

RÉSUMÉ PRATIQUE – FILIÈRE DÉVELOPPEMENT DIGITAL

Option – Application mobiles

M206 - Élaborer une application Android sécurisée

50 heures

SOMMAIRE



1. Manipuler les services Web sous Android

- Activité 1 : Consommation d'une API REST
- Activité 2 : Utilisation d'une API SOAP
- Activité 3 : Sécuriser une application qui communique avec un serveur

2. Gérer les différentes méthodes de stockage de données sous Android

- Activité 1 : Création d'une application qui consomme une API et qui sauvegarde des données sur le téléphone.
- Activité 2 : Sauvegarder des données dans des fichiers sur le téléphone
- Activité 3 : Partager des données entre deux applications

3. Publier une application sur le store

- Activité 1 : Gérer les signatures et versions
- Activité 2 : Utiliser des plateformes d'intégration et déploiements continus

MODALITÉS PÉDAGOGIQUES



1

LE GUIDE DE SOUTIEN

Il contient le résumé théorique et le manuel des travaux pratiques



2

LA VERSION PDF

Une version PDF est mise en ligne sur l'espace apprenant et formateur de la plateforme WebForce Life



3

DES CONTENUS TÉLÉCHARGEABLES

Les fiches de résumés ou des exercices sont téléchargeables sur WebForce Life



4

DU CONTENU INTERACTIF

Vous disposez de contenus interactifs sous forme d'exercices et de cours à utiliser sur WebForce Life

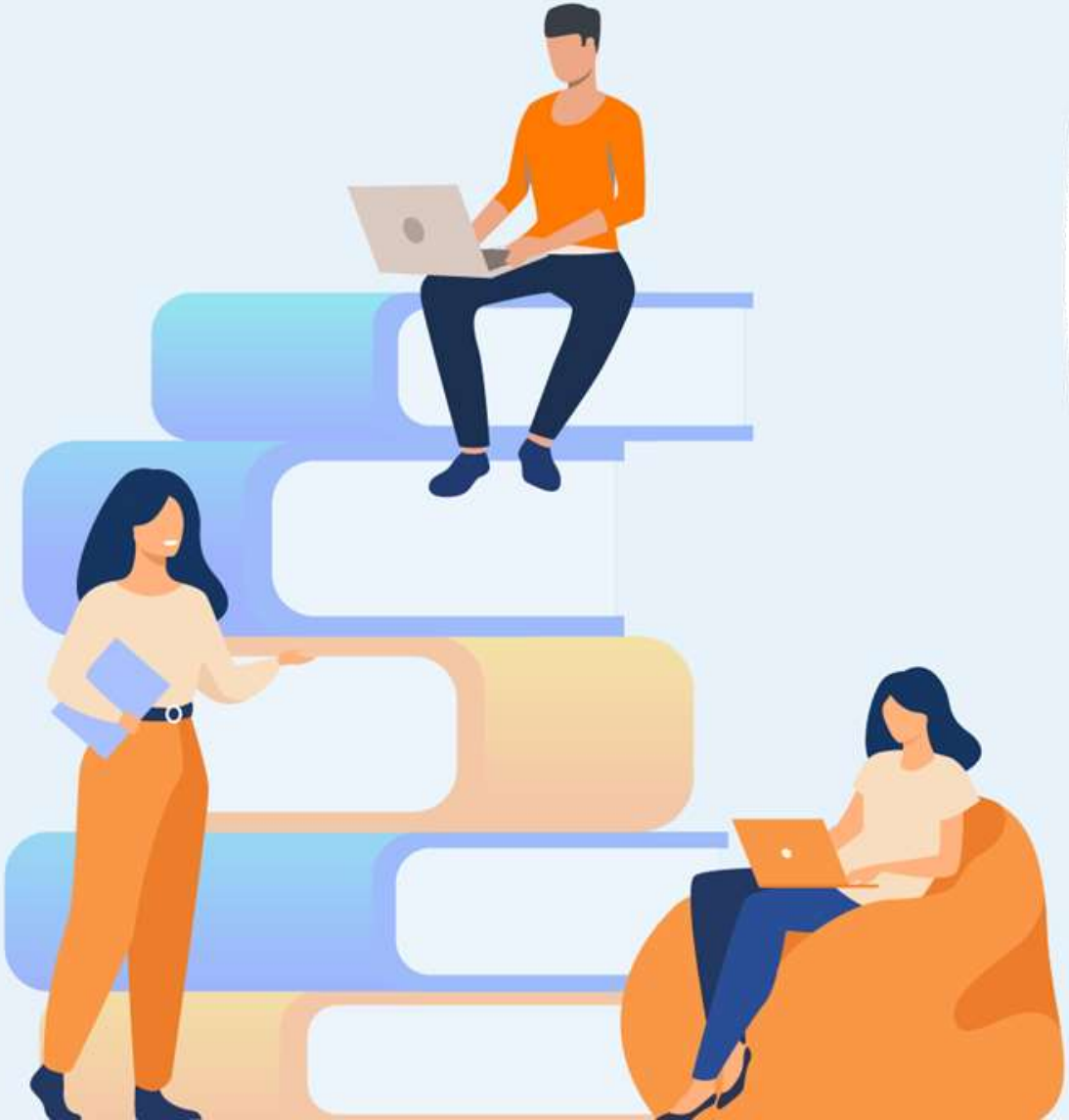


5

DES RESSOURCES EN LIGNES

Les ressources sont consultables en synchrone et en asynchrone pour s'adapter au rythme de l'apprentissage

Copyright - Tout droit réservé - OFPPT



PARTIE 1

Manipuler les services Web sous Android

Dans ce module, vous allez :

- Découvrir Protocole HTTP
- Consommer API sous Android
- Sécuriser les échanges des données

16 heures



Activité 1

Consommation d’une API REST

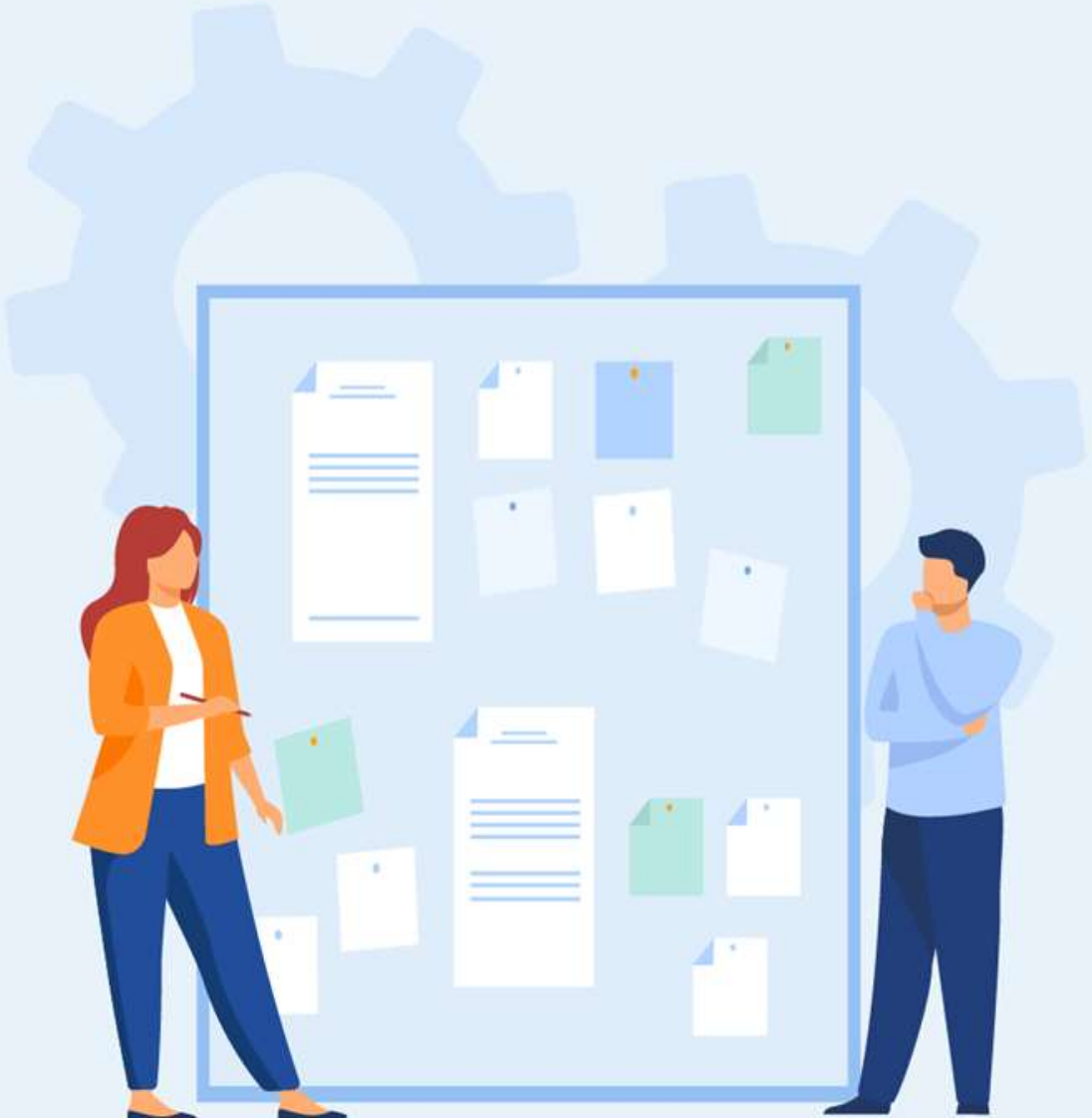
Compétences visées :

- Comprendre le protocole HTTP
- Consommer un API Rest

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l’IDE

06 heures



CONSIGNES

Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

Pour l'apprenant

- Créer un projet du travail
- Exécuter une requête HTTP
- Parser le flux JSON

Conditions de réalisation

- Support de résumé théorique
- Android Studio installé
- Retrofit ou Volley

Critère de réussite

- Le stagiaire est-il capable de :
  Manipuler un service Web API Rest

La librairie Volley

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

La permission à jouter dans le fichier **AndroidManifest**

```
implementation 'com.android.volley:volley:1.2.1'
```

La librairie **Volley** à ajouter dans le fichier Build.gradle

Pour mettre en place Volley, il faut :

- Ajouter la librairie dans le fichier Build.gradle
- Synchroniser le projet pour apporter les ressources distantes
- Ajouter la permissions INTERNET dans le fichier AndroidManifest.

Copyright - Tout droit réservé - OFPPT

Consommation d'une API REST

En utilisant la librairie **Volley** on peut exécuter une requête HTTP pour récupérer le contenu JSON d'un site d'e-commerce disponible sur l'endPoint (URL) de l'API Rest.

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState:
Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
           var url =
"https://dummyjson.com/products/3"
        var mRequestQueue =
Volley.newRequestQueue(getApplicationContext());

        var mStringRequest = StringRequest(
            Request.Method.GET, url,
            Response.Listener<String> { response
->

findViewById<TextView>(R.id.text).setText(respon
se)
            }, Response.ErrorListener { error ->
Log.i("TAG", "Error :$error") })
    mRequestQueue.add<String>(mStringRequest)
   }}
```



{"id":3,"title":"Samsung Universe 9","description":"Samsung's new variant which goes beyond Galaxy to the Universe","price":1249,"discountPercentage":15.46,"rating":4.09,"stock":36,"brand":"Samsung","category":"smartphones","thumbnail":"https://i.dummyjson.com/data/products/3/thumbnail.jpg","images":["https://i.dummyjson.com/data/products/3/1.jpg"]}

L'exécution

PARTIE 1

Copyright - Tout droit réservé - OFPPT

La librairie Retrofit

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

Les permissions à jouter dans le fichier **AndroidManifest**

```
implementation 'com.squareup.retrofit2:retrofit:2.9.0'
implementation 'com.squareup.retrofit2:converter-gson:2.3.0'
```

La librairie **Retrofit** à ajouter dans le fichier Buid.gradle

Pour mettre en place Retrofit, il faut :

- Ajouter la librairie dans le fichier Build.gradle
- Synchroniser le projet pour apporter les ressources distantes
- Ajouter la permissions INTERNET dans le fichier Android Manifest.

Copyright - Tout droit réservé - OFPPT

PARTIE 1

Consommation d'une API REST

En utilisant la librairie **Retrofit** on peut exécuter une requête HTTP pour récupérer le contenu JSON d'un site d'e-commerce disponible sur l'endPoint (URL) de l'API Rest.

