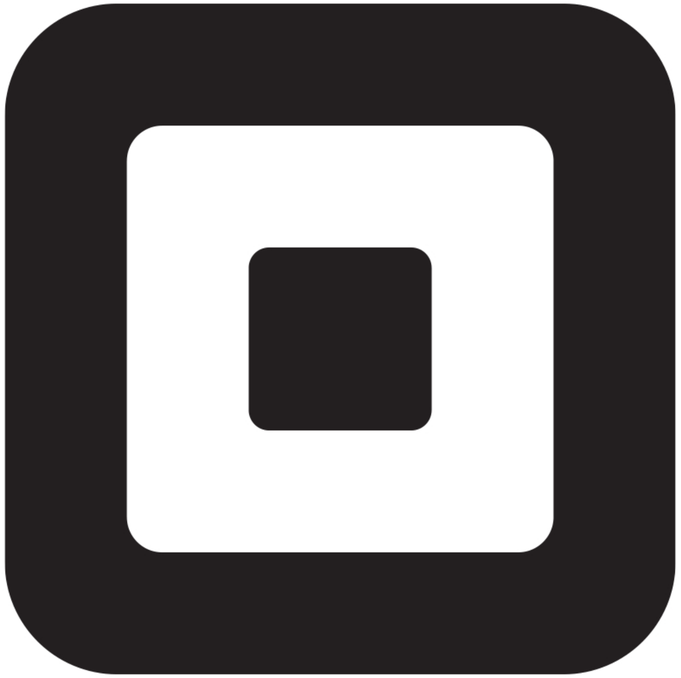
# MOBG56
# DEVELOPPEMENT D'APPLICATIONS MOBILES

# RETROFIT

- **Client HTTP pour Android**
- **Utilise les annotations pour décrire les requête HTTP**
- **Développé par Square (open-source)**
- **Très utilisé dans les développements d'apps connectées**
- **Avantage sur OkHTTP :**
  - Gère le threading
  - Converti automatiquement JSON -> Kotlin
- **Désavantage sur OkHTTP :** plus compliqué à debugger



# INSTALLATION

## Via Gradle :

```
dependencies {

    implementation 'com.squareup.retrofit2:retrofit:2.9.0'

    implementation ‘com.squareup.retrofit2:converter-gson:2.9.0’

}
```

## Ne pas oublier l’authorisation dans le Manifest :

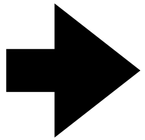
```
<uses-permission android:name="android.permission.INTERNET" />
```

# ÉTAPE 1 : DÉFINIR LES CLASSES RESSOURCE

- **Chaque JSON provenant du service web doit correspondre à une classe Kotlin**
- **Exemple :**

```json
{
  "id": 1,
  "name": "Rick Sanchez",
  "status": "Alive",
  "species": "Human",
  "type": "",
  "gender": "Male",
  "image": "https://rickandmortyapi.com/api/character/avatar/
1.jpeg",
  "episode": [
    "https://rickandmortyapi.com/api/episode/1",
    "https://rickandmortyapi.com/api/episode/2",
    // ...
  ],
}
```

JSON



```kotlin
class Character (
    val id: Long,
    @SerializedName("name")
    val characterName: String,
    val status: String,
    val species: String,
    val type: String,
    val gender: String,
    val image: String,
    val episode: List<String>,
)
```

Kotlin

# ÉTAPE 2 : DÉFINIR LE CLIENT

- **L'interface client décrit les requêtes HTTP qui vont retourner des ressources**

```
public interface MyRetrofitHttpClient {

    @GET ("character/{characterId}")
    fun  getCharacter(@Path("characterId") characterId : Int) : Call<Character>

    @GET ("character")
    fun getAllCharacters( @Query("page") page : Int) : Call<List<Character>>
}
```

→ ex : https://*base_url*/character/3

→ ex: https://*base_url*/character/?page=2

- **Chaque fonction est annotée (@*Get*) pour définir une URL correspondante**
- **Des paramètres peuvent être ajoutés à la requête**
  - ***Path parameter* avec l'annotation @*Path***
  - ***Query parameter* avec l'annotation @Query**

# ÉTAPE 3 : CONSTRUIRE UNE INSTANCE RETROFIT

- L'objet Retrofit va nous permettre de paramétrer et instancier notre client
- On définit un convertisseur JSON -> Kotlin (ici GSON)
- L'URL de base
- Le client est instancié une seule fois et sera réutilisé pour toutes les requêtes

```
val baseUrl = "https://rickandmortyapi.com/api/"
// create a converter JSON -> Kotlin
val jsonConverter : GsonConverterFactory= GsonConverterFactory.create()
// create a Retrofit builder
val retrofitBuiler : Retrofit.Builder = Retrofit.Builder().baseUrl(baseUrl).addConverterFactory(jsonConverter)
// create a Retrofit instance
val retrofit = retrofitBuiler.build()
// create our client
val myHttpClient = retrofit.create<MyRetrofitHttpClient>(MyRetrofitHttpClient::class.java)
```

# ÉTAPE 4 : ENVOYER UNE REQUÊTE

- **On définit un objet *Callback***
  - ***onResponse* :** la requête à réussi
  - ***onFailure* :** une erreur s'est produite
- **Méthode *enqueue* envoie la requête**
  - Méthode asynchrone (la requête est exécutée sur un thread parallèle)

```
val myHttpCall : Call<Character> = myHttpClient.getCharacter(3)

val callback : Callback<Character> = object : Callback<Character>{
    override fun onFailure(call: Call<Character>, t: Throwable) {
        // handle error here
    }

    override fun onResponse(call: Call<Character>, response: Response<Character>) {
        val character = response.body()
        // handle sucess response here
    }
}
myHttpCall.enqueue(callback)
```

# INFORMATIONS COMPLÉMENTAIRES

**Site officiel :** https://square.github.io/retrofit/

**Tutoriel :** https://futurestud.io/tutorials/retrofit-2-basics-of-api-description

**API Rick & Morty :** https://rickandmortyapi.com