SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPT O

06차 안드로이드 세미나

SHOUT OUR PASSION TOGETHER
SOPT

SHOUT OUR PASSION TOGETHER SOPT

세미나 실습 사전 준비!

- 통신 실습에 이용할 라이브러리 세 개 (복붙 해도 됨)

```
implementation 'com.github.bumptech.glide:glide:4.8.0'
implementation 'com.squareup.retrofit2:retrofit:2.4.0'
implementation 'com.squareup.retrofit2:converter-gson:2.4.0'
```

- anko 라이브러리 (복붙 해도 됨)

```
implementation 'org.jetbrains.anko:anko:0.10.7'
```

- RecyclerView 라이브러리

  File → Project Structure → app탭의 Dependencies에서 추가해주세요!!

  본인 버전 종속 관계에 맞춰서 라이브러리가 등록되도록!!

```
implementation 'com.android.support:recyclerview-v7:27.1.1'
```

- **MainActivity, SignUpActivity, BoardActivity, WriteActivty 만들기**

Contents

01

키워드 정리

SHOUT OUR PASSION TOGETHER SOPT

오늘 배울 내용 키워드들은!

- **HTTP**
- **REST API**

- Retrofit2
- JSON / GSON
- Glide

HTTP?

- Hypertext Transfer Protocol의 약자로 **하이퍼텍스트를 주고 받는 방식**에 대한 **약속/규칙**이다.
  쉽게 말해, 하이퍼텍스트 기반(**링크 기반**)으로 데이터를 주고 받는다고 생각하자!

- 가장 성공적인 인터넷 프로토콜.

- 클라이언트가 링크를 통해서 서버로 Request를 보내고,
  서버는 클라이언트에게 Request를 받고 그에 해당하는 Response를 보낸다.

REST API

- Representational State Transfer의 약자로, 2000년도에 웹(Http) 설계의 우수성에 비해 제대로 활용되지 못한 모습을 안타깝게 생각하여 웹의 장점을 최대한 살려서 활용할 수 있는 아키텍쳐로 “로이 필딩”씨가 만드셨습니다.

- REST는 자원, 행위, 표현으로 구성되어 있습니다.
  - 자원 : URI (참고로 URL은 URI에 포함, Uniform Resource Identifier/Locator)
  - 행위 : HTTP Method(GET, POST, PUT, DELETE)를 통해 조회, 삽입, 수정, 삭제를 수행한다.
  - 표현 : 예시) http://restapi.example.com/sports/soccer/players/13
- 웹에 존재하는 모든 자원(이미지, 동영상, DB자원 등)에 직관적인 URI을 부여하여 활용하는 것으로, 자원을 정의하고 자원에 대한 주소를 지정하는 방법론…

METHOD	역할
POST	데이터를 BODY에 숨겨서 서버에 전달하는 방식
GET	URI/URL에 데이터를 포함시켜서 서버에 전달하는 방식
PUT	데이터 업데이트 시 사용
DELETE	데이터 삭제 시 사용

SHOUT OUR PASSION TOGETHER SOPT

통신에 쓰일 라이브러리

- **Retrofit2**
  - HTTP REST API를 자바(코틀린) 인터페이스로 제공

    ➔ 통신 안전하게, 쉽게, 간편하게 할 수 있다.
- **GSON**
  - JSON 스키마에 맞춰서 객체를 생성해준다! (Json ⟷ Object 변환)

```
{
    "name": "남윤환",
    "age": 28
}
```

<JSON>

```
data class User(
        val name : String,
        var age : Int
)
```

```
User("남윤환", 28)
```

<Object>

- **Glide**
  - URL을 통해 ImageView에 이미지 로드를 간편하고 쉽게 해주는 라이브러리
  - 만약 이미지 로드 라이브러리가 없다면, 우리는 외부 URL을 통해 사진을 ImageView에 보여주기 위해서 OOM, 캐시, 병렬처리, 디코딩, 로드 실패 시 처리 로직, 재시도 처리 등.. 고려할 점이 넘쳐납니다!
  - 비슷한 이미지 로더 라이브러리로 “Picasso”가 있는데 서로 장단점이 있으니 검색을 통해 찾아보세요!!

통신 세미나 시작 전 할말! 까먹을까 봐 적어봤어요!

- 오늘 세미나 때 하지 않은 추가적인 통신(좋아요 기능, 수정하기, 삭제하기 등)은 몇 개 뽑아서 다음 세미나 때 잠깐 짚고 넘어갈 예정. or 과제로 낼게요! 다음주는 SOPT 10주년 행사라 2주 뒤 다음 세미나니까 오랜만에 과제!!!
  ➔ 오늘 한 것으로도 충분히 다른 통신을 다~ 할 수 있을 거라고 생각하므로!(지극히 개인적인 생각ㅎㅎ)

- 오늘 배운 내용을 한번에 이해하고 능숙하게 사용하면 좋겠지만, 그러지 않아도 됩니다. 왜냐하면!!

- 통신은 반복적인 코딩이 많습니다!!! 비슷한 패턴 반복!!! 이해 못해도 돼요!!
  그러니까 세미나 자료를 토대로 코드를 정리해서 앱잼 때 재사용하세요!!
  앱잼 때 참조할 코드를 원하면 모두 드리겠습니다!! 걱정마세요~!

