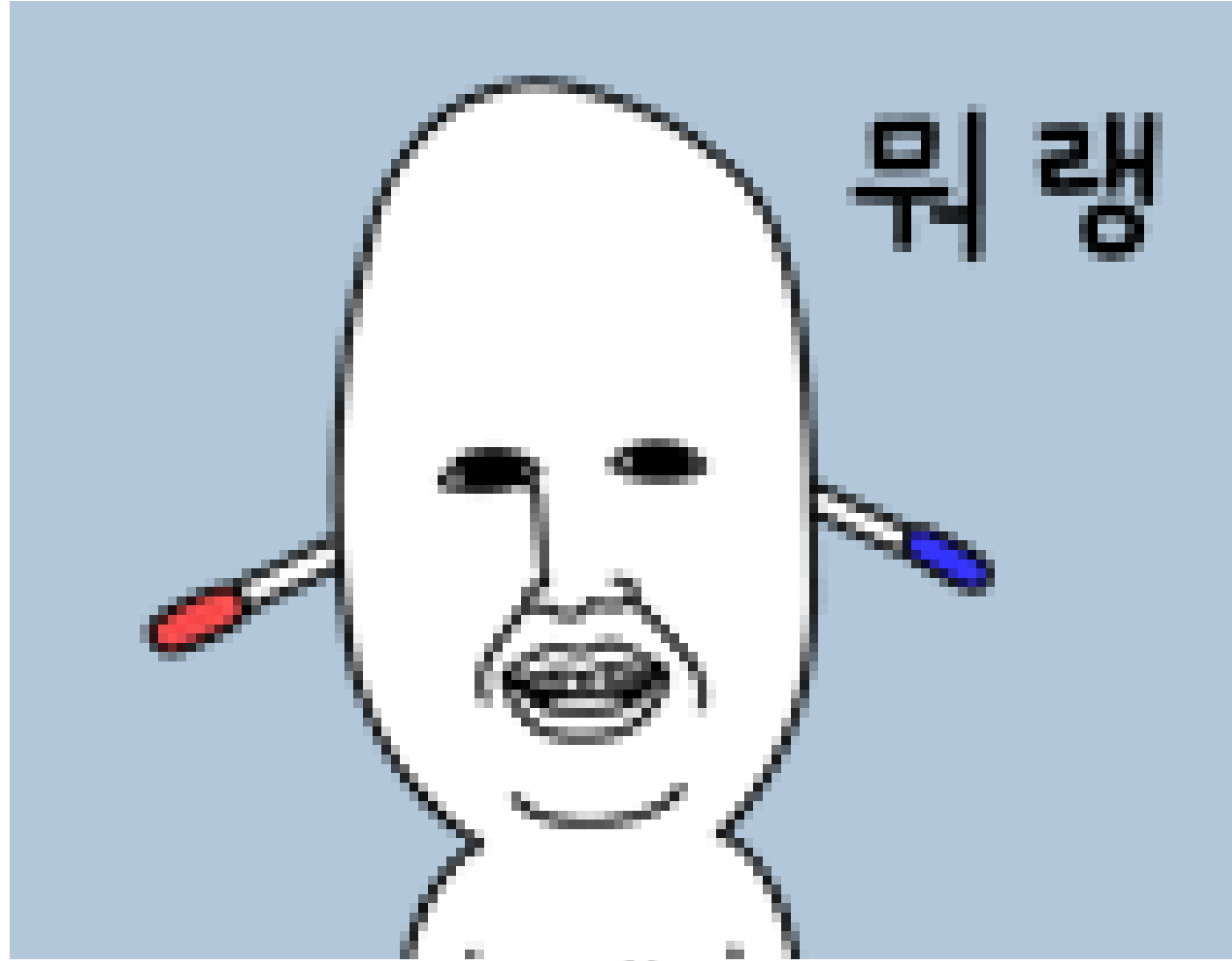


Server

Pouvez-vous comprendre cette phrase ?



Pouvez-vous comprendre cette phrase ?



이 문장을 이해할 수 있나요?



서로 다른 언어



데이터 다루는 방식이 다르다!





서로 다른 언어



데이터 다루는 방식이 다르다!



다 알아듣는 형태가 있을까?





서로 다른 언어



데이터 다루는 방식이 다르다!

Json



다 알아듣는 형태가 없을까?



```
{
  "이름" : "최호준" ,
  "나이" : 26,
  "성별" : "남" ,
  "취미" : { "1번" : "축구" , "2번" : "안드" }
}
```

Json 타입의 장점?

- 텍스트로 이루어져 있어 기계, 사람 모두 이해하기 쉽다.
- 프로그래밍 언어와 플랫폼에 독립적이기 때문에 서로 다른 시스템간에 객체를 교환하기 좋다.

기본 자료형

- Number (정수, 실수 전부다 지원)
- 문자열 (큰 따옴표로 묶인 문자열)
- Boolean (true / false)
- **배열**
- **객체**
- Null (비어있는 값)

*8진수나 16진수를 표현하는 방법은 지원되지 않는다!

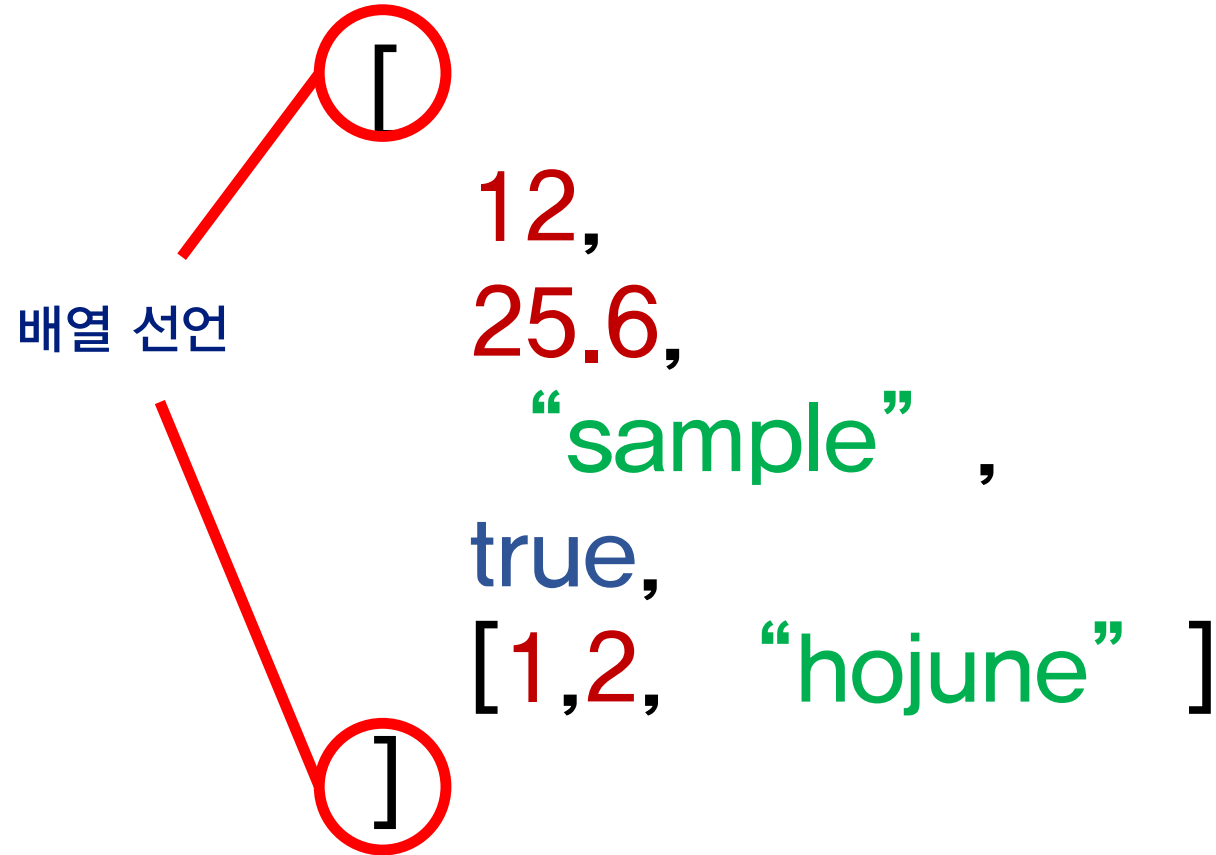
배열

- [] 대괄호로 정의한다
- 아무 자료형이나 들어갈 수 있다.

```
[
  12,
  25.6,
  "sample",
  true,
  [1, 2, "hojune"]
]
```

