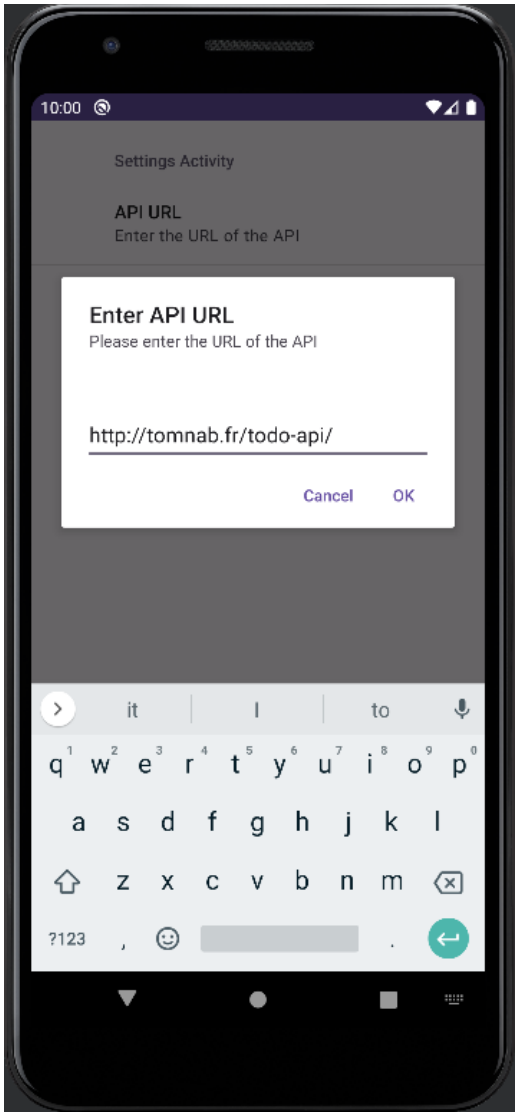
Bastien Poberay

Bilan du TEA2 de PMR

Introduction

Le but de ce second TEA était de créer une connexion entre notre application et une API en ligne afin d'accéder à un profil par le biais de MainActivity, à ses listes grâce à ChoixListActivity et aux items contenus dans ces listes par ShowListActivity.
La seule partie de l'interface qui change réellement est l'onglet settings, ou apparait un champ API URL qui nous permet de renseigner l'url de base de notre API, cette sera ensuite utilisée au travers de notre programme afin d'effectuer les requêtes.



Bastien Poberay

Présentation de quelques éléments du code

Le code est déjà commenté, mais je vais revenir sur quelques éléments clés de cette implémentation.
La création d'une requête requiert plusieurs phases :

Initialisation de retrofit :

```
private fun authenticateUser(pseudo: String, password: String) {
    //Lien de base pour l'api
    val baseUrl = sharedPreferences.getString("base_url", "")
    //Constructeur de requête
    val retrofit = Retrofit.Builder() Retrofit.Builder
        .baseUrl(baseUrl.toString()) Retrofit.Builder
        .addConverterFactory(GsonConverterFactory.create())
        .client(createOkHttpClient())
        .build()
```

Envoi de la requête :

```
val service = retrofit.create(AuthenticationService::class.java)
val call = service.authenticate(pseudo, password)
// Envoi de la requête
call.enqueue(object : Callback<AuthenticationResponse> {
```

Gestion de la réponse :

```
override fun onResponse(
    call: Call<AuthenticationResponse>,
    response: Response<AuthenticationResponse>
) {
    //Si la réponse est positive, on récupère le hash
    if (response.isSuccessful) {
        val authenticationResponse = response.body()
        val hash = authenticationResponse?.hash
        saveHash(hash.orEmpty())
        startChoixListActivity(pseudo)
        //Sinon message d'erreur
    } else {
        displayErrorDialog(
            title: "Erreur",
            message: "L'identification a échoué"
        )
    }
}
```

Bastien Poberay

Structure de la requête :

```
//modèle de la requête
interface AuthenticationService {
    @FormUrlEncoded
    @POST("authenticate")
    fun authenticate(
        @Field("user") user: String,
        @Field("password") password: String
    ): Call<AuthenticationResponse>
}

data class AuthenticationResponse(
    @SerializedName("hash") val hash: String?
)
```

Structure de la réponse :

```
//modèle de la réponse de l'api à la requête d'affichage des listes
data class ResponseModel(
    val version: Double,
    val success: Boolean,
    val status: Int,
    val apiname: String,
    val lists: List<ListItem>
)
```

Et un outil très utile pour comprendre les erreurs de dialogue entre notre application et l'API :

```
//Permet d'avoir un retour dans le logcat des messages d'erreurs liés aux échanges avec l'api
private fun createOkHttpClient(): OkHttpClient {
    val loggingInterceptor = HttpLoggingInterceptor(object : HttpLoggingInterceptor.Logger {
        override fun log(message: String) {
            Log.d( tag: "Retrofit", message)
        }
    })
    loggingInterceptor.level = HttpLoggingInterceptor.Level.BODY

    return OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor)
        .build()
}
```

On implémente ces éléments dans les trois activités MainActivity, ChoixListActivity et ShowListActivity, en modifiant la forme de la requête et la structure de la classe qui va recevoir la réponse, l'initialisation et la gestion de la réponse sont plus ou moins identiques.

Bastien Poberay

Difficultés

Je n'ai malheureusement pas pu implémenter la persistance des cases cochées, j'ai également quelques erreurs lors de l'ajout d'une liste ou d'un item, mais l'application fonctionne, elle ramène à l'activité main en cas de problème.
Le code n'est pas optimal puisqu'il repart de son état lors du dernier TEA

Perspectives

On pourrait imaginer implémenter les boutons de suppression des listes et des items, et pourquoi pas des éléments accessoires comme des outils de filtre par ordre alphabétique, nombre d'items, …
Optimiser la lisibilité et aussi l'ordre du code.
Le côté esthétique peut aussi être grandement amélioré, et de nouveaux messages d'erreurs afin de comprendre la source du problème pourraient être affichés à l'écran.
Pourquoi pas une liste déroulante qui retient les anciens url d'api afin de ne pas avoir à les rerenseigner.

Sources

Surtout des forums tels qu'openclassroom, stackoverflow, ainsi que le site https://developer.android.com/kotlin?hl=fr qui donne des indications sur chaque élément de kotlin.

ChatGPT pour tout ce qui est débogage, reformulation, et la partie XML avec laquelle je ne me suis pas encore totalement familiarisé.