- 중요한 것은 통신이 아니라 통신을 통해 전달 받은 데이터를 우리 생각대로 View에 뿌려주는 로직이 중요!!!
  그러니까 오늘 이해 못했다고 해서 낙심하지 마세요~!

02

Retrofit을 이용한 통신 실습 1

실습 1) 지금부터 톡방에 올리는 각 Activity의 view를 복붙해 주세요!

- MainActivity - 로그인 화면
- SignUpActivity - 회원가입 화면
- BoardActivity – 모든 게시물 보기 화면
- WriteActivity – 게시물 쓰기 화면

모든 소스 github 참조!!

https://github.com/Younanee/do_it_sopt_android_part/tree/dev/retrofit2_example

SHOUT OUR PASSION TOGETHER SOPT

실습 1) Application Class 만들기 - 1

- Application Class란?
  ➔ 여러 Android Component 사이에서 공유 가능한 전역 클래스.
  ➔ 앱이 실행될 때 가장 먼저 실행.
  ➔ 어디서든 context를 통해 접근 가능.

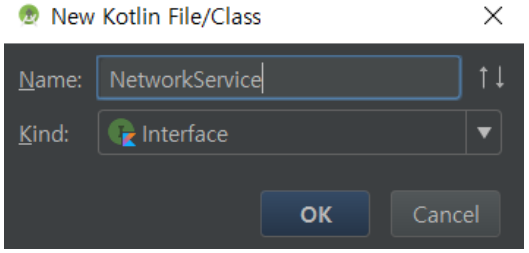
- 사용법
  ① Application()을 상속 받는 Class 만들기
  ② manifest에 등록하기

- 실습은 다음 장으로!

실습 1) Application Class 만들기 - 2

① 일단 Application Class를 만들기 전, 통신에 쓰일 추상 메소드를 모아둔 interface하나를 만들어 줍니다.

- network 패키지를 하나 만들고 그 안에 NetworkService라는 interface를 만듭니다.



- 내용물은 차근 차근 채워 넣어 볼게요!

② 다음은 network 패키지 안에 AppicationController라는 Class를 만듭니다. 그리고 Appication()을 상속 받습니다. 자동완성으로 상속!

```
import android.app.Application
class ApplicationController : Application() {
}
```

③ 그리고 마지막으로 manifest파일에 Application을 등록해 봅시다!

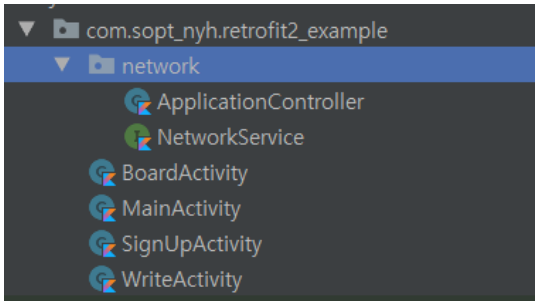
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.sopt_nyh.retrofit2_example">

    <application
        android:name=".network.ApplicationController"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
```

name 속성으로 등록

실습 1) Application Class 만들기 - 2

- 현재까지 모든 과정을 잘 따라왔다면, 사전 세미나 준비를 포함해 다음과 같이 프로젝트가 구성되어야 합니다.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android=http://schemas.android.com/apk/res/android
    package="[redacted]">

    <application
        android:name=".network.ApplicationController"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".SignUpActivity" />
        <activity android:name=".BoardActivity" />
        <activity android:name=".WriteActivity"></activity>
    </application>

</manifest>
```



SHOUT OUR PASSION TOGETHER SOPT

실습 1) Application Class 만들기 – 3

- ApplicationController에 Retrofit을 빌드하는 로직을 넣습니다.

```
import android.app.Application
import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

class ApplicationController : Application() {
    private val baseURL = "http://bghgu.tk:8080/"
    lateinit var networkService: NetworkService

    companion object {
        lateinit var instance: ApplicationController
    }

    override fun onCreate() {
        super.onCreate()
        instance = this
        buildNetWork()
    }

    fun buildNetWork() {
        val retrofit: Retrofit = Retrofit.Builder()
                .baseUrl(baseURL)
                .addConverterFactory(GsonConverterFactory.create())
                .build()
        networkService = retrofit.create(NetworkService::class.java)
    }
}
```

03

Retrofit을 이용한 통신 실습 2

실습2) 회원가입 통신을 해봅시다! - 0

- 빠른 실습 진행을 위해서 통신 외 로직은 모두 복붙을 통해 진행할게요!!
- 일단, 우리는 통신을 하므로 인터넷에 대한 권한을 명시 해야합니다!! manifest에 아래 권한을 넣어주세요!