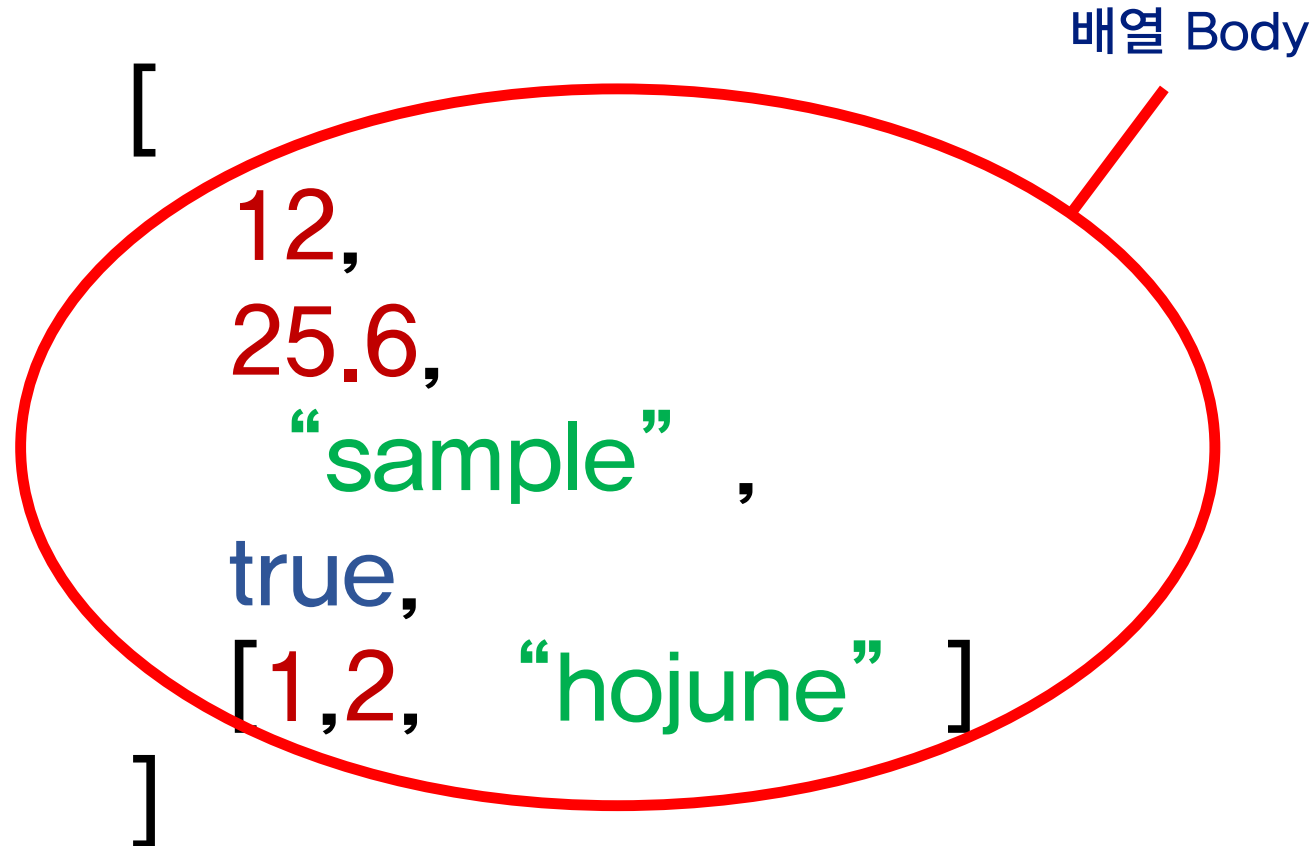
- [] 대괄호로 정의한다
- 아무 자료형이나 들어갈 수 있다.

배열



- [] 대괄호로 정의한다
- 아무 자료형이나 들어갈 수 있다.

배열



- [] 대괄호로 정의한다
- 아무 자료형이나 들어갈 수 있다.

배열

```
[
  12,
  25.6,
  "sample",
  true,
  [1, 2, "hojune"]
]
```

배열 안에 배열도 가능합니다!

객체

– {} 중괄호로 정의한다

– 키 값은 문자열이다.

– 키 값을 이용해서 실제 값에 접근 할 수 있다.

```
{
  "이름" : "최호준" ,
  "나이" : 26,
  "성별" : "남" ,
  "취미" : { "1번" : "축구" , "2번" : "안드" }
}
```



Json

- {} 중괄호로 정의한다
- 키 값은 문자열이다.
- 키 값을 이용해서 실제 값에 접근 할 수 있다.

객체

deSigner
develop
Oper
Planner
sopT

객체 선언

{

“이름” : “최호준” ,

“나이” : 26,

“성별” : “남” ,

“취미” : { “1번” : “축구” , “2번” : “안드” }

}



- {} 중괄호로 정의한다
- 키 값은 문자열이다.
- 키 값을 이용해서 실제 값에 접근 할 수 있다.

객체

객체 Body

```
{  
  "이름" : "최호준",  
  "나이" : 26,  
  "성별" : "남",  
  "취미" : { "1번" : "축구", "2번" : "안드" }  
}
```



- {} 중괄호로 정의한다
- 키 값은 문자열이다.
- 키 값을 이용해서 실제 값에 접근 할 수 있다.

객체

```
{
```

```
  "이름" : "최호준",
```

```
  "나이" : 26,
```

```
  "성별" : "남",
```

```
  "취미" : { "1번" : "축구", "2번" : "안드" }
```

```
}
```

객체를 값으로 가질 수 있다.

배열과 객체를 같이 만들어볼까요?



생각나는 치킨집 이름을 짝 나열해보세요!
치킨집마다 대표메뉴, 평점을 한번 적어봅시다.
이걸 Json으로 나타내면?



배열과 객체를 같이 만들어볼까요?