```
interface GetDataService {
   @get:GET("3")
   val product:
Call<JsonObject?>?
}
```

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

        var url = "https://dummyjson.com/products/"
        var retrofit =  Retrofit.Builder()
            .baseUrl(url)

.addConverterFactory(GsonConverterFactory.create())
            .build()

        val service: GetDataService =
retrofit.create(
            GetDataService::class.java)
        val call: Call<JsonObject?>? =
service.product
        call?.enqueue(object : Callback<JsonObject?>
{
            override fun onResponse(call:
Call<JsonObject?>, response: Response<JsonObject?>)
{

findViewById<TextView>(R.id.text).setText(response.b
ody().toString())
            }
            override fun onFailure(call:
Call<JsonObject?>, t: Throwable) {
                Log.e("TAG",t.message.toString())
            }
```

My Application

{"id":3,"title":"Samsung Universe 9","description":"Samsung's new variant which goes beyond Galaxy to the Universe","price":1249,"discountPercentage":15.46,"rating":4.09,"stock":36,"brand":"Samsung","category":"smartphones","thumbnail":"https://i.dummyjson.com/data/products/3/thumbnail.jpg","images":["https://i.dummyjson.com/data/products/3/1.jpg"]}

L'exécution

Copyright - Tout droit réservé - OFPPT

Activité 2

Utilisation d’une API SOAP



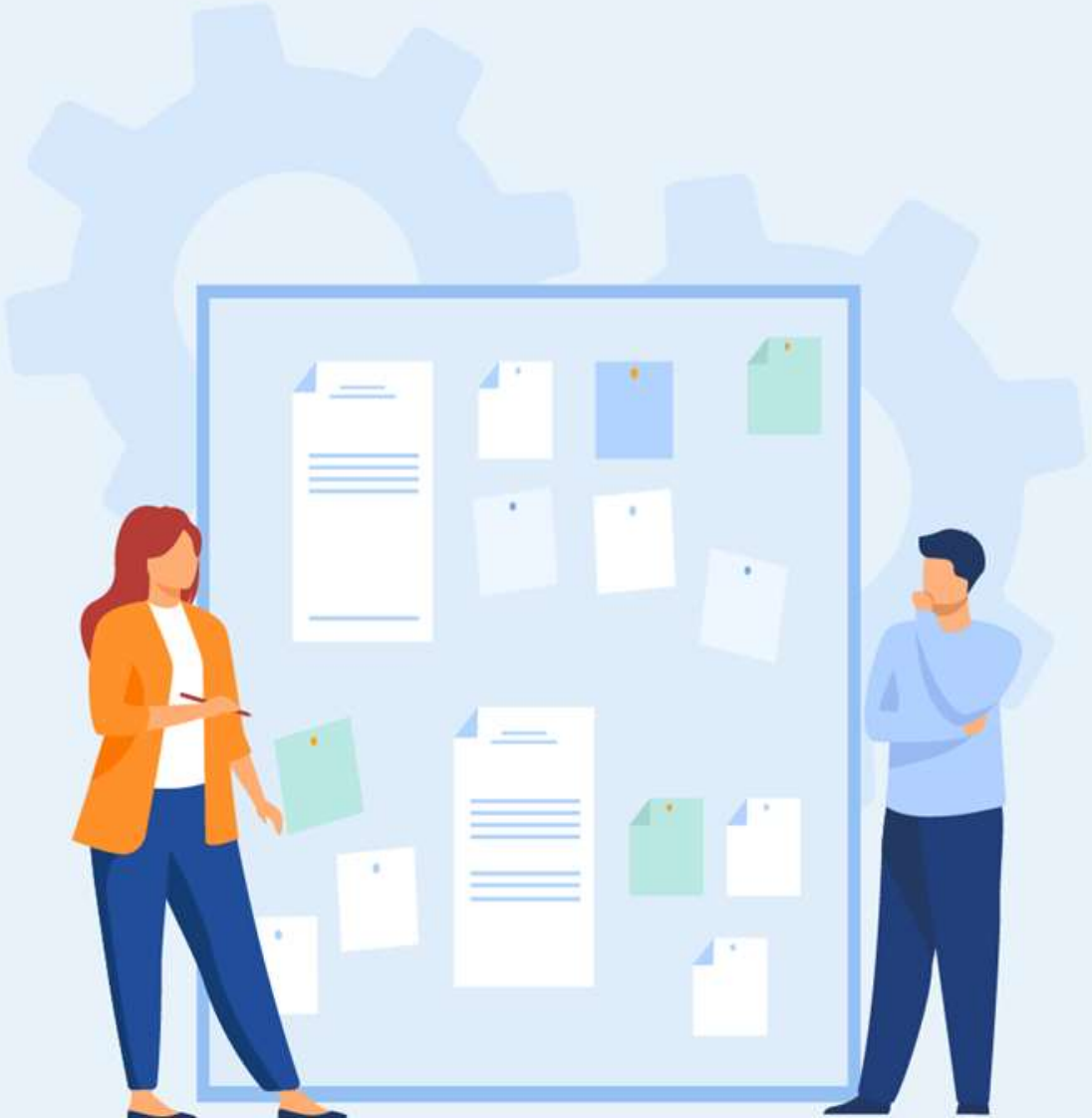
Compétences visées :

- Comprendre le protocole HTTP
- Consommer un API SOAP

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l’IDE

02 heures



CONSIGNES

Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

Pour l'apprenant

- Créer un projet du travail
- Exécuter une requête HTTP
- Parser le flux XML

Conditions de réalisation

- Support de résumé théorique
- Android Studio installé
- Retrofit ou Volley

Critère de réussite

- Le stagiaire est-il capable de :
  Manipuler un service Web API SOAP

Utilisation d'une API SOAP

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    var tv1 = findViewById<View>(R.id.text) as TextView
    try {
        val `is`: InputStream = assets.open("file.xml")
        val dbFactory: DocumentBuilderFactory =
DocumentBuilderFactory.newInstance()
        val dBuilder: DocumentBuilder =
dbFactory.newDocumentBuilder()
        val doc: Document = dBuilder.parse(`is`)
        val element: Element = doc.getDocumentElement()
        element.normalize()
        val nList: NodeList = doc.getElementsByTagName("product")
        for (i in 0 until nList.getLength()) {
            val node: Node = nList.item(i)
            if (node.getNodeType() === Node.ELEMENT_NODE) {
                val element2: Element = node as Element
                tv1.setText((tv1.getText() .toString()+ "\nNom : " +
getValue("name",element2)).toString() + "\n")
  tv1.setText( (tv1.getText().toString()+  "Prix : " + getValue(
"price",  element2)).toString() + "\n" )
  tv1.setText(tv1.getText() .toString()+"----------------------")
}}} catch (e: Exception) {
        e.printStackTrace()}}
```

```
<?xml version="1.0"?>
<records>
    <product>
        <name>Samsung 22</name>
        <price>10 000 DH</price>
    </product>
</records>
```

Flux XML interne

Pour manipuler un flux, on peut passer par Volley/Retrofit en cas d'un API (Rest ou SOAP) à travers une requête HTTP, ou bien on peut utiliser un fichier interne.

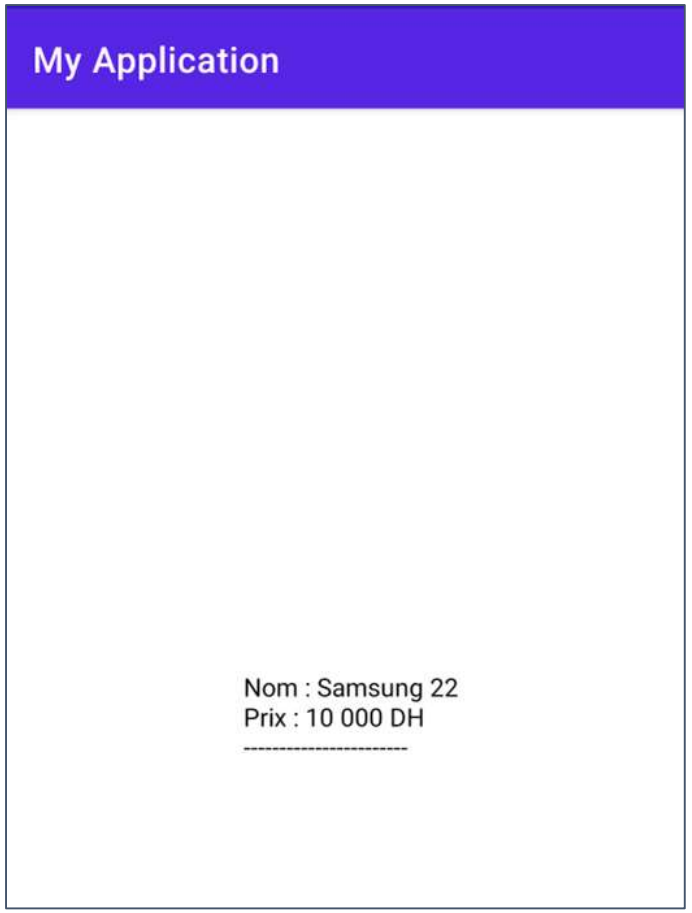
PARTIE 1

Copyright - Tout droit réservé - OFPPT

Utilisation d'une API SOAP

```
private fun getValue(tag: String, element:
Element): String? {
    val nodeList =
element.getElementsByTagName(tag).item(0).chil
dNodes
    val node = nodeList.item(0)
    return node.nodeValue
}
```

La méthode **getValue** qui permet de parser les valeurs de chaque membre de l'arbre XML.



L'exécution

PARTIE 1

Copyright - Tout droit réservé - OFPPT



Activité 3

Sécuriser une application qui communique avec un serveur

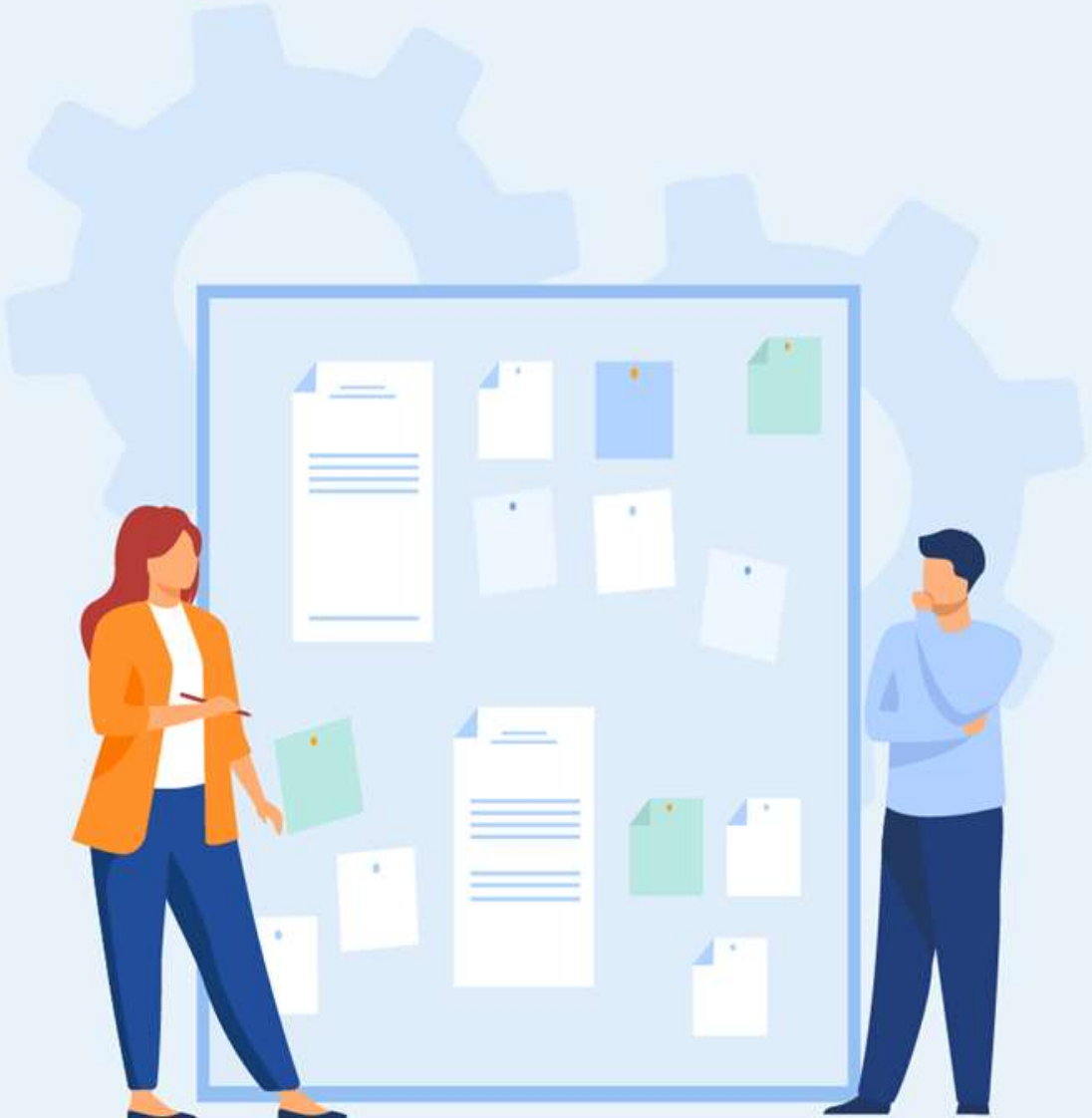
Compétences visées :

- Comprendre le OAuth
- Comprendre le SSL
- Manipuler Proguard

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l'IDE

08 heures



CONSIGNES

Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

Pour l'apprenant

- Créer un projet du travail
- Sécuriser l'échange de l'application
- Protéger le code de l'APK
- Optimiser la taille de l'APK

Conditions de réalisation

- Support de résumé théorique
- Android Studio installé
- Compte OAuth
- Certificat de SSL

Critère de réussite

- Le stagiaire est-il capable de :
  - Sécuriser l'échange entre l'application et le serveur
  - Obfusquer le code source de l'application

Utiliser OAUTH : Google Sign-In