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

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.sopt_nyh.retrofit2_example">
    <uses-permission android:name="android.permission.INTERNET" />
```

- MainActivity.kt 내용물을 아래와 같이 구성해주세요

앱잼 말고 직접 apk 로 내보낼때 매니페스트에 명시한 것 이외에도 권한에 대한 로직을 추가해줘야함
취소했을때 등등

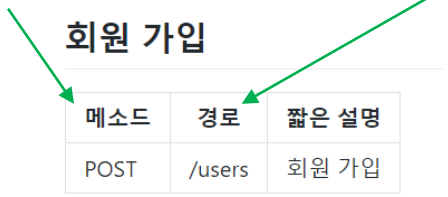
```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        setOnBtnClickListener()
    }

    private fun setOnBtnClickListener(){
        btn_main_act_log_in.setOnClickListener {

        }

        btn_main_act_sign_up.setOnClickListener {
            startActivity<SignUpActivity>()
        }
    }
}
```

실습2) 회원가입 통신을 해봅시다! – 1 본격적으로 회원가입 통신을 시작합니다!

- 우리는 서버 파트장이 제공해준 API 명세서를 통해 실습을 진행합니다.
- https://github.com/bghgu/SOPT-23-API-SERVER/wiki

① 첫번째, 회원가입에 대한 API를 확인합니다.
https://github.com/bghgu/SOPT-23-API-SERVER/wiki/%ED%9A%8C%EC%9B%90%EA%B0%80%EC%9E%85

② HTTP 메소드가 POST임을 확인하고 URL이 /users 인 것을 확인하세요!



회원 가입

메소드	경로	짧은 설명
POST	/users	회원 가입

③ Header와 Body에 들어갈 데이터를 확인하세요! 그리고 응답에 대한 데이터를 확인하세요!!

요청 헤더

```
Content-Type: application/json
```

요청 바디

```
{
    "name" : "테스트",
    "email" : "2",
    "password" : "1234",
    "part" : "서버"
}
```

회원 가입 성공

```
{
    "status": 201,
    "message": "회원 가입 성공",
    "data": null
}
```

실습2) 회원가입 통신을 해봅시다! – 2

- 우리는 회원가입 API를 확인하여 아래와 같은 통신에 쓰일 정보를 알 수 있었습니다.
  - ① HTTP 메소드 : POST
  - ② URL : /users
  - ③ 요청 헤더 데이터
  - ④ 요청 바디 데이터
  - ⑤ 응답 바디 데이터

1. 일단 응답에 대한 바디 데이터를 담을 data class를 만들어 주세요.
   post 디렉토리를 만든 뒤, post 디렉토리 안에 PostSignUpResponse.kt 파일을 만들고 응답 바디 데이터에 대응하여 변수/상수를 정의해줍니다. **꼭!!! 응답 데이터와 같은 데이터 명을 써줘야 합니다!! ← 매우 중요!!**

응답 바디

회원 가입 성공

```
{
    "status": 201,
    "message": "회원 가입 성공",
    "data": null
}
```

=

```
data class PostSignUpResponse(
        val status : String,
        val message : String
)
```

참고로, 응답 데이터를 null으로 주는 데이터는 안 받아도 됩니다! data라는 받는 그릇을 안만들어도 되요!!

실습2) 회원가입 통신을 해봅시다! – 2 이어서…

- 다음은 NetworkService 인터페이스에 통신에 쓰일 추상메소드를 추가해줍니다!
  여기서 HTTP 메소드, URL, 요청 헤더 데이터, 요청 바디 데이터 형식을 명시해줍니다!!

회원 가입

메소드	경로	짧은 설명
POST	/users	회원 가입

요청 헤더

```
Content-Type: application/json
```

요청 바디

```
{
    "name" : "테스트",
    "email" : "2",
    "password" : "1234",
    "part" : "서버"
}
```



```kotlin
import com.google.gson.JsonObject
import retrofit2.Call
import retrofit2.http.Body
import retrofit2.http.Header
import retrofit2.http.POST

interface NetworkService {
    //회원가입
    @POST("/users")
    fun postSignUpResponse(
            @Header("Content-Type") content_type : String,
            @Body() body : JsonObject
    ) : Call<PostSignUpResponse>

}
```

- 서버 요청 바디가 파라메터인지, 폼 인지 꼭!! 확인하세요. 지금은 폼 형식이므로 Json형식으로 보냅니다.

SHOUT OUR PASSION TOGETHER SOPT

실습2) 회원가입 통신을 해봅시다! – 3 이어서…

- 회원가입 버튼을 누르면 서버 통신 후 로그인을 할 MainActivity로 넘어가는 로직입니다.

```
class SignUpActivity : AppCompatActivity() {

    val networkService: NetworkService by lazy {
        ApplicationController.instance.networkService
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_sign_up)

        setOnBtnClickListener()
    }

    private fun setOnBtnClickListener() {
        btn_sign_up_act_complete.setOnClickListener {
            getSignUpResponseData()
        }
        btn_sign_up_act_close.setOnClickListener {
            finish()
        }
    }
```

```
    private fun getSignUpResponse() {
        //EditText에 있는 값 받기
        val input_name: String = et_sign_up_act_name.text.toString()
        val input_pw: String = et_sign_up_act_pw.text.toString()
        val input_email: String = et_sign_up_act_email.text.toString()
        val input_part: String = et_sign_up_act_part.text.toString()
        //Json 형식의 객체 만들기
        var jsonObject = JSONObject()
        jsonObject.put("name", input_name)
        jsonObject.put("email", input_email)
        jsonObject.put("password", input_pw)
        jsonObject.put("part", input_part)
        //Gson 라이브러리의 Json Parser을 통해 객체를 Json으로!
        val gsonObject = JsonParser().parse(jsonObject.toString()) as JsonObject