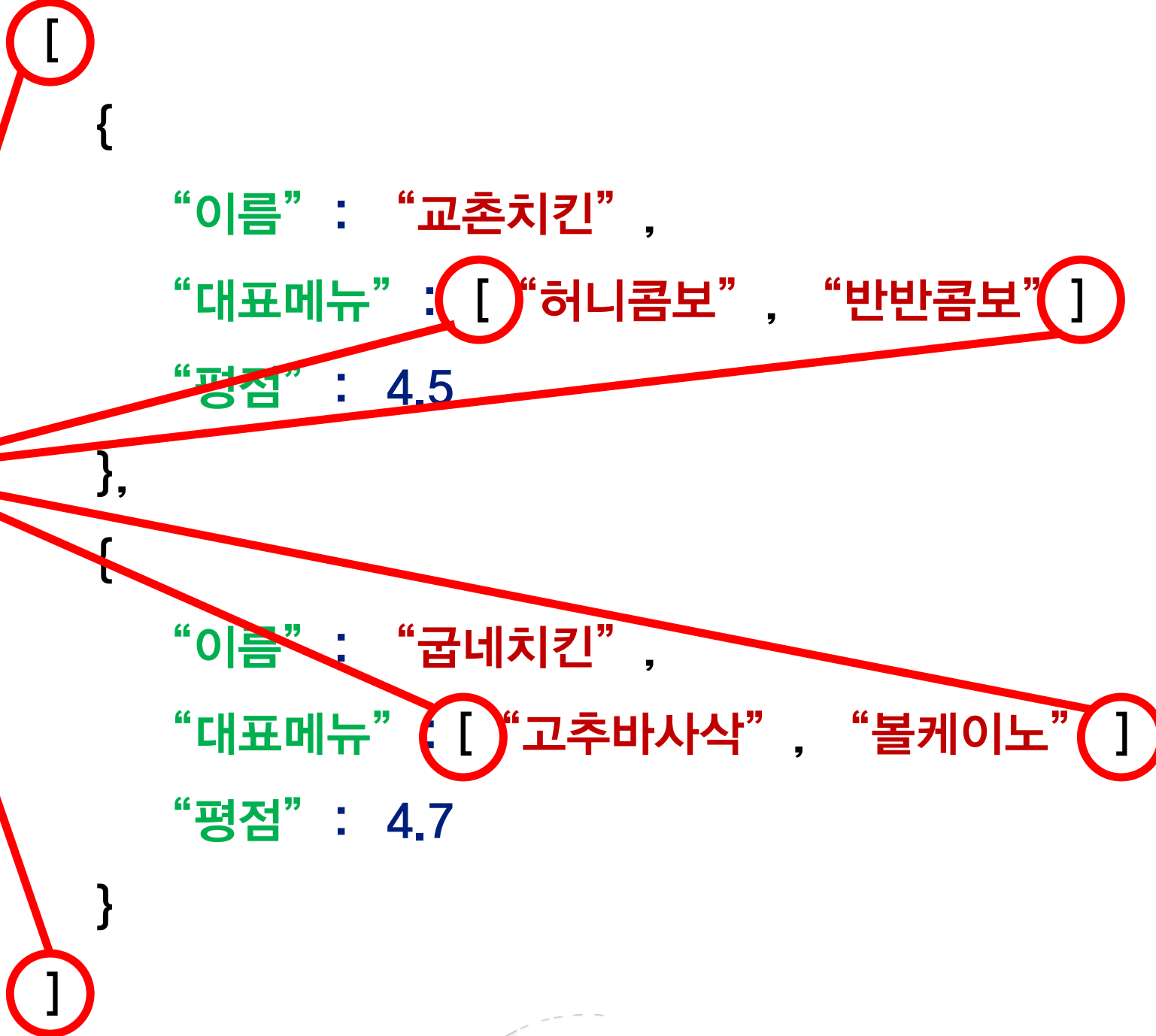
생각나는 치킨집 이름을 짝 나열해보세요!

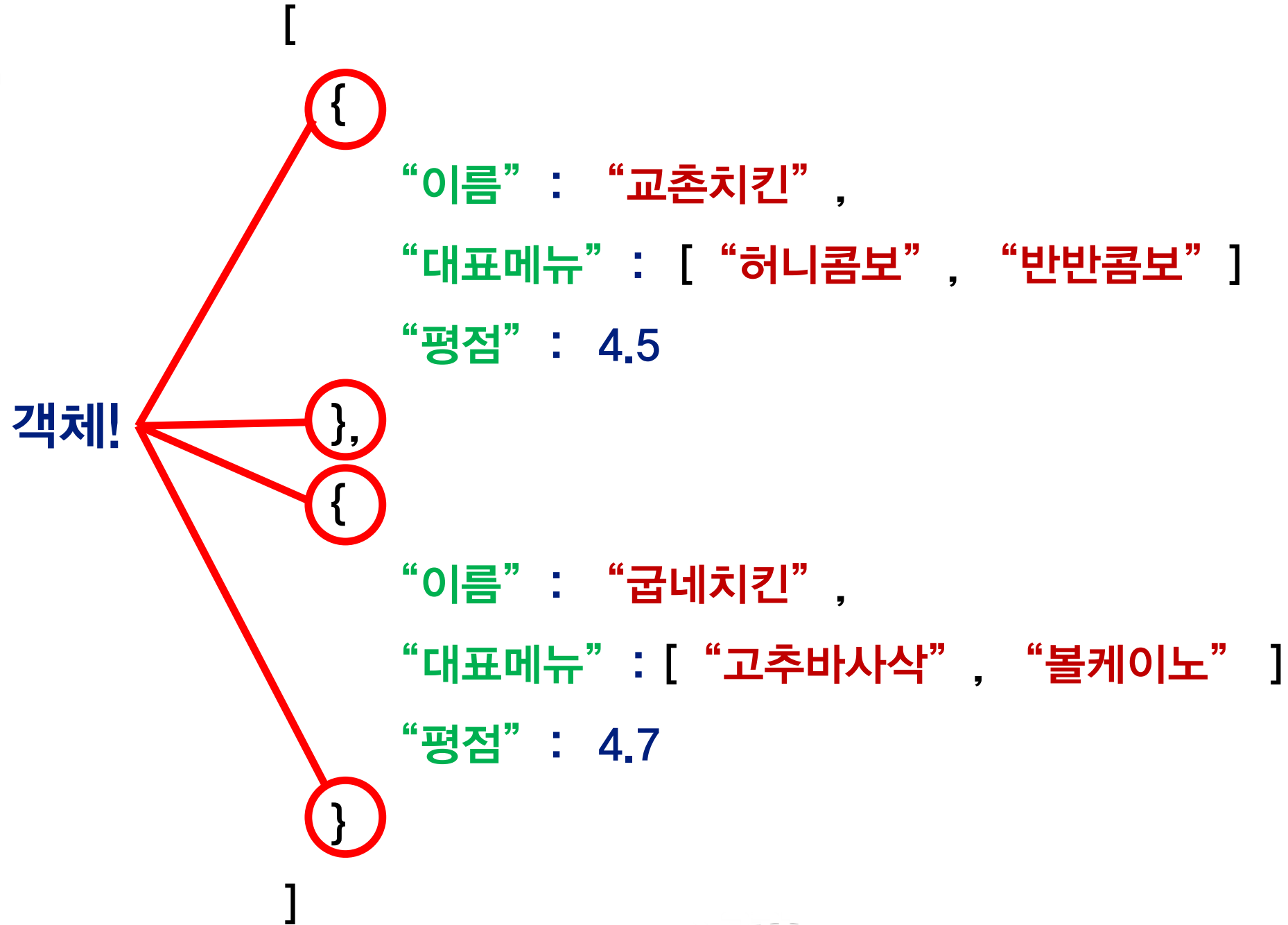
치킨집마다 대표메뉴, 평점을 한번 적어봅시다.