```
apply plugin: 'com.android.application'

dependencies {

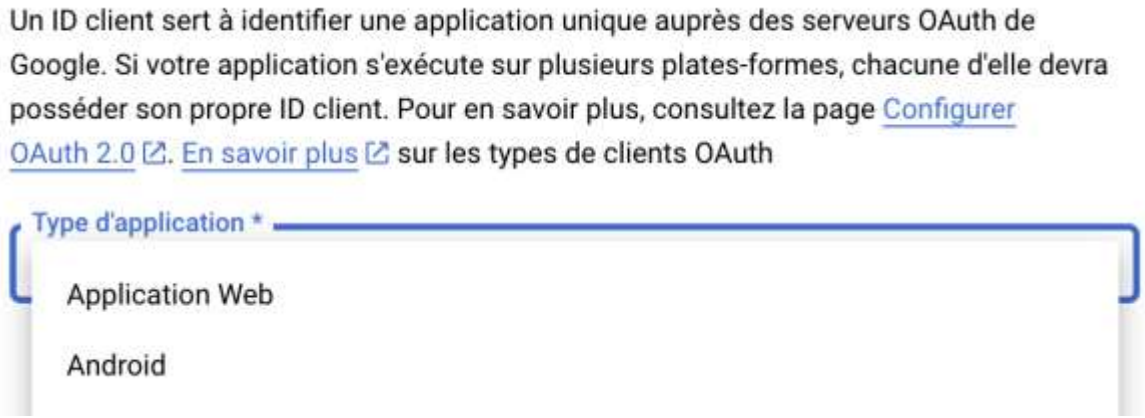
    implementation 'com.google.android.gms:play-services-auth:20.4.1'

}
```

Ajouter les dépendances de Google dans le fichier Build.gradle

Sur la page "**Identifiants**", créez un ID client de type **Android** pour votre application, si vous n'en avez pas déjà un. Vous devrez spécifier le nom du package de votre application et l'empreinte du certificat SHA-1.





PARTIE 1

Copyright - Tout droit réservé - OFPPT

Utiliser OAUTH : Google Sign-In

- Dans la méthode **onCreate** de votre activité de connexion, configurez Google Sign-In pour demander les données utilisateur requises par votre application

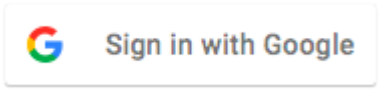
```
val gso: GoogleSignInOptions =
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN
)
    .requestEmail()
    .build()

mGoogleSignInClient = GoogleSignIn.getClient(this, gso)
```

- Dans la méthode **onStart** de votre activité, vérifiez si un utilisateur s'est déjà connecté à votre application avec Google.

```
val account = GoogleSignIn.getLastSignedInAccount(this)
updateUI(account)
```

- Ajoutez le **SignInButton** dans la mise en page de votre application:




```
<com.google.android.gms.common.SignInB
utton
    android:id="@+id/sign_in_button"
    android:layout_width="wrap_content"
   android:layout_height="wrap_content"
/>
```

PARTIE 1

Copyright - Tout droit réservé - OFPPT

Activité 3

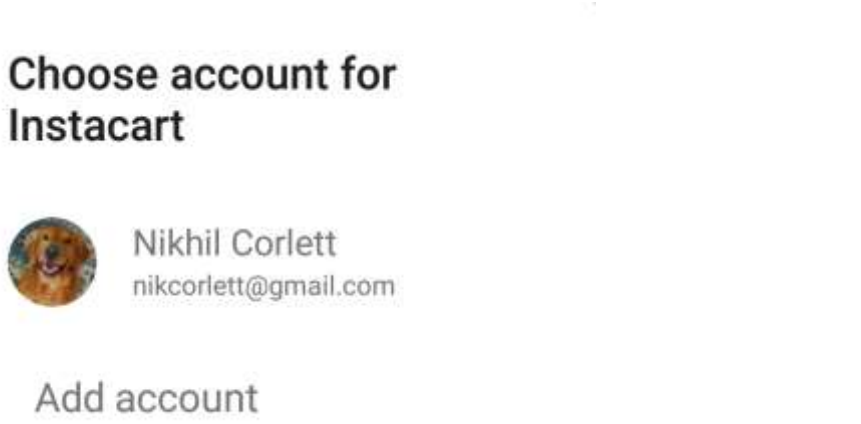
Sécuriser une application qui communique avec un serveur

Utiliser OAUTH : Google Sign-In

- Démarrer le parcours de connexion

```
private fun signIn() {
    val signInIntent: Intent =
mGoogleSignInClient.getSignInIntent()
    someActivityResultLauncher.launch(signInIntent)
}
```

```
var someActivityResultLauncher:
ActivityResultLauncher<Intent> =
registerForActivityResult(
    StartActivityForResult(),
    ActivityResultCallback<Any> { result ->
        if (result.getResultCode() === RESULT_OK) {
            // There are no request codes
            val data: Intent = result.getData()
val task: Task<GoogleSignInAccount> =
    GoogleSignIn.getSignedInAccountFromIntent(data)
         handleSignInResult(task)
        }
    })
```




```
private fun handleSignInResult(completedTask:
Task<GoogleSignInAccount>) {
    try {
        val account: GoogleSignInAccount =
completedTask.getResult(ApiException::class.java)

        updateUI(account)
    } catch (e: ApiException) {
        Log.w(TAG, "signInResult:failed code=" + e.statusCode)
    }
}
```

Vous pouvez obtenir l'adresse e-mail de l'utilisateur avec **getEmail**.

PARTIE 1

Copyright - Tout droit réservé - OFPPT

Proguard

- Afin de protéger l'APK du projet contre les attaques qui visent à compromettre le code source, il faut activer Proguard avant de générer la version release.
- Proguard offre aussi l'optimisation de la taille de l'APK à travers le compilateur **R8**.

```
buildTypes {
    debug {
        debuggable true
        minifyEnabled false
        shrinkResources false
    }
    release {
        minifyEnabled true
        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        shrinkResources true
    }
}
```

R8

- R8 est un outil de réduction d'application utilisé pour réduire la taille de votre application. Cet outil présent dans Android Studio fonctionne avec les règles de Proguard. R8 convertira le code de votre application en code Dalvik optimisé

Copyright - Tout droit réservé - OFPPT

Proguard

- Il y a des classes et des variables que le développeur doit conserver hors Proguard pour assurer le bon fonctionnement de l'application en utilisant -**keep :**

```
-keep public class * extends android.app.Activity
-keep public class * extends android.app.Application
-keep public class * extends android.app.Service
-keep public class * extends android.content.BroadcastReceiver
-keep public class * extends android.content.ContentProvider
-keep public class * extends android.app.backup.BackupAgentHelper
-keep public class * extends android.preference.Preference
```

- **R8** crée, lors de chaque exécution, un fichier mapping.txt qui contient les informations nécessaires pour remapper les traces de pile sur celles d'origine.

```
java.util.Iterator iterator -> a
java.lang.Object nextItem -> c
java.lang.String code -> b
int value -> a
```


Copyright - Tout droit réservé - OFPPT

Utiliser SSL

Si la configuration de sécurité du serveur oblige la communication à travers un certificat public généré spécifiquement pour votre application, on peut utiliser le client **Retrofit** :
La première chose dont vous avez besoin pour représenter un certificat public en tant que InputStream :

```
val pubCertInputStream = context.resources.openRawResource(R.raw.pub_cert)
```

Ensuite, vous devez représenter la clé privée elle-même (sans le truc "BEGIN / END RSA PRIVATE KEY") sous forme de chaîne, pour créer un objet PrivateKey:

```
private fun loadPrivateKey(): PrivateKey? {
    val encoded: ByteArray = Base64.decode(privateKeyString, Base64.DEFAULT)
    val keySpec = PKCS8EncodedKeySpec(encoded)
    val kf = KeyFactory.getInstance("RSA")
    return kf.generatePrivate(keySpec)
}
```

Remarques

- pub_cert : Le serveur vous fournit ce certificat à ajouter dans le projet Android

-----BEGIN RSA PRIVATE KEY-----
MIIBOgIBAAJBAKj34GkxFhD90vcNLYLInFE
X6Ppy1tPf9Cnzj4p4WGeKLs1Pt8Qu
KUpRKfFLfRYC9AIKjbJTWit+CqvjWYzvQw
ECAwEAAQJAIJLixBy2qpFoS4DSmoEm
o3qGy0t6z09AIJtH+5OeRV1be+N4cDYJKff
GzDa88vQENZiRm0GRq6a+HPGQMd2k
TQIhAKMSvzIBnni7ot/OSie2TmJLY4SwTQ
AevXysE2RbFDYdAiEBCUEaRQnMnbp7
9mxDXDf6AU0cN/RPBjb9qSHDcWZHGzUC
IG2Es59z8ugGrDY+pxLQnwfotadxd+Uy
v/Ow5T0q5gIJAiEAyS4RaI9YG8EWx/2w0T6
7ZUVAw8eOMB6BIUg0Xcu+3okCIBOs
/5OiPgoTdSy7bcF9IGpSE8ZgGKzgYQVZeN
97YE00
-----END RSA PRIVATE KEY-----

Exemple de clé privé

PARTIE 1

Copyright - Tout droit réservé - OFPPT

Activité 3

Sécuriser une application qui communique avec un serveur

Utiliser SSL

Vous devez maintenant construire l'objet **OkHttpClient** :

```
val cf: CertificateFactory =
CertificateFactory.getInstance("X.509")
val x509Certificate: X509Certificate =
cf.generateCertificate(pubCertInputStream) as X509Certificate //1

val keyPair = KeyPair(x509Certificate.publicKey,
loadPrivateKey()) //2

val handshakeCertificates: HandshakeCertificates =
HandshakeCertificates.Builder()
    .addPlatformTrustedCertificates()
    .heldCertificate(HeldCertificate(keyPair, x509Certificate))
    .build() //3

val client = OkHttpClient
    .Builder()
    .sslSocketFactory(handshakeCertificates.sslSocketFactory(),
handshakeCertificates.trustManager)
    .build() // 4
```

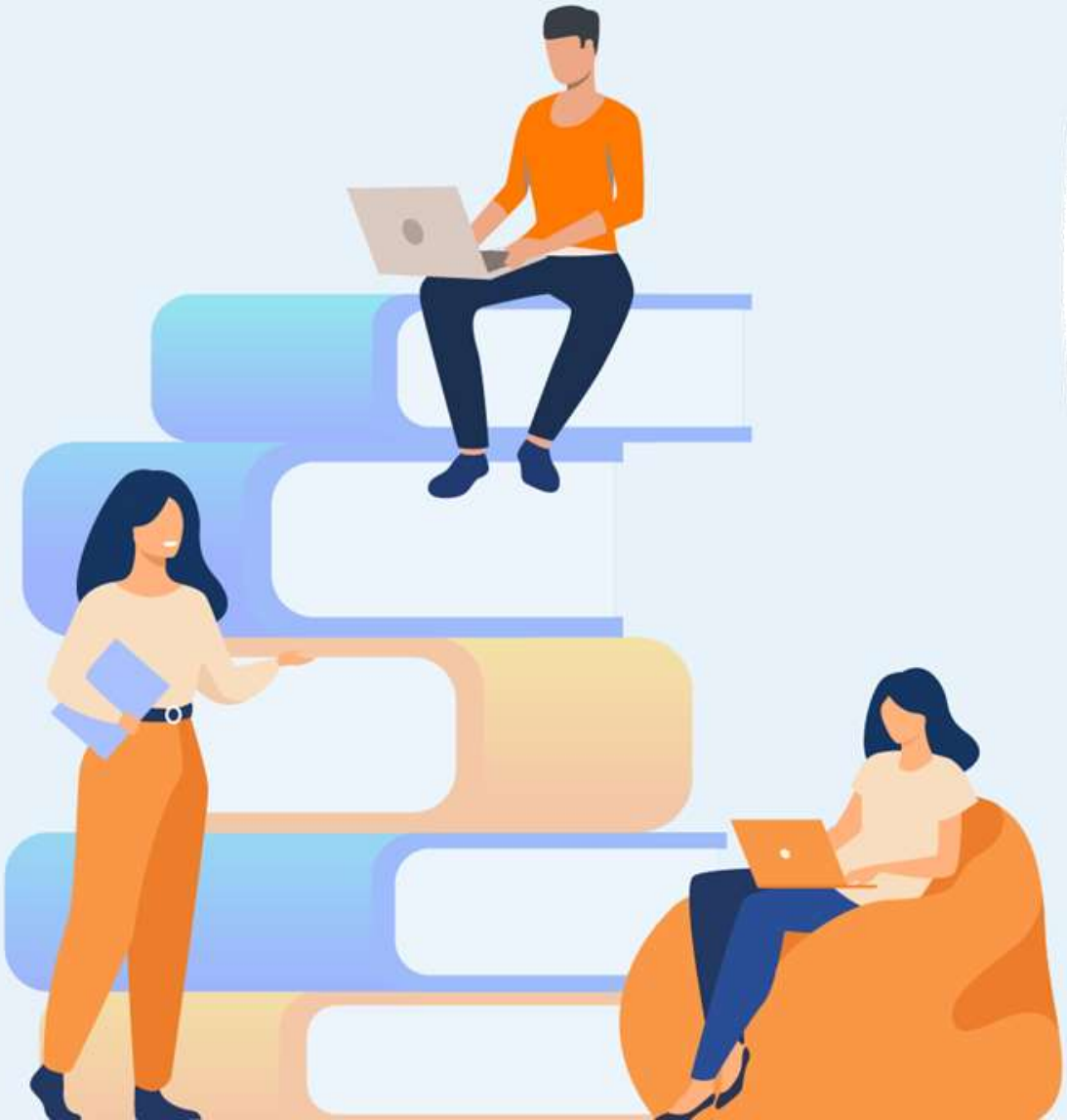
1. Créer CertificateFactory afin de générer un objet X509Certificate basé sur InputStream.
2. Créer KeyPair contenant à la fois des clés privées et publiques.
3. Construire l'objet HandshakeCertificates, qui contient toutes les autorités de certification approuvées par le système Android et votre paire de clés.
4. Construisez OkHttpClient avec votre sslSocketFactory et trustManager personnalisés.

Maintenant vous pouvez communiquer avec le serveur principal en utilisant Retrofit en mode SSL