        val postSignUpResponse: Call<PostSignUpResponse> =
                networkService.postSignUpResponse("application/json", gsonObject)
        postSignUpResponse.enqueue(object : Callback<PostSignUpResponse> {
            override fun onFailure(call: Call<PostSignUpResponse>, t: Throwable) {
                Log.e("sign up fail", t.toString())
            }
            //통신 성공 시 수행되는 메소드
            override fun onResponse(call: Call<PostSignUpResponse>, response: Response<PostSignUpResponse>) {
                if (response.isSuccessful) {
                    toast(response.body()!!.message)
                    finish()
                }
            }
        })
    }
}
```

서버 통신 후 성공했다면,
이 onResponse 메소드 블락 안에서
데이터를 꺼내오고,
View에 뿌려지는 작업을 넣습니다.

실습3) 로그인 통신을 해봅시다! – 1

- 마찬가지로 로그인 API를 확인하여 아래와 같은 통신에 쓰일 정보를 확인합니다.
  - ① HTTP 메소드 : POST
  - ② URL : /login
  - ③ 요청 헤더 데이터
  - ④ 요청 바디 데이터
  - ⑤ 응답 바디 데이터

로그인

메소드	경로	짧은 설명
POST	/login	로그인

요청 헤더

```
Content-Type: application/json
```

요청 바디

```
{
    "email": "bghgu@naver.com",
    "password": "1234"
}
```

응답 바디

로그인 성공

```
{
    "status": 200,
    "message": "로그인 성공",
    "data": {
        "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1
    }
}
```

실습3) 로그인 통신을 해봅시다! – 2

① 응답에 대한 data class를 만듭니다. post 패키지 안에 PostLogInResponse.kr 생성

(주의! data class 명은 상관없지만 데이터 이름은 꼭!! 맞춰주세요!!)

```
data class PostLogInResponse(
        val status : Int,
        val message : String,
        val data : LoginData
)

data class LoginData(
        val token : String
)
```

응답 바디

로그인 성공


```
{
    "status": 200,
    "message": "로그인 성공",
    "data": {
        "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1
    }
}
```


② 다음 NetworkService에 로그인 통신에 이용할 추상 메소드를 추가합니다!

```
//로그인
@POST("/login")
fun postLoginResponse(
        @Header("Content-Type") content_type : String,
        @Body() body : JsonObject
) : Call<PostLogInResponse>
```

실습3) 로그인 통신을 해봅시다! – 3

- 서버 응답 data에 들어있는 token을 저장시키기 위해 지난 시간에 배웠던 SharedPreference를 만들어 봅시다!
  (db 패키지에 SharedPreferenceController.kr을 만들고 아래와 같은 코드를 넣습니다.)

  왜 token을 저장하는가?

  통신 API를 보면 헤더에 token이 필요한 통신이 있으므로! 저장 시킨 뒤 두고 두고 꺼내 쓰기 위해서!

  + 자동 로그인을 구현하기 위해서 ㅎㅎ!!

```
object SharedPreferenceController{
    private val USER_NAME = "MYKEY"
    private val myAuth = "myAuth"

    fun setAuthorization(context: Context, authorization : String){
        val pref = context.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE) //현재 내 기기에서만 볼수 있는 데이터
        val editor = pref.edit()
        editor.putString(myAuth, authorization)
        editor.commit()
    }

    fun getAuthorization(context: Context) : String {
        val pref = context.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE) //현재 내 기기에서만 볼수 있는 데이터
        return pref.getString(myAuth, "")
    }

    fun clearSPC(context: Context){
        val pref = context.getSharedPreferences(USER_NAME, Context.MODE_PRIVATE) //현재 내 기기에서만 볼수 있는 데이터
        val editor = pref.edit()
        editor.clear()
        editor.commit()
    }
}
```

SHOUT OUR PASSION TOGETHER SOPT

실습3) 로그인 통신을 해봅시다! – 4

- MainActivity.kr에 아래 코드를 구현합니다.

```
class MainActivity : AppCompatActivity() {
    val networkService: NetworkService by lazy {
        ApplicationController.instance.networkService
    }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        setOnBtnClickListener()
        //자동 로그인
        if (SharedPreferenceController.getAuthorization(this).isNotEmpty()){
            startActivity<BoardActivity>()
        }
    }

    private fun setOnBtnClickListener(){
        btn_main_act_log_in.setOnClickListener {
            getLoginResponse()
        }

        btn_main_act_sign_up.setOnClickListener {
            startActivity<SignUpActivity>()
        }
    }
```

```
    private fun getLoginResponse(){
        if (et_main_act_email.text.toString().isNotEmpty() && et_main_act_pw.text.toString().isNotEmpty()){
            val input_email = et_main_act_email.text.toString()
            val input_pw = et_main_act_pw.text.toString()
            val jsonObject : JSONObject = JSONObject()
            jsonObject.put("email", input_email)
            jsonObject.put("password", input_pw)
            val gsonObject : JsonObject = JsonParser().parse(jsonObject.toString()) as JsonObject

            val postLogInResponse = networkService.postLoginResponse("application/json", gsonObject)
            postLogInResponse.enqueue(object : Callback<PostLogInResponse>{
                override fun onFailure(call: Call<PostLogInResponse>, t: Throwable) {
                    Log.e("Login fail", t.toString())
                }

                override fun onResponse(call: Call<PostLogInResponse>, response: Response<PostLogInResponse>) {
                    if (response.isSuccessful){
                        val token = response.body()!!.data.token
                        //저번 시간에 배웠던 SharedPreference에 토큰을 저장!
                        SharedPreferenceController.setAuthorization(this@MainActivity, token)
                        toast(SharedPreferenceController.getAuthorization(this@MainActivity))
                        startActivity<BoardActivity>()
                    }
                }
            })
        }
    }
}
```

04

Retrofit을 이용한 통신 실습 3

실습4) 게시물 쓰기 통신을 해봅시다!! -1

- 일단, 이미지를 보내는 통신이 들어가기 때문에, 앨범을 통해 서버로 보낼 사진을 선택할 것 입니다!
  그러므로 앨범에 접근하는 권한이 있어야 합니다! 앞선 인터넷 권한과 마찬가지로 추가해주세요! manifest에!

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

- 다음은 앞선 통신과 마찬가지로 게시물 쓰기 API를 확인합니다.

글 작성

메소드	경로	짧은 설명
POST	/contents	글 작성

요청 헤더