이걸 Json으로 나타내면?

```
[
  {
    "이름" : "교촌치킨" ,
    "대표메뉴" : [ "허니콤보" , "반반콤보" ]
    "평점" : 4.5
  },
  {
    "이름" : "굽네치킨" ,
    "대표메뉴" : [ "고추바사삭" , "볼케이노" ]
    "평점" : 4.7
  }
]
```

배열!





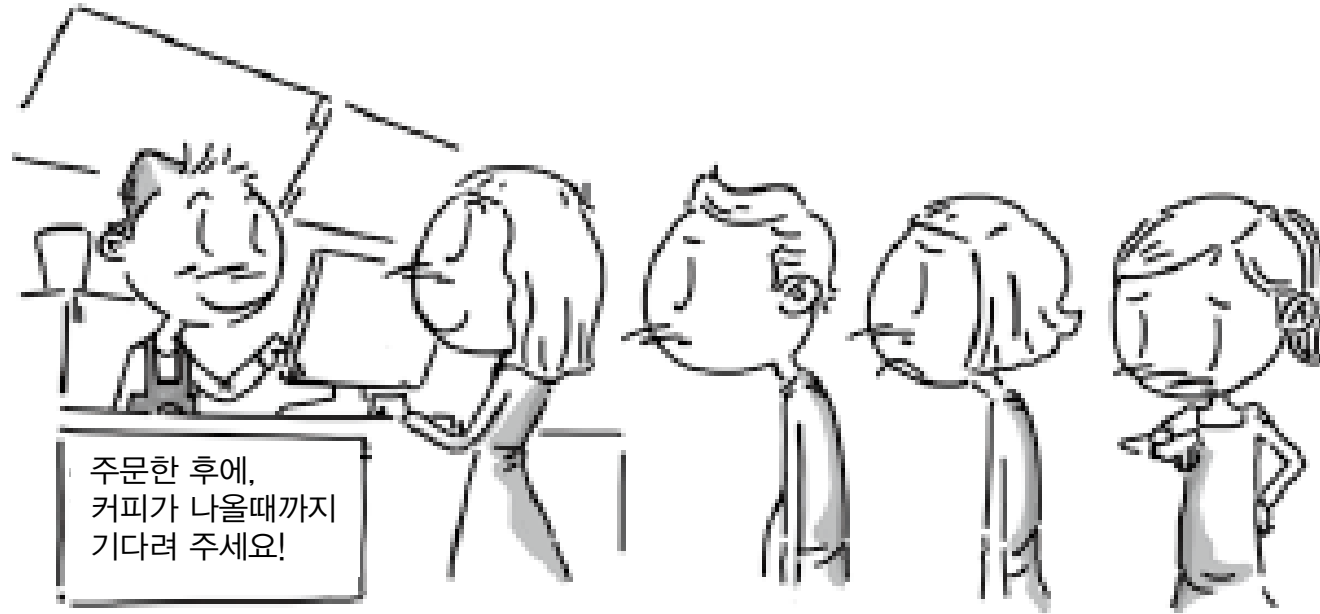

```
{
  "login": "octocat",
  "id": 1,
  "node_id": "MDQ6VXNlcjE=",
  "avatar_url": "https://github.com/images/error/octocat_happy.gif",
  "gravatar_id": "",
  "url": "https://api.github.com/users/octocat",
  "html_url": "https://github.com/octocat",
  "followers_url": "https://api.github.com/users/octocat/followers",
  "following_url": "https://api.github.com/users/octocat/following{/other_user}",
  "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/octocat/starred{/owner}/{/repo}",
  "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
  "organizations_url": "https://api.github.com/users/octocat/orgs",
  "repos_url": "https://api.github.com/users/octocat/repos",
  "events_url": "https://api.github.com/users/octocat/events{/privacy}",
  "received_events_url": "https://api.github.com/users/octocat/received_events",
  "type": "User",
  "site_admin": false,
  "name": "monalisa octocat",
  "company": "GitHub",
  "blog": "https://github.com/blog",
  "location": "San Francisco",
  "email": "octocat@github.com",
  "hireable": false,
  "bio": "There once was...",
  "public_repos": 2,
  "public_gists": 1,
  "followers": 20,
  "following": 0,
  "created_at": "2008-01-14T04:33:35Z",
  "updated_at": "2008-01-14T04:33:35Z"
}
```

이건 뭘까요?

맞습니다!
객체 입니다.

```
{
  "login": "octocat",
  "id": 1,
  "node_id": "MDQ6VXNlcjE=",
  "avatar_url": "https://github.com/images/error/octocat_happy.gif",
  "gravatar_id": "",
  "url": "https://api.github.com/users/octocat",
  "html_url": "https://github.com/octocat",
  "followers_url": "https://api.github.com/users/octocat/followers",
  "following_url": "https://api.github.com/users/octocat/following{/other_user}",
  "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/octocat/starred{/owner}/{/repo}",
  "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
  "organizations_url": "https://api.github.com/users/octocat/orgs",
  "repos_url": "https://api.github.com/users/octocat/repos",
  "events_url": "https://api.github.com/users/octocat/events{/privacy}",
  "received_events_url": "https://api.github.com/users/octocat/received_events",
  "type": "User",
  "site_admin": false,
  "name": "monalisa octocat",
  "company": "GitHub",
  "blog": "https://github.com/blog",
  "location": "San Francisco",
  "email": "octocat@github.com",
  "hireable": false,
  "bio": "There once was...",
  "public_repos": 2,
  "public_gists": 1,
  "followers": 20,
  "following": 0,
  "created_at": "2008-01-14T04:33:35Z",
  "updated_at": "2008-01-14T04:33:35Z"
}
```

03 동기과 비동기





동기



비동기

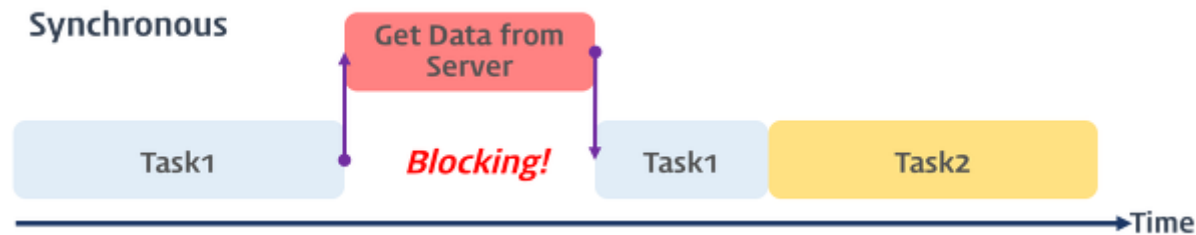


동기(Synchronous Processing model)

직렬적으로 태스크를 수행한다.

어떤 작업이 수행 중이면 **다음 작업은 대기**하게 된다.

Ex) 서버에서 데이터를 가져와 화면에 표시하는 작업 수행할 때,
서버에 데이터를 요청하고 데이터가 응답될 때까지 **이후 태스크들은 블로킹(작업 중단)**된다.