```
val retrofit: Retrofit = retrofit.client(client).build()
```

PARTIE 1

Copyright - Tout droit réservé - OFPPT



PARTIE 2

Gérer les différentes méthodes de stockage de données sous Android

Dans ce module, vous allez :

- Gérer des fichiers sous Android
- Manipuler des bases de données sous Android
- Partager des données entre applications

18 heures



Activité 1

Création d’une application qui consomme une API et qui sauvegarde des données sur le téléphone

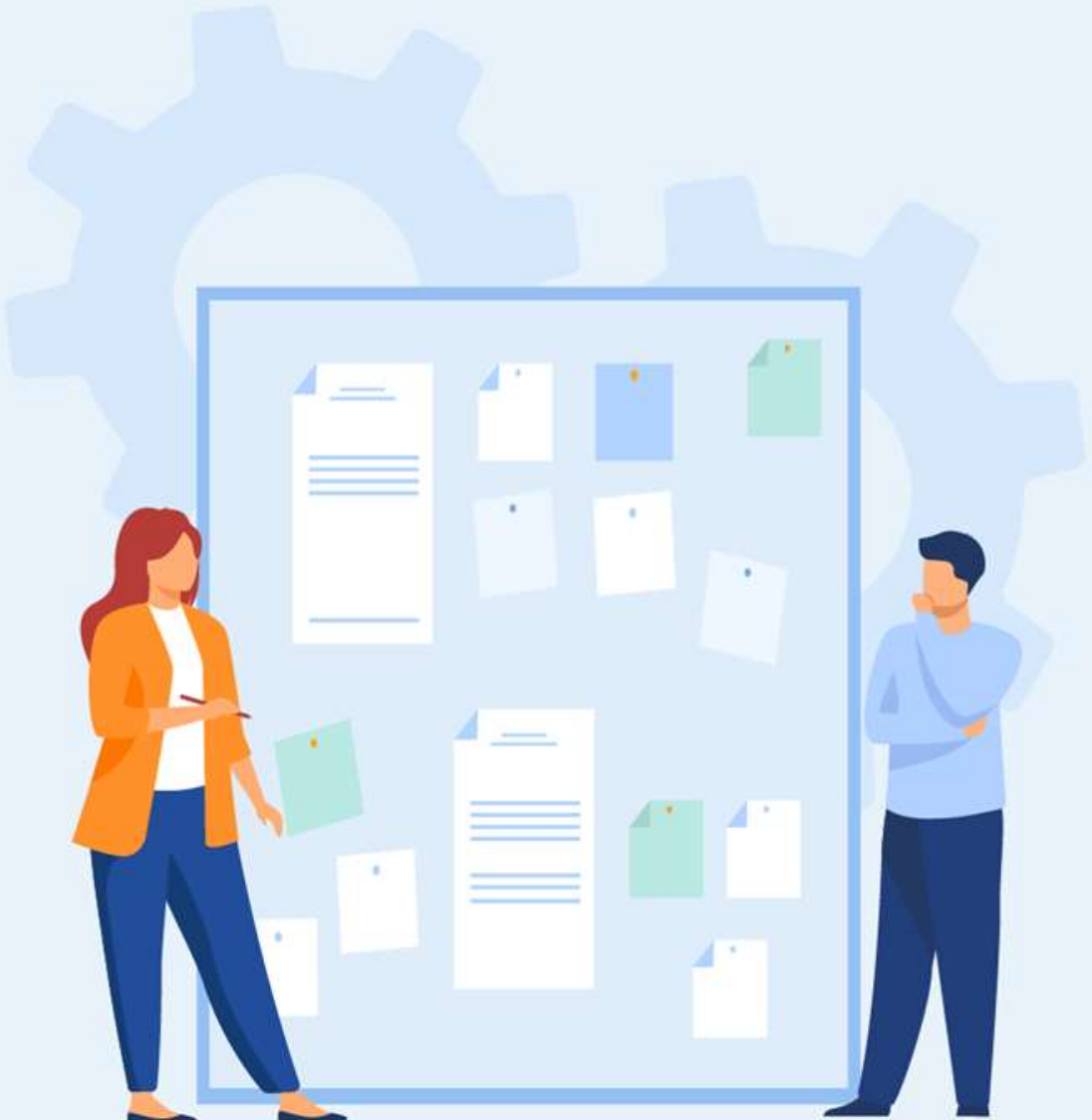
Compétences visées :

- Comprendre le protocole HTTP
- Consommer un API Rest/SOAP
- Manipuler la base de données Room SQLite
- Manipuler le SharedPreference

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l’IDE

10 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

2. Pour l'apprenant

- Créer un projet du travail
- Consommer un API (Rest/SOAP)
- Stocker les données sur téléphone

3. Conditions de réalisation :

- Support de résumé théorique
- Android Studio installé
- Mettre en place Room SQLite

4. Critères de réussite :

- Le stagiaire est-il capable de :
  Consommer un API (Rest/SOAP)
  Manipuler les données stockés sur le téléphone

Activité 1

Création d'une application qui consomme une API et qui sauvegarde des données sur le téléphone

PARTIE 2

SQLite Room

L'application consomme un API Rest du site e-commerce, et stocke les informations dans la base de données SQLite Room.

```
override fun onCreate(savedInstanceState:
Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val url =
"https://dummyjson.com/products/3"
 Response.ErrorListener { error ->
Log.i("TAG", "Error :$error") }

    val mRequestQueue =
Volley.newRequestQueue(getApplicationCont
ext());
    val jsObjRequest: JsonObjectRequest =
object : JsonObjectRequest(
        Method.GET, url, null,
        Response.Listener<JSONObject?> {
response ->
            println(response)
            handleProduct(response)
        },
        Response.ErrorListener { /**/ }) {
    }
    mRequestQueue.add(jsObjRequest)
}
```

MainActivity

```
private fun handleProduct(response: JSONObject) {
    val productDAO =
ProductDatabase.getDatabase(application).productDAO()

    println("title : "+
response.get("title").toString())
    println("price : "+
response.get("price").toString())
    Thread {
        val parsedProduct =
Product(response.get("id").toString().toInt() ,
            response.get("title").toString(),
            response.get("description").toString(),
            response.get("price").toString().toInt()
        )
        productDAO.insertProduct(parsedProduct)
        println(" size : "+productDAO.getAll()?.size)
        productDAO.delete(parsedProduct)
    }.start() }
```

Remarques

- le **Thread** utilisé pour accéder à la base de données, on peut le remplacer par une **coroutine**

Copyright - Tout droit réservé - OFPPT

Activité 1

Création d'une application qui consomme une API et qui sauvegarde des données sur le téléphone

SQLite Room

```
@Dao
interface ProductDAO {

    @Query("SELECT * FROM product")
    fun getAll(): List<Product?>?
    @Insert(onConflict =
OnConflictStrategy.REPLACE)
    fun insertProduct(product:
Product?)
    @Delete
    fun delete(product: Product?)
}
```

ProductDAO

Remarques

- **Synchronized** : bloque l'appel à la méthode du thread suivant tant que l'exécution du thread précédent n'est pas terminée.

```
@Database(entities = [Product::class], version = 1,
exportSchema = false)
abstract class ProductDatabase : RoomDatabase() {

    abstract fun productDAO(): ProductDAO
    companion object {
        @Volatile
        private var INSTANCE:ProductDatabase? = null

        fun getDatabase(context:
Context):ProductDatabase{
            val tempInstance = INSTANCE
            if (tempInstance != null) {
                return tempInstance
            }
            synchronized(this) {
                val instance =
Room.databaseBuilder(context.applicationContext,

ProductDatabase::class.java,"database").build()
                INSTANCE = instance
                return instance
            }}}}
```

ProductDatabase

```
@Entity(tableName =
"product")
class Product {

    @PrimaryKey
    var id : Int = 0
    var title : String = ""
    var description : String
= ""
    var price : Int = 0

    constructor(id: Int,
title: String, description:
String, price: Int) {
        this.id = id
        this.title = title
        this.description =
description
        this.price = price
    }}
```

Product

PARTIE 2

Copyright - Tout droit réservé - OFPPT

SQLite Room

Pour charger les ressources de la librairie SQLite Room, vous devez :

- Ajouter le plugin '**kotlin-kapt**' dans le fichier Build.gradle →

```
id 'kotlin-kapt'
```

- Ajouter la librairie dans le fichier Build.gradle

```
implementation "androidx.room:room-runtime:2.5.0"
kapt "androidx.room:room-compiler:2.5.0"
```

```
I/System.out: title : Samsung Universe 9
I/System.out: price : 1249
I/System.out:  size : 1
```

L'exécution

Remarques

- **Kapt** (l'outil de traitement des annotations Kotlin) vous permet d'utiliser des processeurs d'annotations Java avec du code Kotlin, même si ces processeurs ne sont pas spécifiquement compatibles avec Kotlin.

Copyright - Tout droit réservé - OFPPT

Activité 1

Création d'une application qui consomme une API et qui sauvegarde des données sur le téléphone

SharedPreferences

L'application consomme un API Rest du site e-commerce, et stocke les informations dans un SharedPreference.
il faut exécuter la méthode **commit()** pour enregistrer la valeur d'entrée.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val url = "https://dummyjson.com/products/3"
 Response.ErrorListener { error -> Log.i("TAG", "Error :$error") }

    val mRequestQueue = Volley.newRequestQueue(getApplicationContext());
    val jsObjRequest: JsonObjectRequest = object : JsonObjectRequest(
        Method.GET, url, null,
        Response.Listener<JSONObject?> { response ->
            val sharedPreferences = application.getSharedPreferences("MysharedPreference",
Context.MODE_PRIVATE)

sharedPreferences.edit().putString("title",response.get("title").toString()).commit()

sharedPreferences.edit().putString("price",response.get("price").toString()).commit()

        },
        Response.ErrorListener { /**/ }) {

    }
    mRequestQueue.add(jsObjRequest)
}
```

PARTIE 2

Copyright - Tout droit réservé - OFPPT

SharedPreferences

La lecture des données stockées dans le SharedPreference, le 2ème paramètre présente la valeur par défaut si la valeur attendue est introuvable .