```
Authorization: token
```

요청 바디

```
{
    "title" : "제목",
    "contents" : "내용",
    "photo" : 파일
}
```

응답 바디

글 작성 성공

```
{
    "status": 201,
    "message": "글 작성 성공",
    "data": null
}
```

실습4) 게시물 쓰기 통신을 해봅시다!! -2

- 앞선 작업과 마찬가지로 응답 바디 형식에 맞춰 data class를 만들어주세요!(post/PostWriteBoardResponse.kt)

응답 바디

글 작성 성공

```
{
    "status": 201,
    "message": "글 작성 성공",
    "data": null
}
```

```
data class PostWriteBoardResponse(
        val status : String,
        val message : String
)
```

- 마찬가지로 null을 보내는 data는 아예 명시하지 않아도 됩니다!

실습4) 게시물 쓰기 통신을 해봅시다!! -3

- 사진과 같은 File을 전송하는 통신은 @Multipart라는 조금 다른 방식으로 통신합니다.
  다음과 같은 코드를 NetworkService에 추상 메소드로 추가해주세요!

요청 헤더

```
Authorization: token
```

요청 바디

```
{
    "title" : "제목",
    "contents" : "내용",
    "photo" : 파일
}
```

```kotlin
//게시판 글쓰기
@Multipart
@POST("/contents")
fun postWriteBoardResponse(
        @Header("Authorization") token : String,
        @Part("title") title : RequestBody,
        @Part("contents") contents : RequestBody,
        @Part photo: MultipartBody.Part?
) : Call<PostWriteBoardResponse>
```

<NetworkService.kt>

- 이전처럼 헤더에 Content-Type를 붙이는 것이 아니라, @Multipart라는 어노테이션을 통해 타입을 명시합니다.
- 그리고 이전 요청 바디와 형식이 같지만, body에 json 형식의 데이터를 보내는 것이 아니라,
  @Part를 통해 데이터를 담습니다. @Part에 String 타입을 넣을 땐 String이 아닌 RequestBody에 담아줘야 합니다.
  (Int나 Double 등 다른 타입의 경우는 그대로 써줘도 됩니다!) 예시 → `@Part("age") age : Int,`

실습4) 게시물 쓰기 통신을 해봅시다!! – 4 : WriteActivity 코드 첫번째

- WriteActivity.kt의 코드는 다음과 같습니다. 일단 인스턴스 변수/상수와 앨범을 여는 로직을 확인해주세요.

```
class WriteActivity : AppCompatActivity() {
    val REQUEST_CODE_SELECT_IMAGE: Int = 1004
    private var mImage: MultipartBody.Part? = null

    val networkService: NetworkService by lazy {
        ApplicationController.instance.networkService
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_write)
        setOnBtnClickListener()
    }

    private fun setOnBtnClickListener() {
        btn_write_act_show_album.setOnClickListener {
            //앨범 여는 로직
            val intent = Intent(Intent.ACTION_PICK)
            intent.type = android.provider.MediaStore.Images.Media.CONTENT_TYPE
            intent.data = android.provider.MediaStore.Images.Media.EXTERNAL_CONTENT_URI
            startActivityForResult(intent, REQUEST_CODE_SELECT_IMAGE)
        }
        btn_write_act_complete.setOnClickListener {
            getWriteBoardResponse()
        }
    }
```

실습4) 게시물 쓰기 통신을 해봅시다!! – 5 : WriteActivity 코드 두번째

- 이어서, 아래 코드는 앨범에서 사진을 선택했을 때, onActivityResult로 사진을 받아오고,
  서버로 전송할 수 있는 image file로 변환 하는 과정입니다. + Glide를 통해 ImageView에 사진 띄우는 로직!
- mImage라는 인스턴스 변수에, 서버로 전송할 수 있도록 이미지 파일을 가공하여 넣습니다.

```
//앨범에서 사진을 선택했을때 실행되는 메소드
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == REQUEST_CODE_SELECT_IMAGE) {
        if (resultCode == Activity.RESULT_OK) {
            data?.let {
                var seletedPictureUri = it.data
                val options = BitmapFactory.Options()
                val inputStream: InputStream = contentResolver.openInputStream(seletedPictureUri)
                val bitmap = BitmapFactory.decodeStream(inputStream, null, options)
                val byteArrayOutputStream = ByteArrayOutputStream()
                bitmap.compress(Bitmap.CompressFormat.JPEG, 20, byteArrayOutputStream)
                val photoBody = RequestBody.create(MediaType.parse("image/jpg"), byteArrayOutputStream.toByteArray())
                //첫번째 매개변수 String을 꼭! 꼭! 서버 API에 명시된 이름으로 넣어주세요!!!
                mImage = MultipartBody.Part.createFormData("photo", File(seletedPictureUri.toString()).name, photoBody)
                //Glide을 사진 URI를 ImageView에 넣은 방식. 외부 URI가 아니라 굳이 Glide을 안써도 되지만 ㅎㅎ!
                Glide.with(this@WriteActivity).load(seletedPictureUri).thumbnail(0.1f).into(iv_write_act_choice_image)
            }
        }
    }
}
```

SHOUT OUR PASSION TOGETHER SOPT

실습4) 게시물 쓰기 통신을 해봅시다!! – 5 : WriteActivity 코드 세번째

- 이어서, 아래 코드는 데이터들을 RequestBody 객체로 만들어서 서버로 요청을 보내는 코드입니다.