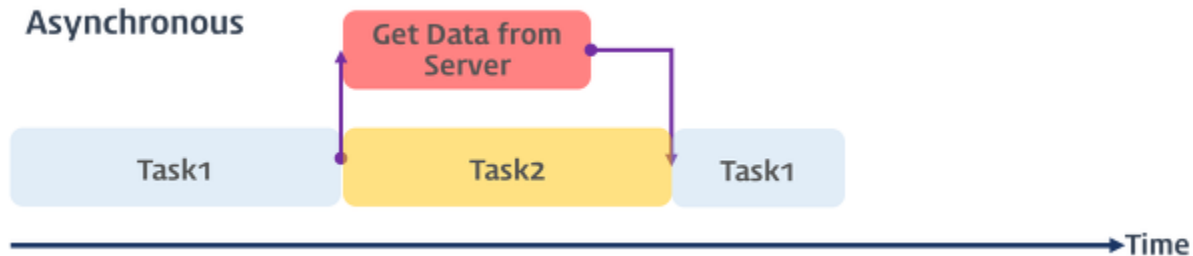




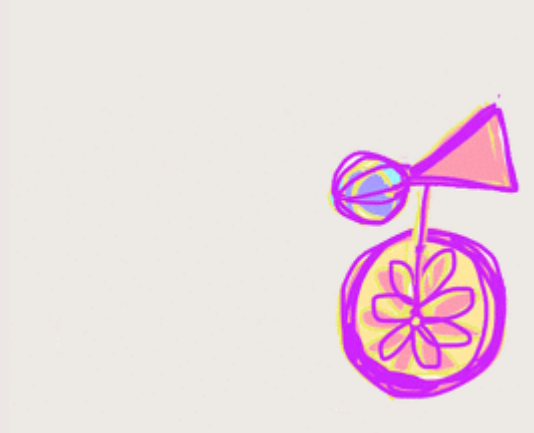
비동기 (Asynchronous Processing Model)

병렬적으로 태스크를 수행한다.

어떤 작업이 수행 중이더라도 **다음 태스크를 실행**한다.
Ex) 서버에서 데이터를 가져와 화면에 표시하는 작업 수행할 때,
서버에 데이터를 요청하고 데이터가 응답될 때까지 **대기하지 않고 즉시 다음 태스크를 수행**한다.
이후 서버로부터 데이터가 응답되면 이벤트가 발생한다.



동기와 비동기



동기와 비동기



달린다()

폭죽()

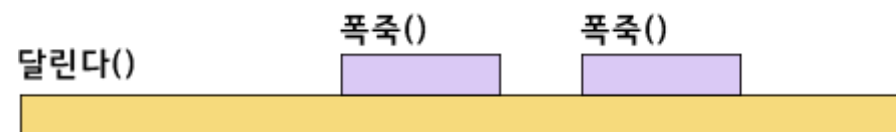
달린다()

폭죽()

달린다()



동기와 비동기




04 Restful API

Representational **S**tate **T**ransfer – HTTP 장점을 최대한 활용할 수 있는 아키텍처.
서버에 존재하는 데이터에 접근하기 위한 대표적인 규칙!!!

1. **자원(Resource)** – URI (http://서버 이름 / 자원 식별)

2. **행위(Verb)** – HTTP Method

- 1. GET : 데이터 자체 단순 요구
- 2. POST : 데이터 제출, 서버의 상태 변화
- 3. PUT : 데이터 변경
- 4. DELETE : 데이터 삭제

An aerial photograph of a densely packed urban area, likely a city center. The image shows a vast number of multi-story buildings, mostly with light-colored facades and red-tiled roofs. The buildings are closely packed together, creating a complex, textured pattern of rooftops and windows. The lighting suggests it might be late afternoon or early morning, with some shadows visible. The overall scene conveys a sense of a highly populated and built-up environment.

1. 자원(Resource) – URI (<http://서버 이름 / 자원 식별>)

1. 자원(Resource) – URI (http://서버 이름 / 자원 식별)

건물 A 주소



1. 자원(Resource) – URI (http://서버 이름 / 자원 식별)



303호

그 밖의 여러 API들.. [네이버 API](#)

네이버 아이디로 로그인 [↗](#)

다음은 네이버 아이디로 로그인 API에서 사용하는 주요 요청 URL과 메서드, 응답 형식입니다.

요청 URL	메서드	응답 형식	설명
<code>https://nid.naver.com/oauth2.0/authorize</code>	GET/POST	-	네이버 아이디로 로그인 인증을 요청합니다.
<code>https://nid.naver.com/oauth2.0/token</code>	GET/POST	JSON	접근 토큰의 발급, 갱신, 삭제를 요청합니다.
<code>https://openapi.naver.com/v1/nid/me</code>	GET	JSON	네이버 회원의 프로필을 조회합니다.

그 밖의 여러 API들.. [카카오 API](#)

서버에 어떤 용도의 요청을 할 것인가?

Request

URL

서버에 어떤 서비스 요청할 것인가?

```
GET /v3/search/book HTTP/1.1
Host: dapi.kakao.com
Authorization: KakaoAK {app_key}
```

Parameter

서버에 어떤 정보 보낼 것인가?

Name	Type	Description	Required
query	String	검색을 원하는 질의어	O
sort	String	결과 문서 정렬 방식, accuracy(정확도순) 또는 recency(최신순), 기본 값 accuracy	X
page	Integer	결과 페이지 번호, 1~50 사이의 값, 기본 값 1	X
size	Integer	한 페이지에 보여질 문서 수, 1~50 사이의 값, 기본 값 10	X
target	String	검색 필드 제한 사용 가능한 값: title(제목), isbn (ISBN), publisher(출판사), person(인명)	X

그 밖의 여러 API들.. 깃허브 API

서버에 어떤 용도의 요청을 할 것인가?

All API access is over HTTPS, and accessed from `https://api.github.com`. All data is sent and received as JSON.

GET users

서버에 어떤 서비스 요청할 것인가?

BASE_URL : 깃 허브 API 접근 위한 도메인

Parameters

Name	Type	Description
since	string	The integer ID of the last User that you've seen.

서버에 어떤 정보 보낼 것인가?

Response

Status: 200 OK
Link: <https://api.github.com/users?since=135>; rel="next"

```
[
  {
    "login": "octocat",
    "id": 1,
    "node_id": "MDQ6VXN1c2E=",
    "avatar_url": "https://github.com/images/error/octocat_happy.gif",
    "gravatar_id": "",
    "url": "https://api.github.com/users/octocat",
    "html_url": "https://github.com/octocat",
    "followers_url": "https://api.github.com/users/octocat/followers",
    "following_url": "https://api.github.com/users/octocat/following{/other_user}",
    "gists_url": "https://api.github.com/users/octocat/gists{/gist_id}",
    "starred_url": "https://api.github.com/users/octocat/starred{/owner}/{/repo}",
    "subscriptions_url": "https://api.github.com/users/octocat/subscriptions",
    "organizations_url": "https://api.github.com/users/octocat/orgs",
    "repos_url": "https://api.github.com/users/octocat/repos",
    "events_url": "https://api.github.com/users/octocat/events{/privacy}",
    "received_events_url": "https://api.github.com/users/octocat/received_events",
    "type": "User",
    "site_admin": false
  }
]
```

어떤 응답을 받을 수 있는가?

✓ BASE URL : 13.209.144.115:3333/

서버에 어떤 용도의 요청을 할 것인가?

서버에 어떤 서비스 요청할 것인가?

서버에 어떤 정보 보낼 것인가?

```
{
  "id": "gngsn",
  "password": "qwerty"
}
```

- id : 사용자 아이디
- password : 사용자 비밀번호

< Success >

```
{
  "status": 200,
  "success": true,
  "message": "로그인 성공",
  "data": {
    "jwt": "eyJhbGciOiJIU"
  }
}
```

어떤 응답을 받을 수 있는가?

POSTMAN을 이용해서 로그인 부분을 한 번 테스트 해봅시다!

로그인 API 문서 여길 보시면서

영상 1. postman 이용하기 를 참고해주세요.