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

val sharedPreferences =
application.getSharedPreferences("MysharedPreferen
ce", Context.MODE_PRIVATE)

      val title =
sharedPreferences.getString("title","")
      val price =
sharedPreferences.getString("price","")

      println(title)
      println(price)
}
```

Mode	Description
MODE_APPEND	Cela ajouta les nouvelles préférences aux préférences déjà existantes
MODE_ENABLE_WRITE_AHEAD_LOGGING	Indicateur d'ouverture de la base de données. Lorsqu'il est défini, il a fait la journalisation en écriture anticipée par défaut
MODE_MULTI_PROCESS	Cette méthode vérifiera la modification des préférences même si l'instance de la préférence partagée a déjà été chargée
MODE_PRIVATE	En définissant ce mode, le fichier n'est accessible qu'à l'aide de l'application appelante
MODE_WORLD_READABLE	Ce mode permet à une autre application de lire les préférences
MODE_WORLD_WRITEBLE	Ce mode permet à une autre application d'écrire les préférences

Les modes utilisés dans le SharedPreference

Copyright - Tout droit réservé - OFPPT



Activité 2

Sauvegarder des données dans des fichiers sur le téléphone

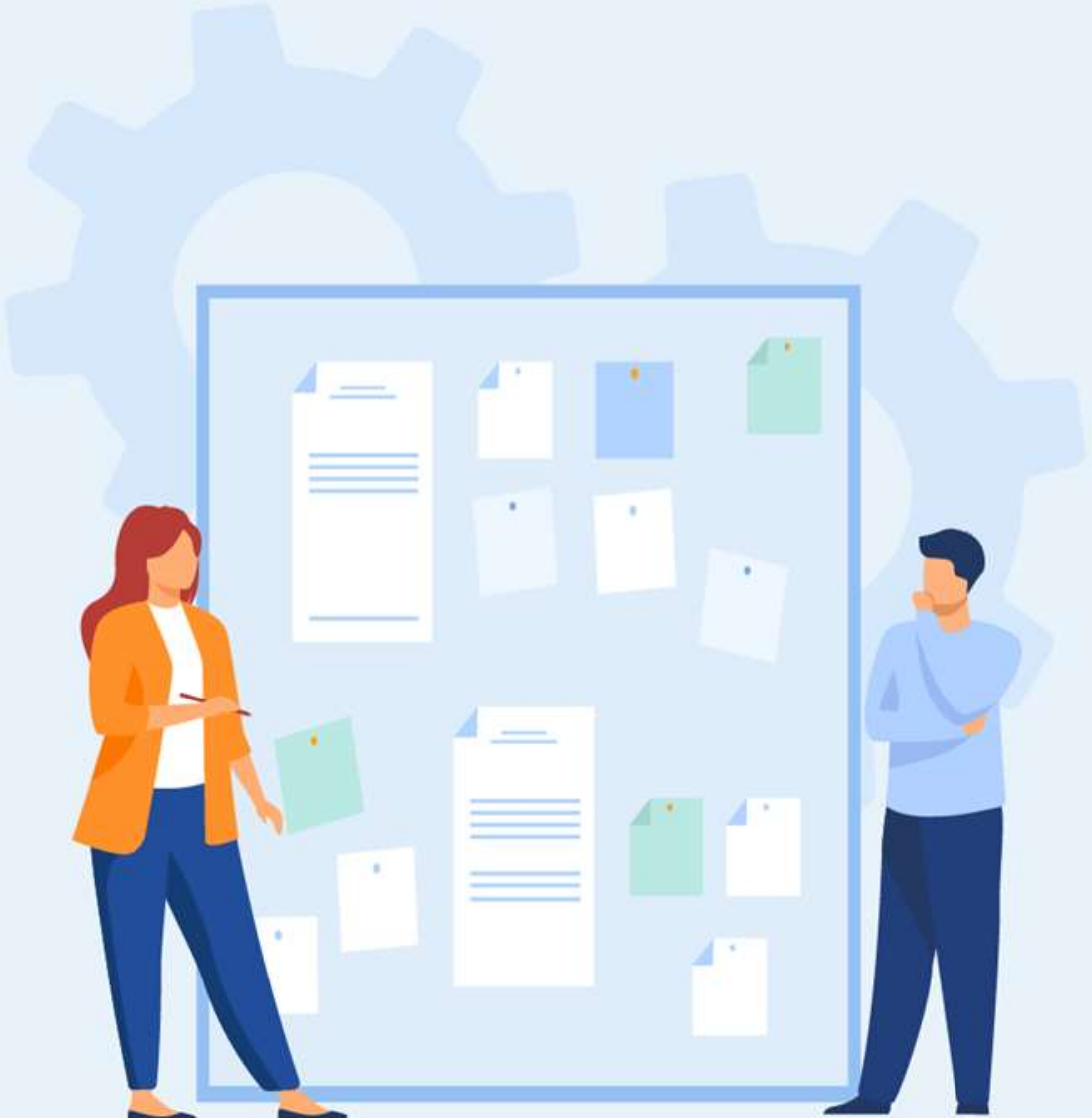
Compétences visées :

- Comprendre le protocole HTTP
- Consommer un API Rest/SOAP
- Manipuler les fichiers

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l'IDE

08 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

2. Pour l'apprenant

- Créer un projet du travail
- Consommer un API (Rest/SOAP)
- Stocker les données sur téléphone
- Manipuler les fichiers sur téléphone

3. Conditions de réalisation :

- Support de résumé théorique
- Android Studio installé

4. Critères de réussite :

- Le stagiaire est-il capable de :
  Consommer un API (Rest/SOAP)
  Manipuler les données stockés sur le téléphone

Activité 2

Sauvegarder des données dans des fichiers sur le téléphone

Manipuler les fichiers

L'application consomme un API Rest du site e-commerce, et stocke les informations dans un fichier.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val url = "https://dummyjson.com/products/3"
 Response.ErrorListener { error -> Log.i("TAG", "Error
:$error") }

    val mRequestQueue =
Volley.newRequestQueue(getApplicationContext());
    val jsObjRequest: JsonObjectRequest = object :
JsonObjectRequest(
        Method.GET, url, null,
        Response.Listener<JSONObject?> { response ->
            writeToFile("Title "
+response.get("title").toString()
    +", price "+response.get("price").toString(),
this@MainActivity)
println( readFromFile(this@MainActivity))

        },
        Response.ErrorListener { /**/ }) {

    }
    mRequestQueue.add(jsObjRequest)
}
```

```
private fun writeToFile(data: String,
context: Context) {
    try {
        val outputStreamWriter =

OutputStreamWriter(context.openFileOu
tput("config.txt", MODE_PRIVATE))
        outputStreamWriter.write(data)
        outputStreamWriter.close()
    } catch (e: IOException) {
        Log.e("Exception", "File write
failed: " + e.toString())
    }
}
```

PARTIE 2

Copyright - Tout droit réservé - OFPPT

Manipuler les fichiers

```
private fun readFromFile(context: Context): String? {
    var ret = ""
    try {
        val inputStream: InputStream? =
context.openFileInput("config.txt")
        if (inputStream != null) {
            val inputStreamReader = InputStreamReader(inputStream)
            val bufferedReader = BufferedReader(inputStreamReader)
            var receiveString: String? = ""
            val stringBuilder = StringBuilder()
            while (bufferedReader.readLine().also { receiveString =
it } != null) {
                stringBuilder.append("\n").append(receiveString)
            }
            inputStream.close()
            ret = stringBuilder.toString()
        }
    } catch (e: FileNotFoundException) {
        Log.e("login activity", "File not found: " + e.toString())
    } catch (e: IOException) {
        Log.e("login activity", "Can not read file: $e")
    }
    return ret
}
```

```
I/System.out: Title Samsung Universe 9, price 1249
```

L'exécution

Le fichier créé par l'application se trouve sur ce lien **path**.



Le système Android protège les fichiers créés par l'application, il faut avoir l'autorisation nécessaire pour le modifier.

PARTIE 2

Copyright - Tout droit réservé - OFPPT



Activité 3

Partager des données entre deux applications

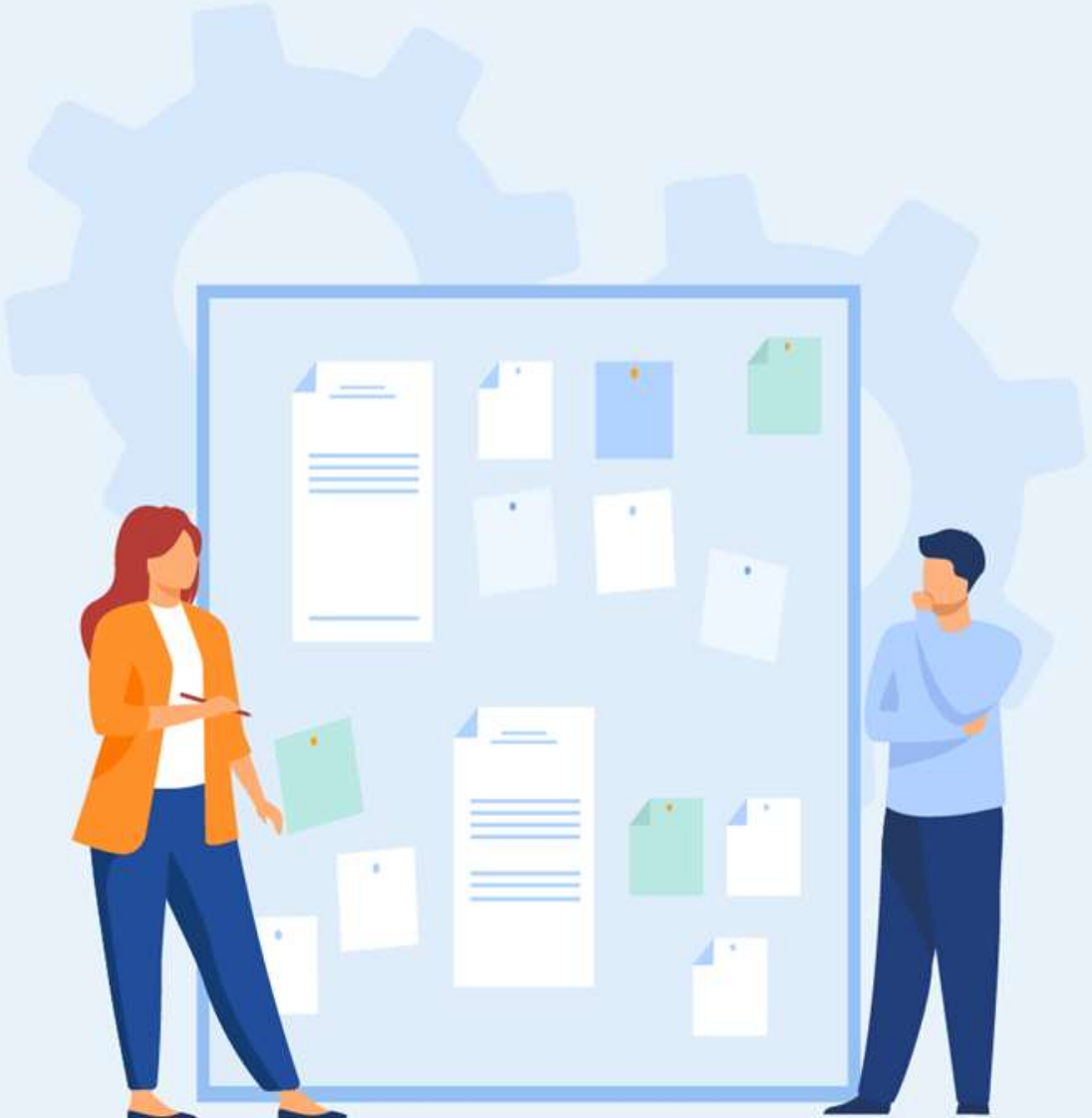
Compétences visées :

- Comprendre le protocole HTTP
- Consommer un API Rest/SOAP
- Manipuler la base de données Room SQLite
- Manipuler le SharedPreference

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l'IDE

08 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

2. Pour l'apprenant

- Créer un projet du travail
- Consommer un API (Rest/SOAP)
- Partager les données sur téléphone

3. Conditions de réalisation :

- Support de résumé théorique
- Android Studio installé
- Mettre en place ContentProvider

4. Critères de réussite :

- Le stagiaire est-il capable de :
  Implémenter le ContentProvider
  Partager les données stockés sur le téléphone

Activité 3

Partager des données entre deux applications

Content Provider

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val url = "https://dummyjson.com/products/3"
 Response.ErrorListener { error -> Log.i("TAG",
"Error :$error") }

    val mRequestQueue =
Volley.newRequestQueue(getApplicationContext());
    val jsObjRequest: JsonObjectRequest = object :
JsonObjectRequest( Method.GET, url, null,
        Response.Listener<JSONObject?> { response ->
                handleProduct(response)
        },
        Response.ErrorListener { /**/ }) {
    }
    mRequestQueue.add(jsObjRequest)}
```

```
private fun handleProduct(response: JSONObject) {
  val productDAO =
ProductDatabase.getDatabase(application).productDAO()


   Thread {
  val parsedProduct = Product(response.get("id").toString().toInt()
          response.get("title").toString(),
          response.get("description").toString(),
          response.get("price").toString().toInt()
      )
      productDAO.insertProduct(parsedProduct)
      val product = productDAO.getAll()?.get(0)
      val values = ContentValues()

values.put(MyContentProvider.name, product?.title.toString())
contentResolver.insert(MyContentProvider.CONTENT_URI, values)
   }.start()
}
```

On ajoute un **ContentProvider** dans le projet de manipulation d'API Rest pour partager les données avec une autre application.

PARTIE 2

Copyright - Tout droit réservé - OFPPT

Content Provider

