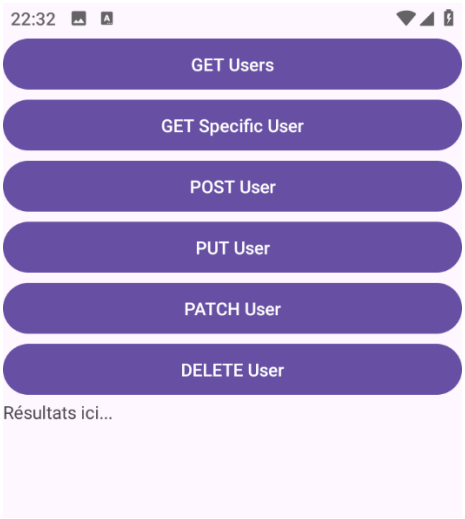
Consommation d'un web service en utilisant Retrofit

Travail à faire :

Créer une application une interface avec les éléments suivants :



1) Consommation du web service

On veut que cette application consomme les données générées par un Web service en utilisant la bibliothèque Retrofit.

a) Dépendances :

Ajoutez ces dépendances dans votre fichier build.gradle (Module: app) pour utiliser Retrofit et Gson pour la sérialisation JSON :

```
dependencies {
    implementation 'com.squareup.retrofit2:retrofit:2.9.0'
    implementation 'com.squareup.retrofit2:converter-gson:2.9.0'
    implementation 'com.squareup.okhttp3:logging-interceptor:4.9.0'  // Pour le
logging des requêtes
}
```

Remarque ! Dans les nouvelles versions, les dépendances sont organisés dans le fichier « libs.versions.toml » du gradle.

Dans ce cas, dans la section « versions » de ce fichier, ajouter :

```
retrofit = "2.9.0"
okhttp = "4.9.0"
```

dans la section « libraries », ajouter :

```
retrofit = { group = "com.squareup.retrofit2", name = "retrofit",
version.ref = "retrofit" }

      converter-gson = { group = "com.squareup.retrofit2", name =
      "converter-gson", version.ref = "retrofit" }

logging-interceptor = { group = "com.squareup.okhttp3", name = "logging-
interceptor", version.ref = "okhttp" }
```

b) Permissions :

Ajoutez dans AndroidManifest.xml :

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

c) Classes de Modèle

Créez une classe User.kt de données simple pour représenter une entité (par exemple, un utilisateur). Utilisez des annotations Gson pour la sérialisation.

```
data class User (@SerializedName("id") val id: Int? = null,
                 @SerializedName("name") val name: String?=null,
                 @SerializedName("email") val email: String?=null
)
```

d) Interface API avec Retrofit

Définissez une interface pour les appels CRUD. Utilisez l'url du fichier json JSONPlaceholder (https://jsonplaceholder.typicode.com) comme API de test.

```
interface ApiService {
    @GET("users")
    fun getUsers(): Call<List<User>>
    @POST("users")
    fun createUser(@Body user: User): Call<User>
    @PUT("users/{id}")
    fun updateUser(@Path("id") id: Int, @Body user: User): Call<User>
    @PATCH("users/{id}")
    fun patchUser(@Path("id") id: Int, @Body user: User): Call<User>
    @DELETE("users/{id}")
    fun deleteUser(@Path("id") id: Int): Call<Void>
    @GET("users/{id}")
    fun getUser(@Path("id") id: Int): Call<User>
}
```

e) Singleton pour Retrofit

Créez un objet singleton pour initialiser Retrofit.

```
object RetrofitClient {
    private const val BASE_URL = "https://jsonplaceholder.typicode.com/"
    val apiService: ApiService by lazy {
        val logging = HttpLoggingInterceptor().apply {
            level = HttpLoggingInterceptor.Level.BODY
        }
        val client = OkHttpClient.Builder()
            .addInterceptor(logging)
            .build()
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .client(client)
            .build()
            .create(ApiService::class.java)
    }
}
```