포스트맨을 설치하지 않았다면? 포스트맨 설치

gyeongseon edited this page 17 hours ago · 4 revisions

✓ BASE URL : 13.209.144.115:3333/

유저

- `/user/signup` - 회원가입
- `/user/signin` - 로그인

▼ Pages 4

Find a Page...[Home](#) 로그인

✓ 회원가입

SOPT_TEST_API

Clone this wiki locally

<https://github.com/gngsn/26>

05 Retrofit의 활용

0. 서버와 필요한 데이터 논의

1. 라이브러리 추가

2. API문서 보고 Request / Response 객체 설계

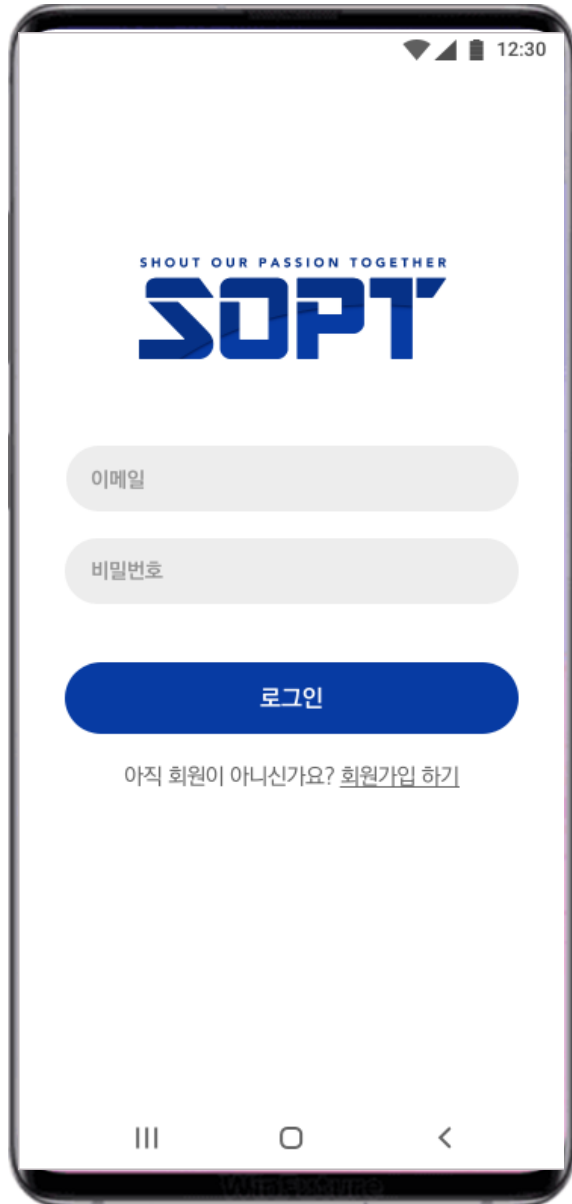
[Retrofit 공식 문서](#)

3. Retrofit Interface 설계

[로그인 API 문서](#)

4. Retrofit Interface 실제 구현체 만들기

5. Callback 등록하며 통신 요청



무엇을 줄 것이고,
무엇을 받을 것인가?
를 논의하면 됩니다!

아이디, 비밀번호를 넘겨 줄거야.
성공 여부를 받아야겠어.



아이디, 비밀번호를 넘겨 줄거야.

성공 여부를 받아야겠어.

Request Header

```
{  
  "Content-Type": "application/json"  
}
```

넘겨줄 때 양식이라고 생각하면 됩니다.
Json타입

Request Body

```
{  
  "id": "gngsn",  
  "password": "qwerty"  
}
```

- id : 사용자 아이디
- password : 사용자 비밀번호



아이디, 비밀번호를 넘겨 줄거야.

성공 여부를 받아야겠어.

Response

< Success >

```
{
  "status": 200,
  "success": true,
  "message": "로그인 성공",
  "data": {
    "jwt": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZHIjOiJm5hbWUiOiJnbmdzbiIsIm1hdCI6MT
  }
}
```

라이브러리 추가하기!

```
//Retrofit 라이브러리 : https://github.com/square/retrofit  
implementation 'com.squareup.retrofit2:retrofit:2.6.2'  
  
//Retrofit 라이브러리 응답으로 가짜 객체를 만들기 위해  
implementation 'com.squareup.retrofit2:retrofit-mock:2.6.2'  
  
  
//객체 시리얼라이즈를 위한 Gson 라이브러리 :  
https://github.com/google/gson  
implementation 'com.google.code.gson:gson:2.8.6'  
  
//Retrofit 에서 Gson 을 사용하기 위한 라이브러리  
implementation 'com.squareup.retrofit2:converter-gson:2.6.2'
```

AndroidManifest.xml에 아래 내용 추가해주세요!

```
<uses-permission android:name="android.permission.INTERNET" />  
  
<application  
    android:usesCleartextTraffic="true"
```

0. 서버와 필요한 데이터 논의

1. 라이브러리 추가

2. API문서 보고 Request / Response 객체 설계

[Retrofit 공식 문서](#)