```
class MyContentProvider : ContentProvider() {
   companion object {

  const val PROVIDER_NAME =
"com.example.myapplication.provider"

        const val URL = "content://$PROVIDER_NAME/products"

        // parsing the content URI
        val CONTENT_URI = Uri.parse(URL)
        const val id = "id"
        const val name = "name"
        const val uriCode = 1
        var uriMatcher: UriMatcher? = null
        private val values: HashMap<String, String>? = null

        const val DATABASE_NAME = "productsDB"
        const val TABLE_NAME = "products"
        const val DATABASE_VERSION = 1

        const val CREATE_DB_TABLE =
            (" CREATE TABLE " + TABLE_NAME
                    + " (id INTEGER PRIMARY KEY AUTOINCREMENT, "
                    + " name TEXT NOT NULL);")
```

```
init {
uriMatcher = UriMatcher(UriMatcher.NO_MATCH)
uriMatcher!!.addURI(PROVIDER_NAME,"products",uriCode)
uriMatcher!!.addURI(PROVIDER_NAME,"products/*",uriCode)
        } }

   override fun getType(uri: Uri): String? {
        return when (uriMatcher!!.match(uri)) {
            uriCode -> "vnd.android.cursor.dir/products"
            else -> throw
IllegalArgumentException("Unsupported URI: $uri")
        }
   }
   override fun onCreate(): Boolean {
  val context = context
 val dbHelper = DatabaseHelper(context)
        db = dbHelper.writableDatabase
        return if (db != null) {
            true
        } else false
   }
```

PARTIE 2

Copyright - Tout droit réservé - OFPPT

Activité 3

Partager des données entre deux applications

PARTIE 2

Content Provider

```
override fun query( uri: Uri, projection:
Array<String>?, selection: String?,
     selectionArgs: Array<String>?, sortOrder: String?
   ): Cursor? {
       var sortOrder = sortOrder
       val qb = SQLiteQueryBuilder()
       qb.tables = TABLE_NAME
       when (uriMatcher!!.match(uri)) {
           uriCode -> qb.projectionMap = values
           else -> throw
IllegalArgumentException("Unknown URI $uri")
       }
       if (sortOrder == null || sortOrder === "") {
           sortOrder = id
       }
       val c = qb.query(
           db, projection, selection, selectionArgs,
null,
           null, sortOrder
       )
       c.setNotificationUri(context!!.contentResolver,
uri)
       return c
   }
```

```
 // adding data to the database
   override fun insert(uri: Uri, values: ContentValues?): Uri?
{
       val rowID = db!!.insert(TABLE_NAME, "", values)
       if (rowID > 0) {
           val _uri =
               ContentUris.withAppendedId(CONTENT_URI, rowID)
           context!!.contentResolver.notifyChange(_uri, null)
           return _uri
       }
       throw SQLiteException("Failed to add a record into
$uri")
   }

   override fun update(
       uri: Uri, values: ContentValues?, selection: String?,
       selectionArgs: Array<String>?
   ): Int {
       var count = 0
       count = when (uriMatcher!!.match(uri)) {
           uriCode -> db!!.update(TABLE_NAME, values,
selection, selectionArgs)
           else -> throw IllegalArgumentException("Unknown URI
$uri")
       }
       context!!.contentResolver.notifyChange(uri, null)
       return count
   }
```

Copyright - Tout droit réservé - OFPPT

Content Provider

```
 override fun delete(
        uri: Uri,
        selection: String?,
        selectionArgs: Array<String>?
    ): Int {
        var count = 0
        count = when (uriMatcher!!.match(uri)) {
            uriCode -> db!!.delete(TABLE_NAME,
selection, selectionArgs)
            else -> throw
IllegalArgumentException("Unknown URI $uri")
        }
   context!!.contentResolver.notifyChange(uri, null)
        return count
    }
```

```
 private var db: SQLiteDatabase? = null

    // creating a database
    private class DatabaseHelper  // defining a constructor
    internal constructor(context: Context?) : SQLiteOpenHelper(
        context,DATABASE_NAME, null, DATABASE_VERSION
    ) {
        // creating a table in the database
        override fun onCreate(db: SQLiteDatabase) {
            db.execSQL(CREATE_DB_TABLE)
        }

        override fun onUpgrade(
            db: SQLiteDatabase,
            oldVersion: Int,
            newVersion: Int
        ) {
            db.execSQL("DROP TABLE IF EXISTS $TABLE_NAME")
            onCreate(db)
        }
    }
}
```

PARTIE 2

Copyright - Tout droit réservé - OFPPT

Content Provider

Pour utiliser le ContentProvider il faut l'ajouter dans le fichier AndroidManifest aussi.
Le projet actuel partage la valeur de **title** avec une deuxième application.

```
<provider
    android:name="com.example.myapplication.MyContentProvider"
    android:authorities="com.example.myapplication.provider"
    android:enabled="true"
    android:exported="true"/>
```

```
I/System.out: title : Samsung Universe 9
I/System.out: price : 1249
```

L'exécution

Copyright - Tout droit réservé - OFPPT

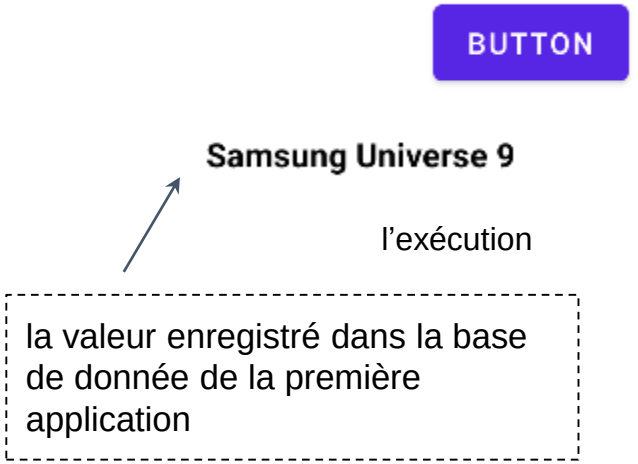
Activité 3

Partager des données entre deux applications

Le partage des données

```
findViewById<Button>(R.id.button)
    .setOnClickListener({val cursor =
getContentResolver().query(Uri.parse("content://com.example.myappl
ication.provider/products"), null, null, null, null);