```
private fun getWriteBoardResponse() {
    val input_title = et_write_act_title.text.toString()
    val input_contents = et_write_act_content.text.toString()
    if (input_title.isNotEmpty() && input_contents.isNotEmpty()) {
        //Multipart 형식은 String을 RequestBody 타입으로 바꿔줘야 합니다!
        val token = SharedPreferenceController.getAuthorization(this)
        var title = RequestBody.create(MediaType.parse("text/plain"), input_title)
        var contents = RequestBody.create(MediaType.parse("text/plain"), input_contents)

        val postWriteBoardResponse = networkService.postWriteBoardResponse(token, title, contents, mImage)

        postWriteBoardResponse.enqueue(object : Callback<PostWriteBoardResponse> {
            override fun onFailure(call: Call<PostWriteBoardResponse>, t: Throwable) {
                Log.e("write fail", t.toString())
            }

            override fun onResponse(call: Call<PostWriteBoardResponse>, response: Response<PostWriteBoardResponse>) {
                if (response.isSuccessful) {
                    toast(response.body()!!.message)
                    finish()
                }
            }
        })
    }
}
```

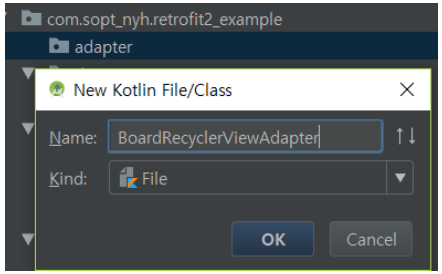
실습5) 이제 게시물 리스트를 구현해봅시다! 드디어 GET방식을 써볼 시간이에요! -1

- 일단 게시물들을 띄울 RecyclerView를 구현해 봅시다!!!
- 그 전에! 라이브러리 하나만 더 추가해 볼게요! 저번 시간에 알려드린 간편하게 둥근 ImageView 쓸 수 있는 것!

```
implementation 'de.hdodenhof:circleimageview:2.2.0'
```

- RecyclerView의 Item view는 카톡 단톡방으로 드리겠습니다! rv_item_board.xml을 만들어서 복붙!

- 오늘은 통신에 집중하기 위해서!! RecyclerViewAdapter 코드도 드리겠습니다!
  (일단 adapter 패키지를 만들고 그 안에 BoardRecyclerViewAdapter.kt 파일만 만들어주세요!)



실습5) 이제 게시물 리스트를 구현해봅시다! 드디어 GET방식을 써볼 시간이에요! -2

- 하나의 Board(게시물) item에 들어갈 데이터들이 무엇인지 확인하고 data class를 만들어 봅시다!

응답 바디

모든 글 조회 성공


```
{
    "status": 200,
    "message": "모든 글 조회 성공",
    "data": [
        {
            "b_id": 11,
            "b_title": "글 제목2",
            "b_contents": "내용내용내용",
            "b_date": "2018-11-03T13:47:35.000+0000",
            "u_id": 2,
            "b_like": 0,
            "b_photo": null,
            "auth": false,
            "like": false
        },
        {
            "b_id": 20,
            "b_title": "10",
            "b_contents": "",
            "b_date": "2018-11-03T13:47:35.000+0000",
            "u_id": 2,
            "b_like": 0,
            "b_photo": null,
            "auth": false,
            "like": false
        }
    ]
}
```


JSON에서 “[ ]”은 Array이라고 생각하시면 됩니다!
우리는 Array에 들어갈 value를 담을 data class를 만들 것 건데, 오른쪽 초록 박스가 하나의 value에요!
그러므로..

<data/BoardData.kt>

```
data class BoardData(
        val b_id : Int,
        val b_title : String,
        val b_contents : String,
        val b_date : String,
        val u_id : Int,
        val b_like : Int,
        val b_photo : String,
        val auth : Boolean,
        var like : Boolean
)
```

이렇게! json의 데이터 key 이름에 맞춰서 data class를 구성합니다! BoardData라는 객체 이름은 중요하지 않으니까 본인이 식별할 수 있는 이름으로!!!

SHOUT OUR PASSION TOGETHER SOPT

실습5) 이제 게시물 리스트를 구현해봅시다! 드디어 GET방식을 써볼 시간이에요! -3

- BoardRecyclerViewAdapter.kt – 앞서 만든 BoardData를 ArrayList로 받습니다!