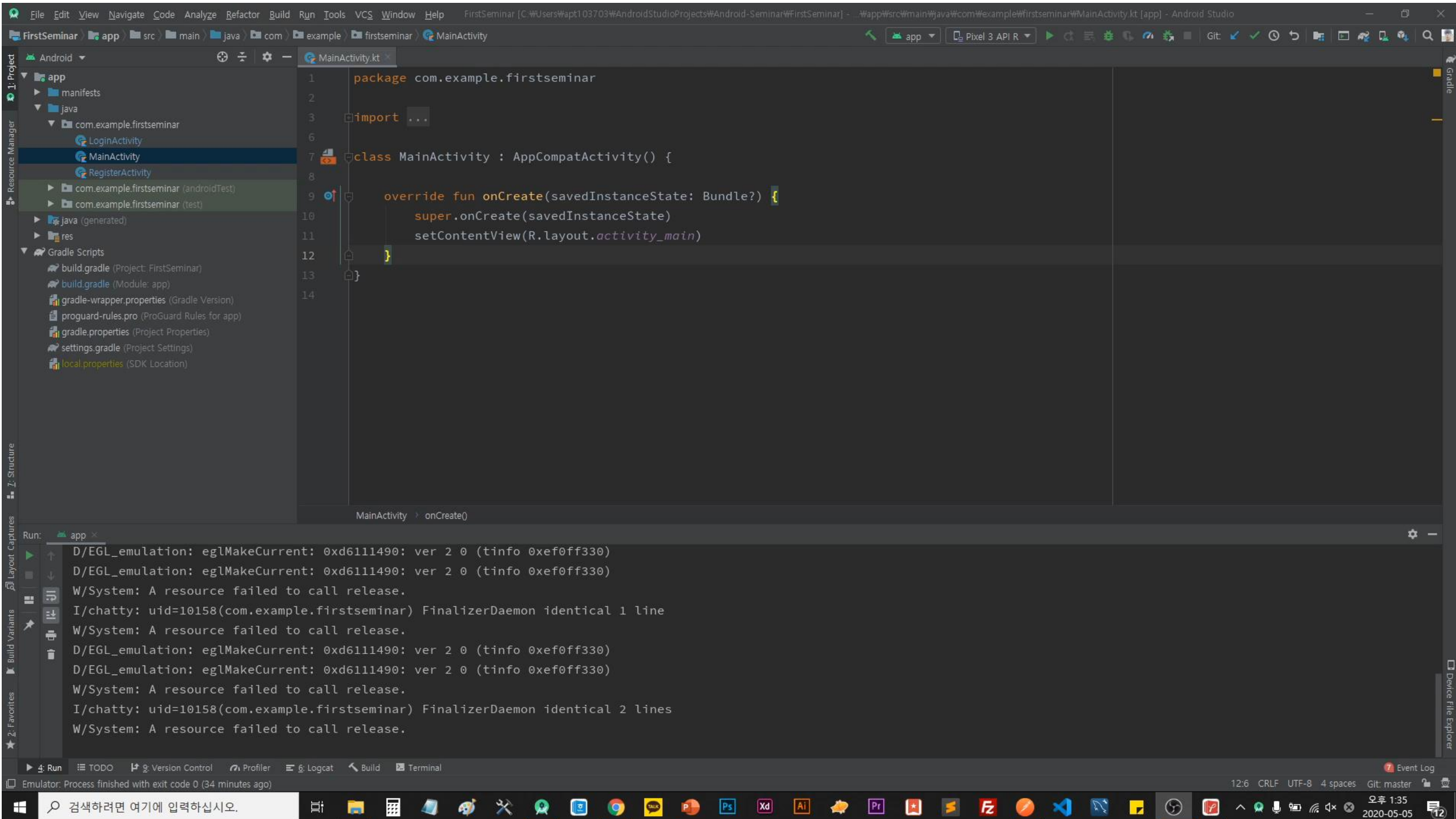
3. Retrofit Interface 설계

[로그인 API 문서](#)

4. Retrofit Interface 실제 구현체 만들기

5. Callback 등록하며 통신 요청

영상 2. 설계 작업을 참고해주세요



서버에서 정한 변수 말고 제가 이름을 지어주고 싶어요!
서버의 응답 변수를 다 선언해주어야 하나요?

서버에서 정한 변수 말고 제가 이름을 지어주고 싶어요!

서버의 응답 변수를 다 선언해주어야 하나요?

```
data class ResponseLogin(  
    val status : Int,  
    val success : Boolean,  
    val message : String,  
    val data : SomeData?  
)
```

```
data class ResponseLogin(  
    val status : Int,  
    val success : Boolean,  
    @SerializedName("data")  
    val responseData : SomeData?  
)
```


SerializedName으로 서버에서 지어준 이름을 지정해주면
responseData 처럼 원하는 변수 명으로 바뀌서 이용이 가능합니다!

서버에서 정한 변수 말고 제가 이름을 지어주고 싶어요!

서버의 응답 변수를 다 선언해주어야 하나요?

```
data class ResponseLogin(  
    val status : Int,  
    val success : Boolean,  
    @SerializedName("data")  
    val responseData : SomeData?  
)
```

```
{  
    "status": 200,  
    "success": true,  
    "message": "로그인 성공",  
    "data": {  
        "jwt": "eyJhbGciOiJIU  
    }  
}
```



@SerializedName으로 서버에서 지어준 이름을 지정해주면
responseData 처럼 원하는 변수 명으로 바뀌어서 이용이 가능합니다!

서버에서 정한 변수 말고 제가 이름을 지어주고 싶어요!

서버의 응답 변수를 다 선언해주어야 하나요?

```
data class ResponseLogin(  
    val status : Int,  
    val success : Boolean,  
    val message : String,  
    val data : SomeData?  
)
```

```
data class ResponseLogin(  
    val status : Int,  
    val success : Boolean,  
    @SerializedName("data")  
    val responseData : SomeData?  
)
```

서버에서 특정 데이터를 받아오고 싶지 않으면
굳이 응답 객체 내에 변수를 만들지 않아도 됩니다!

0. 서버와 필요한 데이터 논의

1. 라이브러리 추가

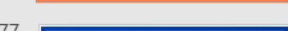
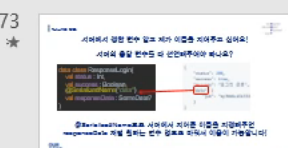
2. API문서 보고 Request / Response 객체 설계 [Retrofit 공식 문서](#)

3. Retrofit Interface 설계 [로그인 API 문서](#)

4. Retrofit Interface 실제 구현체 만들기

5. Callback 등록하며 통신 요청

영상 3. interface 실제 구현체 만들기 를 참고해주세요.



de**S**igner
devel**O**per
Planner
sop**T**

Retrofit 공식 문서

로그인 API 문서

5. Callback 등록하며 통신 요청

영상 3. interface 실제 구현체 만들기를 참고해주세요

**OUR
SOFT**

- ☒ 단색 채우기(S)
- ☐ 그라데이션 채우기(G)
- ☐ 그림 또는 질감 채우기(P)
- ☐ 패턴 채우기(A)
- ☐ 배경 그래픽 숨기기(H)

색(C)

투명도(T) 1 0%

모두 적용(L) 배경 원래대로(B)

싱글톤. 코드 아무데서나 접근 가능하고, 객체는 단 하나만!

```
object RequestToServer{  
    var retrofit = Retrofit.Builder()  
        .baseUrl("http://13.209.144.115:3333")  
        .addConverterFactory(GsonConverterFactory.create())  
        .build()  
  
    var service: RequestInterface =  
        retrofit.create(RequestInterface::class.java)  
}
```

Retrofit으로 받아온 json 데이터를
데이터 클래스로 변환 하기 쉽게 해줌!

retrofit 객체의 create 호출.
Interface 클래스 타입을 넘겨
실제 구현체를 만들어준다.

0. 서버와 필요한 데이터 논의

1. 라이브러리 추가

2. API문서 보고 Request / Response 객체 설계 [Retrofit 공식 문서](#)

3. Retrofit Interface 설계

[로그인 API 문서](#)

4. Retrofit Interface 실제 구현체 만들기

5. Callback 등록하며 통신 요청

영상 4. 통신하기 를 참고해주세요.

response.*isSuccessful*

여기 마우스 가져가서
Ctrl + 클릭

```
/** Returns true if {@link #code()} is in the range [200..300). */
```

이런 정보들이 적혀있습니다.

isSuccessful은 statusCode가 200~300 사이 일때 true를 리턴합니다.

→ 클라 요청이 잘 되서 응답을 받아 왔구나~!

궁금한 코드들이 생기면 이용해보세요!

Call<Type> 비동기적으로 Type을 받아오는 객체

Callback<Type> Type 객체를 받아왔을 때, 프로그래머가 할 행동

```
requestToServer.service.requestLogin(
    RequestLogin(
        id = et_id.text.toString(),
        password = et_password.text.toString()
    ) //로그인 정보를 전달
```

1. Call 타입이 리턴됨.

2. 실제 서버 통신을 비동기적으로 요청

```
).enqueue(object : Callback<ResponseLogin> { //Callback등록. Retrofit의 Callback을 import 해줘야 함!
```

```
    override fun onFailure(call: Call<ResponseLogin>, t: Throwable) {
        //통신 실패
    }
```

3. 비동기 요청 후 응답을 받았을 때 수행할 행동이 정의된 곳

```
    override fun onResponse(
        call: Call<ResponseLogin>,
        response: Response<ResponseLogin>
    ) {
        //통신 성공
        if(response.isSuccessful) { //statusCode가 200~300 사이일때. 응답 body 이용 가능
            if(response.body()!!.success) { //ResponseLogin의 success가 true인 경우 -> 로그인
                Toast.makeText(this@LoginActivity, "로그인 성공", Toast.LENGTH_SHORT).show()
                val intent = Intent(this@LoginActivity, MainActivity::class.java)
                startActivity(intent)
                finish()
            } else {
                Toast.makeText(this@LoginActivity, "아이디/비밀번호를 확인하세요!", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