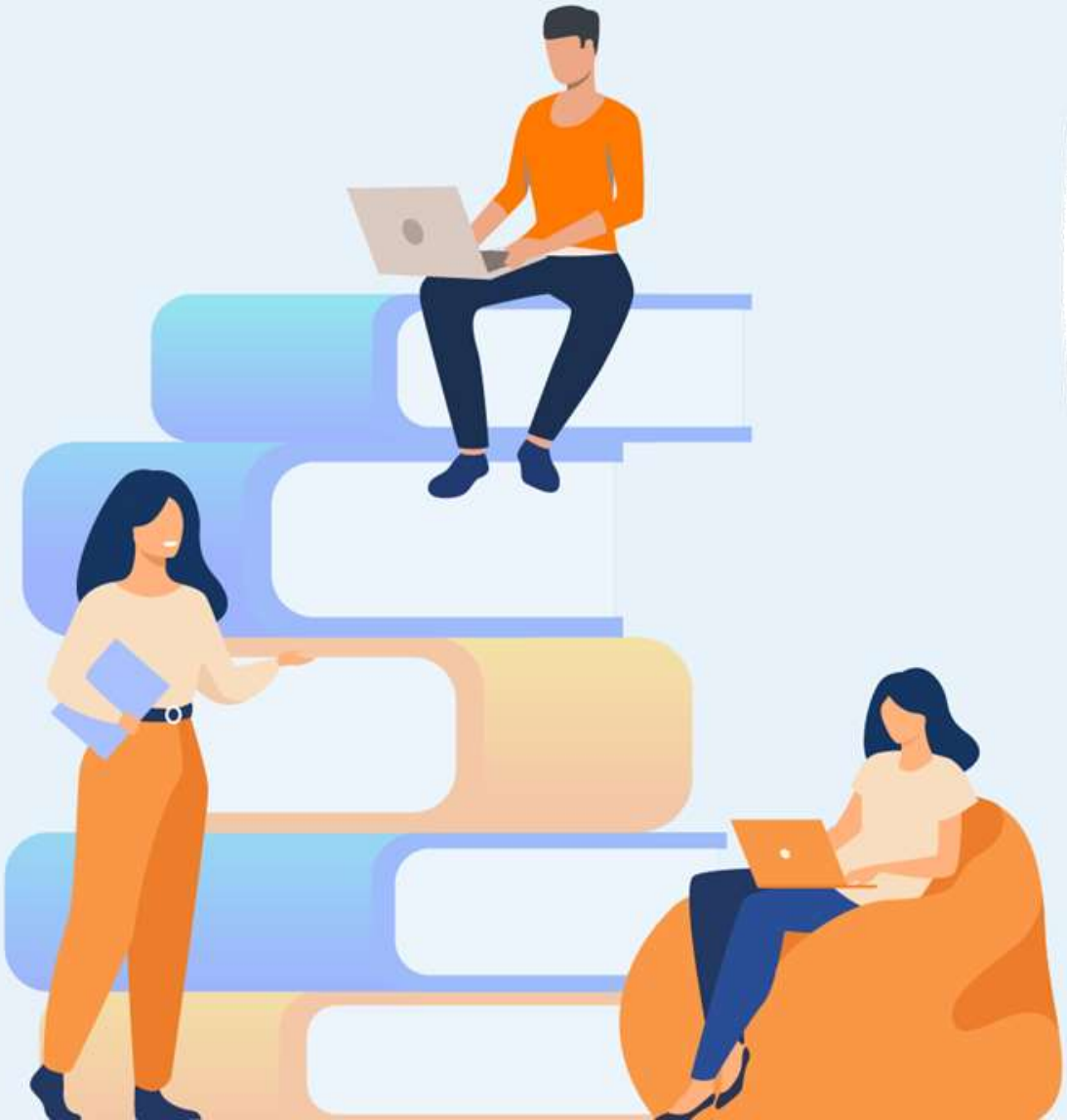
        if (cursor != null && cursor!!.moveToFirst()) {
            val strBuild = StringBuilder()
            while (!cursor!!.isAfterLast) {
                strBuild.append("""
${cursor.getString(cursor.getColumnIndex("name"))}""".trimIndent()
)
                cursor.moveToNext()
            }
       ((findViewById<TextView>(R.id.text)) as
(TextView)).setText(strBuild)
        }
    })
```



La 2ème application reçoit les valeurs partagées par la première application en utilisant le **ContentProvider**

PARTIE 2

Copyright - Tout droit réservé - OFPPT



PARTIE 3

Publier une application sur le store

Dans ce module, vous allez :

- Gérer des signatures et versions
- Publier d’une application sur le Play store
- Utiliser des plateformes de déploiements continus

16 heures



Activité 1

Gérer les signatures et versions

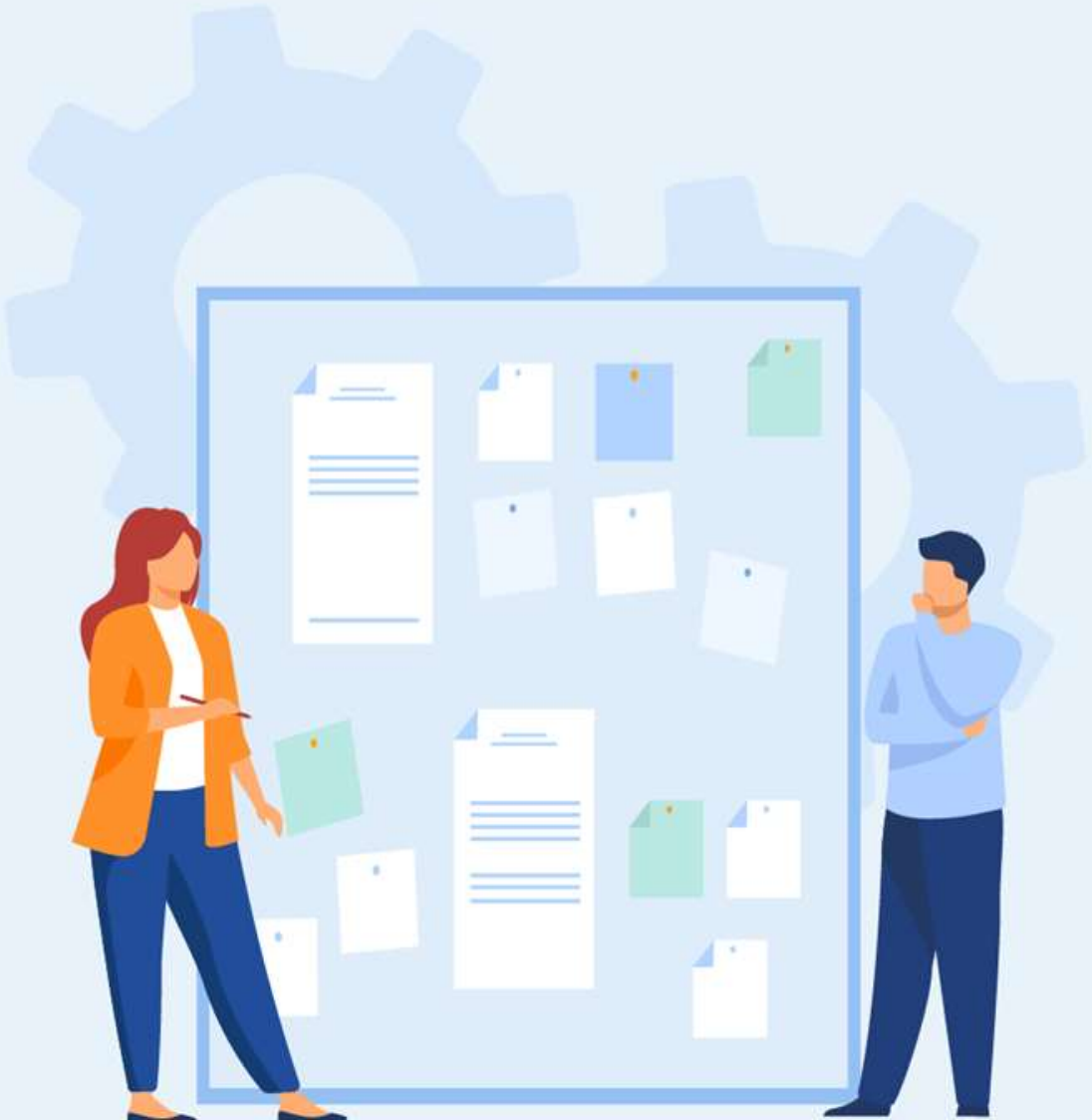
Compétences visées :

- Signer une application
- Préparer une version de Play Store

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l’IDE

06 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

2. Pour l'apprenant

- Créer un projet du travail
- Créer un KeyStore
- Générer un APK Signé
- Déployer l'AAB

3. Conditions de réalisation :

- Support de résumé théorique
- Android Studio installé
- Compte Play Console

4. Critères de réussite :

- Le stagiaire est-il capable de :
  - Créer un KeyStore (Automatique et Manuel)
  - Générer la version release de l'application
  - Déployer l'AAB dans le Play Store

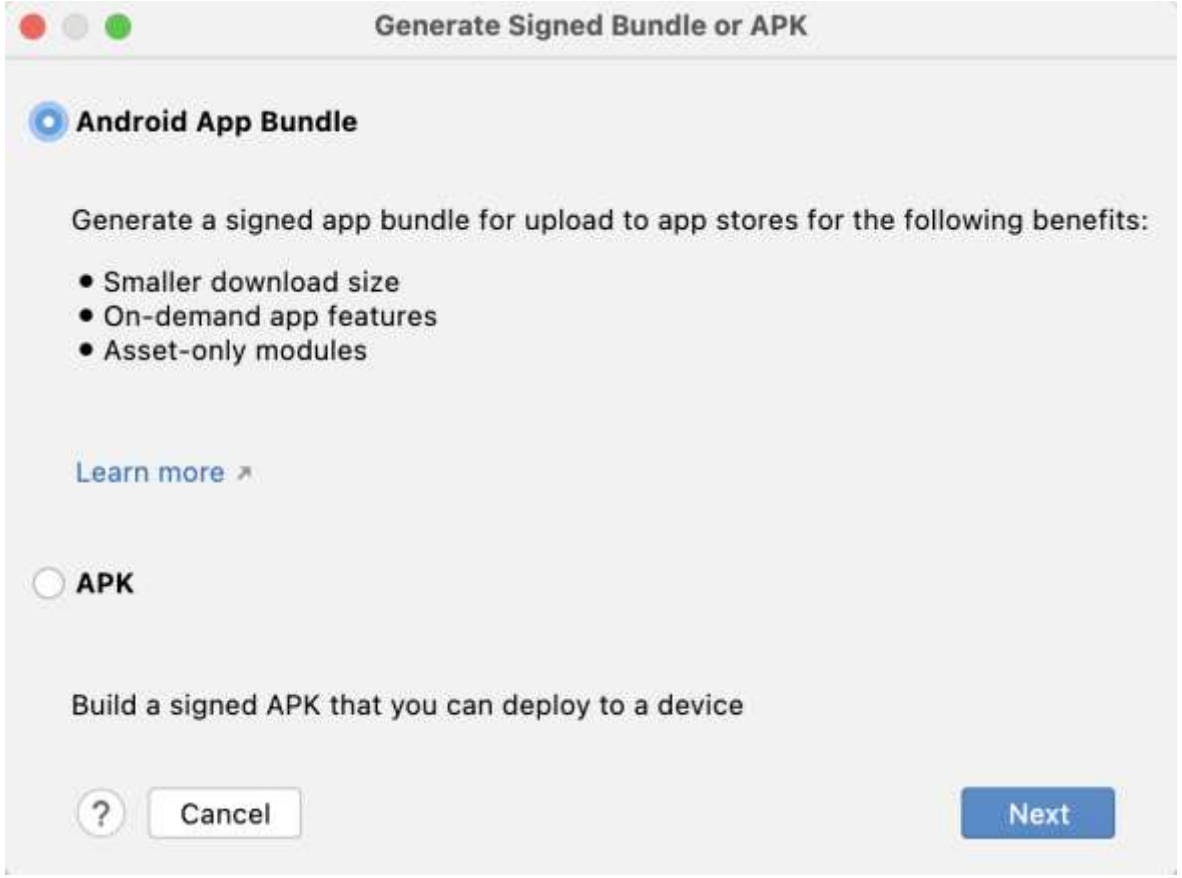
Générer un APK signé

APK : Android Package est un format de fichiers pour le système d'exploitation Android. sous forme une collection de fichiers pour Android assemblés et compressés en ZIP.

AAB : Android App Bundle est un format de fichier conçu pour la publication d'applications Android sur les magasins d'applications. Il doit inclure le code compilé de l'application ainsi que ses ressources.



source : huaweicentral.com



PARTIE 3

Copyright - Tout droit réservé - OFPPT

Générer un Key Store

Il y a 2 méthodes pour générer l'upload key et keystore :

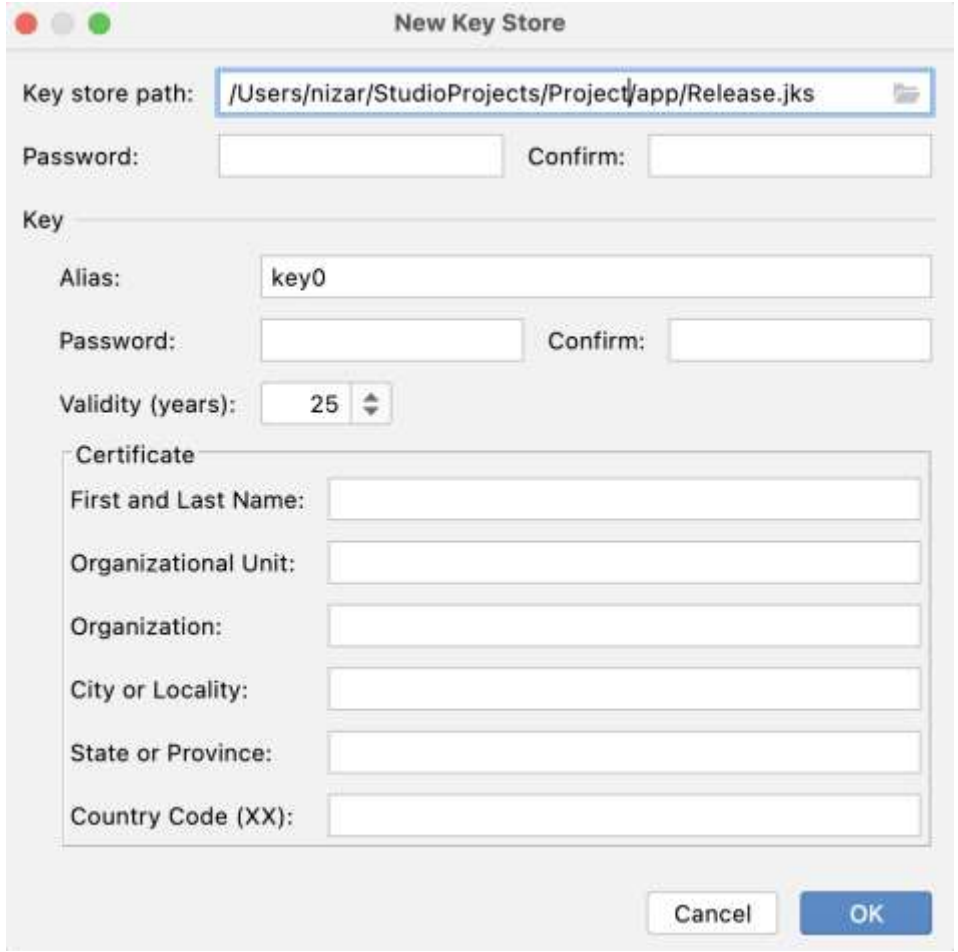
- **Générez une clé en utilisant Android Studio :**
  comme indiqué sur la capture d'écran

- **Générez une clé manuellement :**

  On exécute cette commande dans le terminal :

keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -keysize 2048 -validity 18250

« **my-release-key.keystore** » : votre fichier de clés.
« **validity** » : durée de validité de la clé. 50 ans ~ 18250 jours.



PARTIE 3

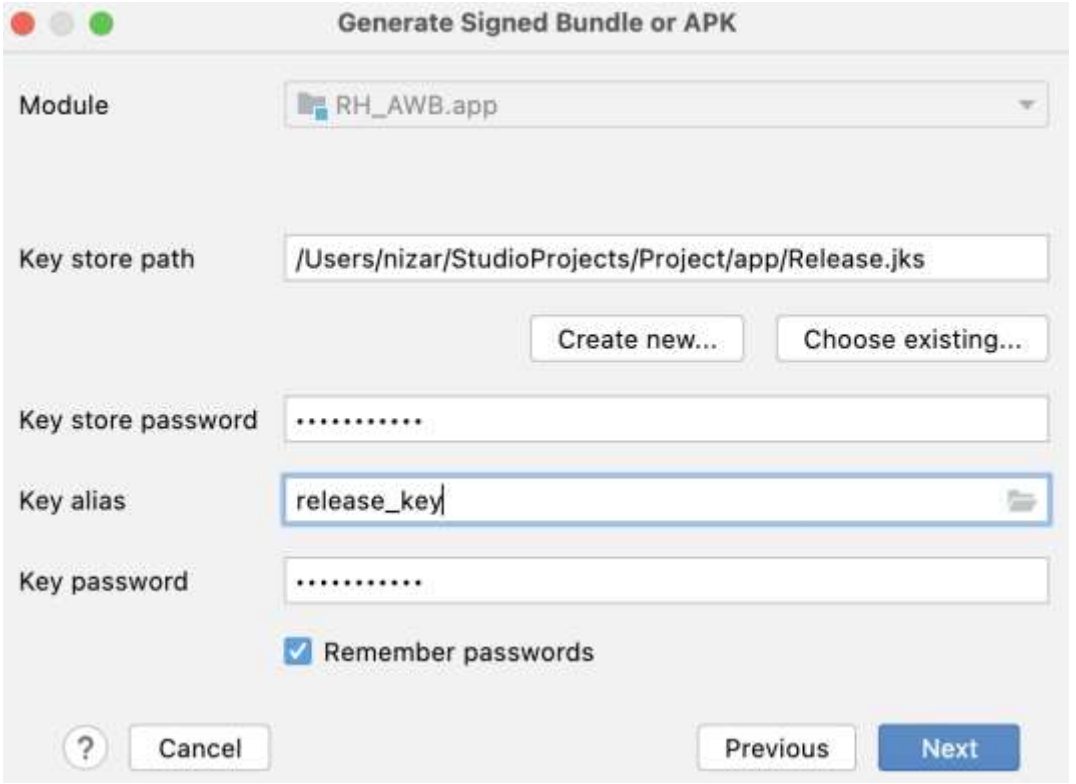
Copyright - Tout droit réservé - OFPPT

Signer une application

Pour signer une application, nous pouvons utiliser 2 méthodes :

- Android Studio
- Code sur Build.gradle