f) Activité Principale

Utiliser ces fonctions qui seront appelés lors des clicks sur le bouton pour réaliser sa requête et afficher son résultat dans le textView :

```
    ////////GET
    private fun getUsers() {
        RetrofitClient.apiService.getUsers().enqueue(object :
Callback<List<User>> {
            override fun onResponse(call: Call<List<User>>, response:
Response<List<User>>) {
                if (response.isSuccessful) {
                    val users = response.body()
                    tvResult.text = users?.joinToString("\n") { "ID:
${it.id}, Name: ${it.name}, Email: ${it.email}" } ?: "Aucun utilisateur"
                } else {
                    tvResult.text = "Erreur GET: ${response.code()}"
                }
            }
            override fun onFailure(call: Call<List<User>>, t: Throwable) {
                tvResult.text = "Échec GET: ${t.message}"
                Toast.makeText(this@MainActivity, "Erreur réseau",
Toast.LENGTH_SHORT).show()
            }
        })
    }

////////////POST
private fun createUser() {
    val newUser = User(name = "John Doe", email = "john@example.com")
    RetrofitClient.apiService.createUser(newUser).enqueue(object :
Callback<User> {
        override fun onResponse(call: Call<User>, response: Response<User>)
{
            if (response.isSuccessful) {
                val user = response.body()
                tvResult.text = "Utilisateur créé: ID ${user?.id}, Name:
```

```
${user?.name}"
                } else {
                    tvResult.text = "Erreur POST: ${response.code()}"
                }
            }
            override fun onFailure(call: Call<User>, t: Throwable) {
                tvResult.text = "Échec POST: ${t.message}"
            }
        })
    }

    /////////////// PUT
    private fun updateUser() {
        val updatedUser = User(id = 1, name = "Jane Doe", email =
"jane@example.com")
        RetrofitClient.apiService.updateUser(1, updatedUser).enqueue(object
: Callback<User> {
            override fun onResponse(call: Call<User>, response:
Response<User>) {
                if (response.isSuccessful) {
                    tvResult.text = "Utilisateur mis à jour:
${response.body()?.name}"
                } else {
                    tvResult.text = "Erreur PUT: ${response.code()}"
                }
            }
            override fun onFailure(call: Call<User>, t: Throwable) {
                tvResult.text = "Échec PUT: ${t.message}"
            }
        })
    }

    ///////////////PATCH
    private fun patchUser() {
        val partialUser = User(email = "newemail@example.com")  // PATCH
partiel
        RetrofitClient.apiService.patchUser(1, partialUser).enqueue(object
: Callback<User> {
            override fun onResponse(call: Call<User>, response:
Response<User>) {
                if (response.isSuccessful) {
                    tvResult.text = "Utilisateur patché:
${response.body()?.email}"
                } else {
                    tvResult.text = "Erreur PATCH: ${response.code()}"
                }
            }
            override fun onFailure(call: Call<User>, t: Throwable) {
                tvResult.text = "Échec PATCH: ${t.message}"
            }
        })
    }

    ////// DELETE

    private fun deleteUser() {
        RetrofitClient.apiService.deleteUser(1).enqueue(object :
Callback<Void> {
            override fun onResponse(call: Call<Void>, response:
Response<Void>) {
                if (response.isSuccessful) {
```

```
                    tvResult.text = "Utilisateur supprimé avec succès"
                } else {
                    tvResult.text = "Erreur DELETE: ${response.code()}"
                }
            }

            override fun onFailure(call: Call<Void>, t: Throwable) {
                tvResult.text = "Échec DELETE: ${t.message}"
            }
        })
    }

  //// GET One
  private fun getSpecificUser() {

      RetrofitClient.apiService.getUser(1).enqueue(object : Callback<User>
{
          override fun onResponse(call: Call<User>, response:
Response<User>) {
              if (response.isSuccessful) {
                  val user = response.body()
                  tvResult.text = "Utilisateur: ID ${user?.id}, Name:
${user?.name}, Email: ${user?.email}"
              } else {
                  tvResult.text = "Erreur GET Specific: ${response.code()}"
              }
          }

          override fun onFailure(call: Call<User>, t: Throwable) {
              tvResult.text = "Échec GET Specific: ${t.message}"
          }
      })
  }
```

2) Application des opérations de CRUD avec Retrofit sur un Web service que vous générer de votre projet d'intégration sur lequel vous travaillez ce semestre.