```
class BoardRecyclerViewAdapter(val ctx: Context, val dataList: ArrayList<BoardData>) :
RecyclerView.Adapter<BoardRecyclerViewAdapter.Holder>() {
    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): Holder {
        val view = LayoutInflater.from(ctx).inflate(R.layout.rv_item_board, parent, false)
        return Holder(view)
    }

    override fun getItemCount(): Int = dataList.size

    override fun onBindViewHolder(holder: Holder, position: Int) {
        holder.title.text = dataList[position].b_title
        holder.like_쳇.text = dataList[position].b_like.toString()
        holder.date.text = dataList[position].b_date

        val requestOptions = RequestOptions()
//        requestOptions.placeholder(R.drawable.기본적으로 띄울 이미지)
//        requestOptions.error(R.drawable.에러시 띄울 이미지)
//        requestOptions.override(150)
        Glide.with(ctx)
                .setDefaultRequestOptions(requestOptions)
                .load(dataList[position].b_photo)
                .thumbnail(0.5f)
                .into(holder.image)

    }

    inner class Holder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val title: TextView = itemView.findViewById(R.id.tv_rv_item_board_title) as TextView
        val like_cnt: TextView = itemView.findViewById(R.id.tv_rv_item_board_like_cnt) as TextView
        val date: TextView = itemView.findViewById(R.id.tv_rv_item_board_date) as TextView
        val image: ImageView = itemView.findViewById(R.id.iv_rv_item_board_image) as ImageView
    }
}
```

Glide의 사용법을 집고 넘어 갈게요!

가장 기본적으로 Glide는

Glide.with(context or activity)

.load(이미지 URI)

.into(띄워질 ImageView Id)

with, load, into 메소드로 구현할 수 있어요!

나머지는 선택적으로 사용하면 됩니다!

실습5) 사전 작업이 끝났으니 본격적으로..! 모든 게시물보기 통신을 시작해 볼게요! - 4

- 일단 앞선 통신 작업처럼 모든 게시물 보기 API 명세서를 보고 통신 구현에 필요한 정보를 알아냅니다!

모든 글 조회

메소드	경로	짧은 설명
GET	/contents?offset={offset}&limit={limit}	모든 글 조회

QueryString 설명

Parameter	설명	예시	값 범위
offset	시작 번호(기본값 = 0)	offset=0	0 이상 정수
limit	가져올 데이터 갯수(기본값 = 10)	limit=10	1 이상 정수

요청 헤더

```
Content-Type: application/json
```

요청 헤더

```
Content-Type: application/json
```

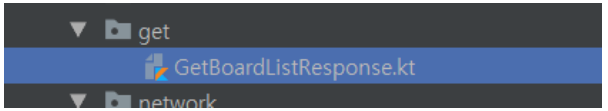
응답 바디

모든 글 조회 성공


```
{
    "status": 200,
    "message": "모든 글 조회 성공",
    "data": [
        {
            "b_id": 11,
            "b_title": "글 제목2",
            "b_contents": "내용내용내용",
            "b_date": "2018-11-03T13:47:35.000+0000",
            "u_id": 2,
            "b_like": 0,
            "b_photo": null,
            "auth": false,
            "like": false
        },
        {
            "b_id": 20,
            "b_title": "10",
            "b_contents": "",
            "b_date": "2018-11-03T13:47:35.000+0000",
            "u_id": 2,
            "b_like": 0,
            "b_photo": null,
            "auth": false,
            "like": false
        }
    ]
}
```

실습5) 사전 작업이 끝났으니 본격적으로..! 모든 게시물보기 통신을 시작해 볼게요! - 5

- Get 패키지 속 GetBoardListResponse.kt를 만듭니다!
  그리고 다음과 같이 응답 바디 데이터에 대한 틀을 만들어주세요!



응답 바디

모든 글 조회 성공


```
{
    "status": 200,
    "message": "모든 글 조회 성공",
    "data": [
        {
            "b_id": 11,
            "b_title": "글 제목2",
            "b_contents": "내용내용내용",
            "b_date": "2018-11-03T13:47:35.000+0000",
            "u_id": 2,
            "b_like": 0,
            "b_photo": null,
            "auth": false,
            "like": false
        },
        {
            "b_id": 20,
            "b_title": "10",
            "b_contents": "",
            "b_date": "2018-11-03T13:47:35.000+0000",
            "u_id": 2,
            "b_like": 0,
            "b_photo": null,
            "auth": false,
            "like": false
        }
    ]
}
```


```
data class GetBoardListResponse(
        val status : Int,
        val message : String,
        val data : ArrayList<BoardData>
)
```

앞서 말한 대로, JSON에서 “[ ]”는 Array를 의미하므로,
data라는 상수에 ArrayList<BoardData> 타입을 줍니다!
BoardData는 직전에 만든 그 data class에요!

```
data class BoardData(
        val b_id : Int,
        val b_title : String,
        val b_contents : String,
        val b_date : String,
        val u_id : Int,
        val b_like : Int,
        val b_photo : String,
        val auth : Boolean,
        var like : Boolean
)
```

실습5) 사전 작업이 끝났으니 본격적으로..! 모든 게시물보기 통신을 시작해 볼게요! - 6

- 다음은 NetworkService에 모든 게시물 보기 통신에 대한 추상 메소드를 만듭니다!

모든 글 조회

메소드	경로	짧은 설명
GET	/contents?offset={offset}&limit={limit}	모든 글 조회

QueryString 설명

Parameter	설명	예시	값 범위
offset	시작 번호(기본값 = 0)	offset=0	0 이상 정수
limit	가져올 데이터 갯수(기본값 = 10)	limit=10	1 이상 정수

요청 헤더

```
Content-Type: application/json
```

```
//모든 게시판 보기
@GET("/contents")
fun getBoardListResponse(
        @Header("Content-Type") content_type : String,
        @Query("offset") offset : Int,
        @Query("limit") limit : Int
) : Call<GetBoardListResponse>
```

- URI를 봤을 때,
  이렇게 ?가 있고 “=” 이꼬르~ 가 있으면 @Query를 통해 해당 파라메터를 전달합니다!
- 만약, URI가 @GET(“/contents/{offset}/{limit}”) 라면!!! @Path를 통해 전달!!!