```
.enqueue(object :Callback<ResponseLogin>{ //Callback등록. Retrofit의 Callback을 import 해줘야 함!
    override fun onFailure(call: Call<ResponseLogin>, t: Throwable) {
        //통신 실패
    }
}
```

```
    override fun onResponse(
        call: Call<ResponseLogin>,
        response: Response<ResponseLogin>
    ){
        //통신 성공
        if(response.isSuccessful) { //statusCode가 200~300 사이일때. 응답 body 이용 가능
            if(response.body()!!.success){ //ResponseLogin의 success가 true인 경우 -> 로그인
                Toast.makeText(this@LoginActivity,"로그인 성공",Toast.LENGTH_SHORT).show()
                val intent = Intent(this@LoginActivity, MainActivity::class.java)
                startActivity(intent)
                finish()
            }else{
                Toast.makeText(this@LoginActivity,"아이디/비밀번호를 확인하세요!",Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

과제 안내

필수!

Retrofit 공식 문서 **[기본 과제 1]** 로그인 API 문서

동일한 문서를 활용하여 회원가입 부분 통신을 해봅시다!
(회원가입 UI부분 부터 서버에 맞게 수정하고 진행해주세요!)

Hint) 아이디, 비밀번호, 이름, 이메일, 전화번호를 서버에 전달해 줘야합니다!

필수!

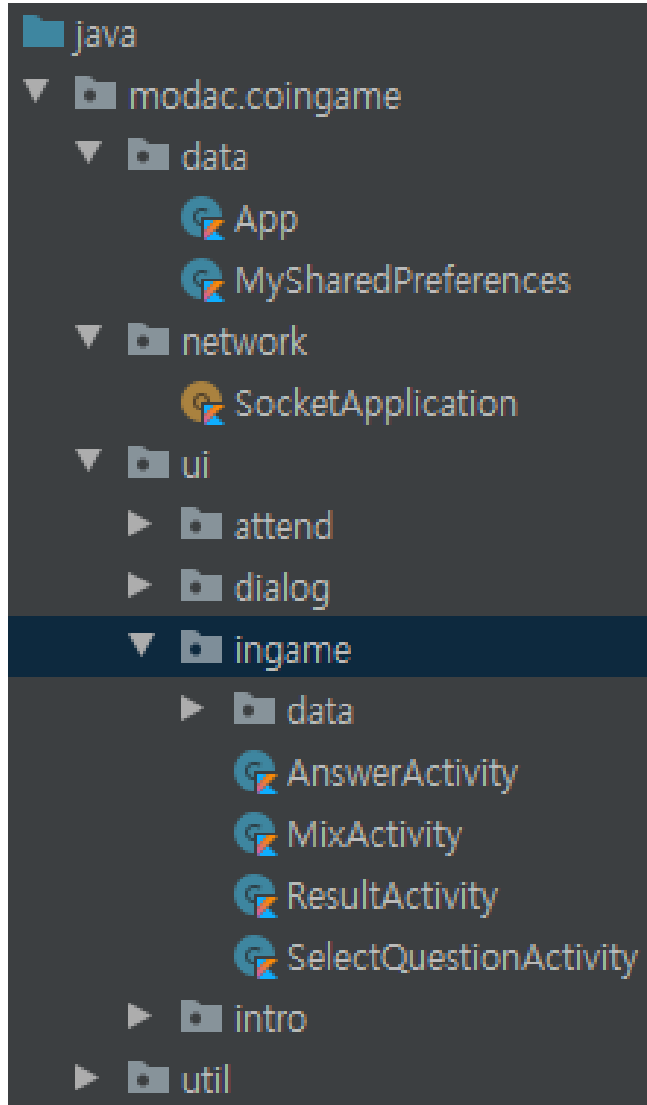
Retrofit 공식 문서

[기본 과제 2]

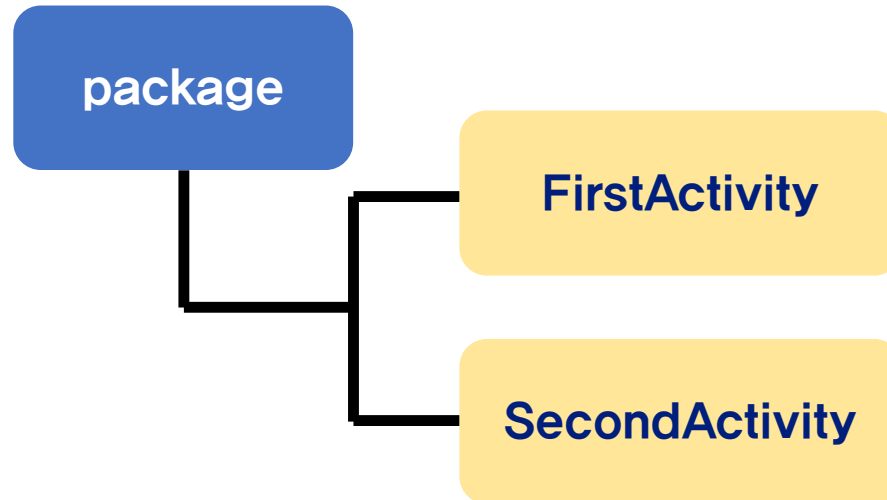
3차 세미나까지 진행 한 내용을 하나의 프로젝트로 합쳐주세요
(복붙을 하면서 하셔도 좋습니다. 전체 윤곽을 파악하세요!!!)

- 패키지 이름 : 첫 글자부터 소문자
- 클래스/인터페이스 이름 : 첫 글자가 대문자.

[기본 과제 2]



파일들이 많이 생기면 package를 통해 폴더처럼 관리할 수 있습니다!
패키지는 소문자, 클래스는 첫 글자가 대문자인 것이 관례입니다.



[Retrofit 공식 문서](#) **[성장 과제]** [카카오 채 검색 API](#)

카카오 OPEN API를 이용해 채 리스트 불러오기!

URL 다루기

요청 URL은 동적으로 부분 치환 가능하며, 이는 메소드 매개변수로 변경이 가능합니다. 부분 치환은 영문/숫자로 이루어진 문자열을 { 와 } 로 감싸 정의해줍니다. 반드시 이에 대응하는 @Path 를 메소드 매개변수에 명시해줘야합니다.

```
@GET("/group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId);
```

쿼리 매개변수도 명시 가능합니다.

```
@GET("/group/{id}/users")
Call<List<User>> groupList(@Path("id") int groupId, @Query("sort") String sort);
```


카카오 OPEN API를 이용해 책 리스트 불러오기!



검색어 입력하면 해당 리스트를 받아 와 띄우도록!

Hint) 리사이클러뷰 사용, 아래 코드 참고

```
@Headers("Authorization: KakaoAK 앱Key")
@GET("/v3/search/book")
fun requestSearchBook(
    @Query("query")bookTitle : String
) : Call<BookData>
```

과제 제출 기한 : 5월 22일 금요일 23시 59분 까지

과제 올린 깃허브 레포지토리 주소 링크 조장님께서
취합하여 파트장에게 전달해주시면 됩니다!

chjune0205@gmail.com

OUR
SOPT

SHOUT OUR PASSION TOGETHER

THANK YOU!!

SHOUT OUR PASSION TOGETHER
SOPT