```
@GET("/contents/{offset}/{limit}")
fun getBoardListResponse(
        @Header("Content-Type") content_type : String,
        @Path("offset") offset : Int,
        @Path("limit") limit : Int
) : Call<GetBoardListResponse>
```

실습5) 사전 작업이 끝났으니 본격적으로..! 모든 게시물보기 통신을 시작해 볼게요! - 7

- 이제 마지막으로!! BoardActivity.kt에 RecyclerView를 달고 통신을 해봅시다!!

<BoardActivity 첫번째 코드>

```
class BoardActivity : AppCompatActivity() {
    val WRITE_ACTIVITY_REQUEST_CODE = 1000
    lateinit var boardRecyclerViewAdapter: BoardRecyclerViewAdapter
    val dataList : ArrayList<BoardData> by lazy {
        ArrayList<BoardData>()
    }
    val networkService: NetworkService by lazy {
        ApplicationController.instance.networkService
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_board)

        setOnBtnClickListener()

        setRecyclerView()

        getBoardListResponse()

    }

    private fun setRecyclerView(){
        boardRecyclerViewAdapter = BoardRecyclerViewAdapter(this, dataList)
        rv_board_act_board_list.adapter = boardRecyclerViewAdapter
        rv_board_act_board_list.layoutManager = LinearLayoutManager(this)
    }
```

SHOUT OUR PASSION TOGETHER SOPT

실습5) 사전 작업이 끝났으니 본격적으로..! 모든 게시물보기 통신을 시작해 볼게요! - 8

- <BoardActivity 두번째 코드>, onActivityResult는 복습 겸해서 만들어봤어요! 왜 만들었는지 설명은 세미나 때!

```
private fun getBoardListResponse(){
    val getBoardListResponse = networkService.getBoardListResponse("application/json", 0, 30)
    getBoardListResponse.enqueue(object : Callback<GetBoardListResponse>{
        override fun onFailure(call: Call<GetBoardListResponse>, t: Throwable) {
            Log.e("board list fail", t.toString())
        }

        override fun onResponse(call: Call<GetBoardListResponse>, response: Response<GetBoardListResponse>) {
            if (response.isSuccessful){
                val temp : ArrayList<BoardData> = response.body()!!.data
                if (temp.size > 0){
                    val position = boardRecyclerViewAdapter.itemCount
                    boardRecyclerViewAdapter.dataList.addAll(temp)
                    boardRecyclerViewAdapter.notifyItemInserted(position)
                }
            }
        }
    })
}
private fun setOnBtnClickListener(){
    btn_board_act_write_board.setOnClickListener {
        startActivityForResult<WriteActivity>(WRITE_ACTIVITY_REQUEST_CODE)
    }
}
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == WRITE_ACTIVITY_REQUEST_CODE){
        if (resultCode == Activity.RESULT_OK){

        }
    }
}
```

마무리

정리 멘트 시간

실습 끝!!! 다시 한번 맨붕 왔을 파트원 분들께 할말!

- 오늘 세미나 때 하지 않은 추가적인 통신(좋아요 기능, 수정하기, 삭제하기 등)은 몇 개 뽑아서 다음 세미나 때 잠깐 짚고 넘어갈 예정. or 과제로 낼게요! 다음주는 SOPT 10주년 행사라 2주 뒤 다음 세미나거든요!
  ➔ 오늘 한 것으로도 충분히 다른 통신을 다~ 할 수 있을 거라고 생각하므로!(지극히 개인적인 생각)

- 오늘 배운 내용을 한번에 이해하고 능숙하게 사용하면 좋겠지만, 그러지 않아도 됩니다. 왜냐하면!!

- 통신은 반복적인 코딩이 많습니다!!! 비슷한 패턴 반복!!! 이해 못해도 돼요!!
  그러니까 세미나 자료를 토대로 코드를 정리해서 앱잼 때 재사용하세요!!
  앱잼 때 참조할 코드를 원하면 모두 드리겠습니다!! 걱정마세요~!

- 중요한 것은 통신이 아니라 통신을 통해 전달 받은 데이터를 우리 생각대로 View에 뿌려주는 로직이 중요!!!
  그러니까 오늘 이해 못했다고 해서 낙심하지 마세요~!

참조 사이트

① Retrofit에 대해 더 알아보기 - https://square.github.io/retrofit/

② Json에 대해 기본적인 정보 - https://zeddios.tistory.com/90/

③ Glide 최신 버전 훑어보고 싶다면 - https://github.com/bumptech/glide/

④ 포스트맨 (API 테스트) - https://www.getpostman.com/

SHOUT · OUR · PASSION · TOGETHER

ODo IT SOPT O

THANK U

세미나 자료 **남윤환**

SHOUT OUR PASSION TOGETHER
SOPT