```
signingConfigs {
    releaseconfig {
        keyAlias 'release_key'
        keyPassword 'password'
        storeFile file('Release.jks')
        storePassword 'password'
    }
}
```



Copyright - Tout droit réservé - OFPPT

Product Flavors

Les saveurs de produits Android Studio sont une fonctionnalité puissante, le plugin Android Gradle permet d'échanger des classes Java/Kotlin au moment de la compilation et ne nécessite pas de bibliothèques supplémentaires.

Cet exemple de productFlavors **(dev** et **prod)** est dans le même projet, mais chaque application générée possède ses propres ressources.

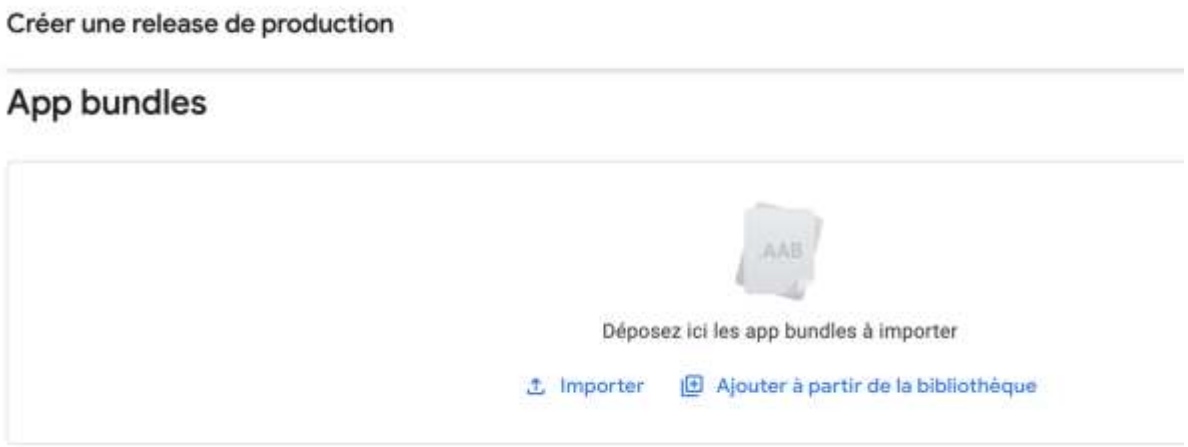
```
productFlavors {
    flavorDimensions "version"
    dev {
        applicationIdSuffix ".dev"
        dimension "version"
        resValue "string", "app_name", "APP-DEV"
    manifestPlaceholders = [appIcon: "@mipmap/ic_launcher"]
    }
    prod {
        dimension "version"
        resValue "string", "app_name", "APP-PROD"
     manifestPlaceholders = [ appIcon: "@mipmap/ic_launcher"]
    }
}
```


Copyright - Tout droit réservé - OFPPT

Déployer manuellement l'AAB

Pour déployer manuellement l'AAB (Android App Bundle), il faut :

- Préparer l'application en mode **release** et **signée.**
- Remplir les informations de l'application sur le PlayStore.
- Uploader les captures d'écrans.



Copyright - Tout droit réservé - OFPPT



Activité 2

Utiliser des plateformes d’intégration et déploiements continus

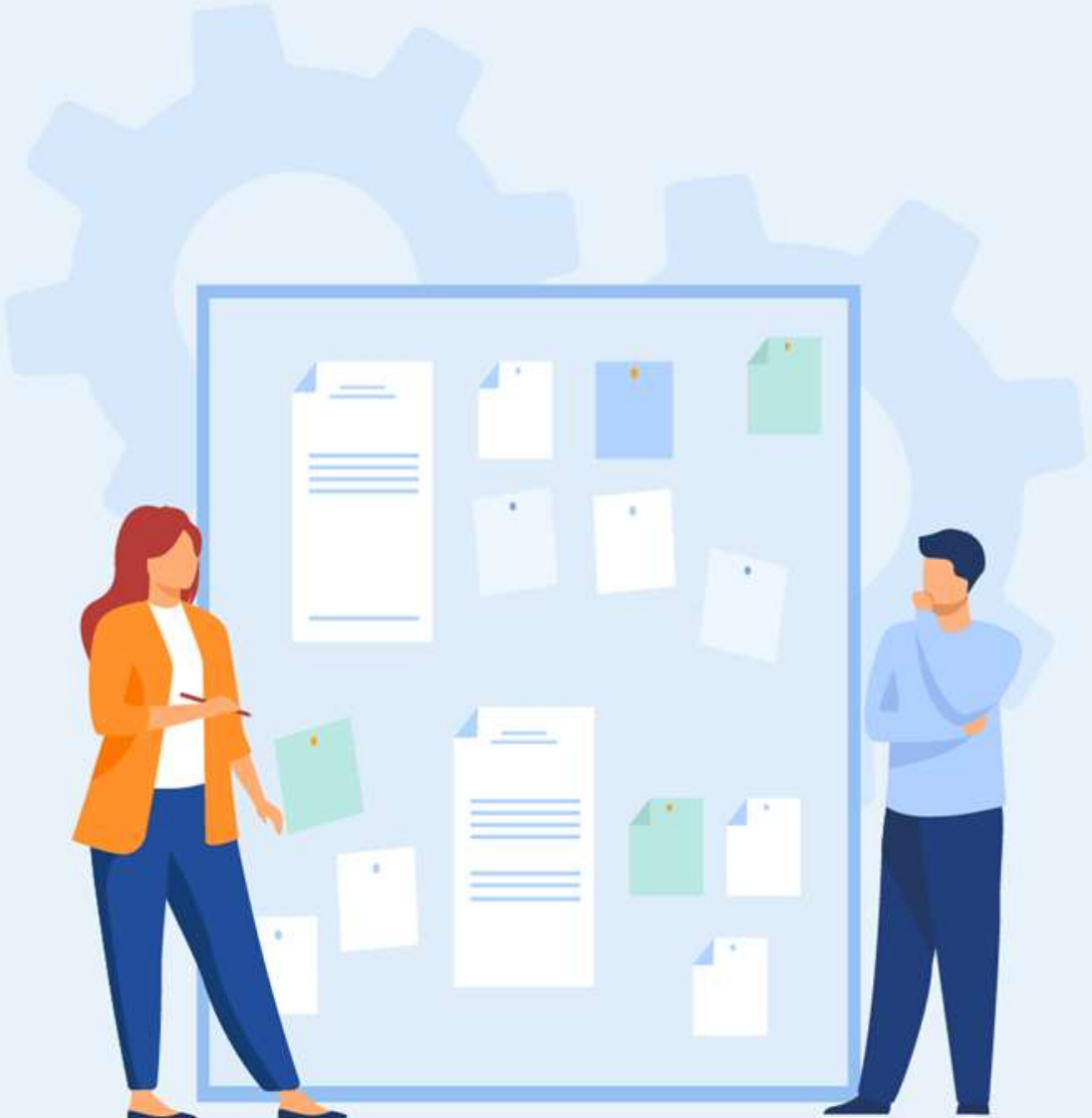
Compétences visées :

- Mettre en place le versionning
- Automatiser le déploiement sur Play Store

Recommandations clés :

- Suivre les instructions du TP
- Préparer la machine pour réaliser le TP
- Exécuter les codes sur l’IDE

10 heures



CONSIGNES

1. Pour le formateur

- Demander aux apprenants de suivre les étapes de l'énnoncé et exécuter le code fourni
- Demander aux apprenants de réaliser le travail sur leurs machines

2. Pour l'apprenant

- Créer un projet du travail
- Manipuler le versionning (GIT)
- Mettre en place un outil CI/CD

3. Conditions de réalisation :

- Support de résumé théorique
- Android Studio installé
- Compte GIT (Gitlab/GitHub)
- Compte Play Console
- Outil CI/CD (Fastlane)

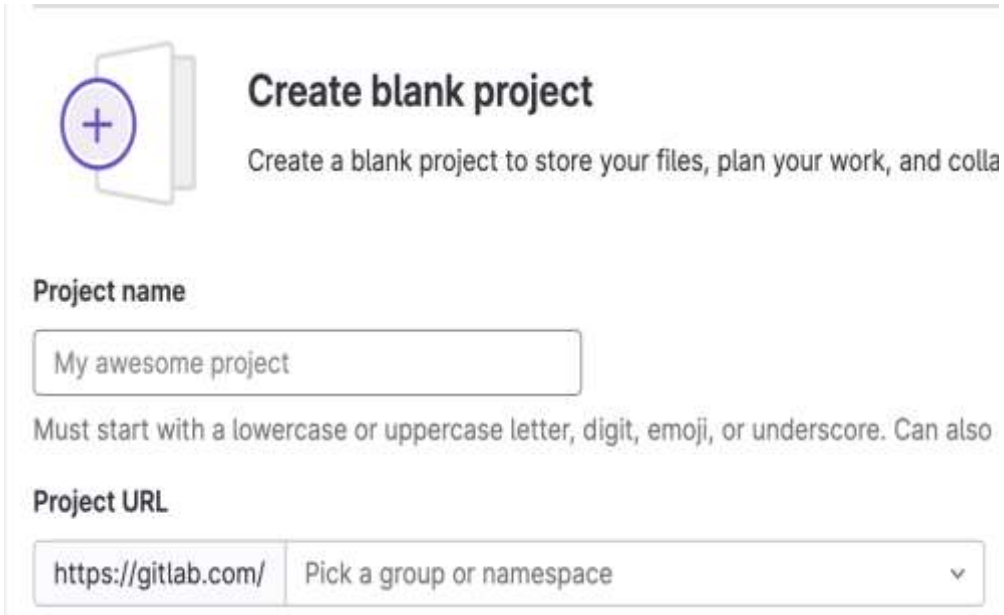
4. Critères de réussite :

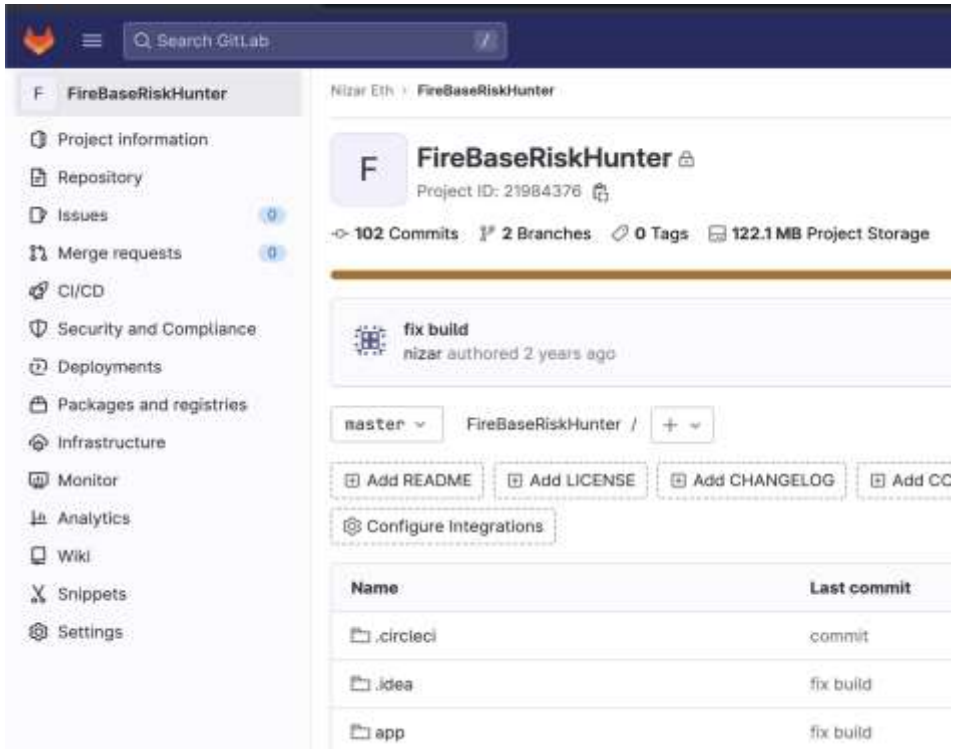
- Le stagiaire est-il capable de :
  - Manipuler le versionning (Git : Branches, Commits…)
  - Déployer l'AAB dans le Play par un outil de CI/CD

PARTIE 3

Gitlab

Après la création du projet sur Android Studio, il faut créer un projet sur Gitlab pour assurer le travail collaboratif et mémoriser les versions.





Une fois le push est réussi, vous pouvez trouver le code source de votre projet sur Gitlab

Copyright - Tout droit réservé - OFPPT

Configuration de Fastlane

1. Configurer Fastlane dans votre projet
2. Prendre et enregistrer automatiquement des captures d'écran
3. Distribuer l'application pour les bêta-testeurs
4. Télécharger les supports marketing et les notes de version de l'application
5. Déployez le bundle fini sur le Google Play Store

La mise en place de Fastlane

installer Fastlane :

sudo gem install -n /usr/local/bin fastlane --verbose

Exécuter Fastlane :

fastlane build

Fichier Fastfile

```
default_platform(:android)

platform :android do
 desc "Runs all the tests"
 lane :test do
   gradle(task: "test")
 end

 desc "Submit a new Beta Build to Firebase App Distribution"
 lane :beta do
   gradle(task: "clean assembleRelease")
   firebase_app_distribution
 end

 desc "Deploy a new version to Google Play"
 lane :deploy do
   gradle(task: "clean assembleRelease")
   upload_to_play_store
 end
end
```

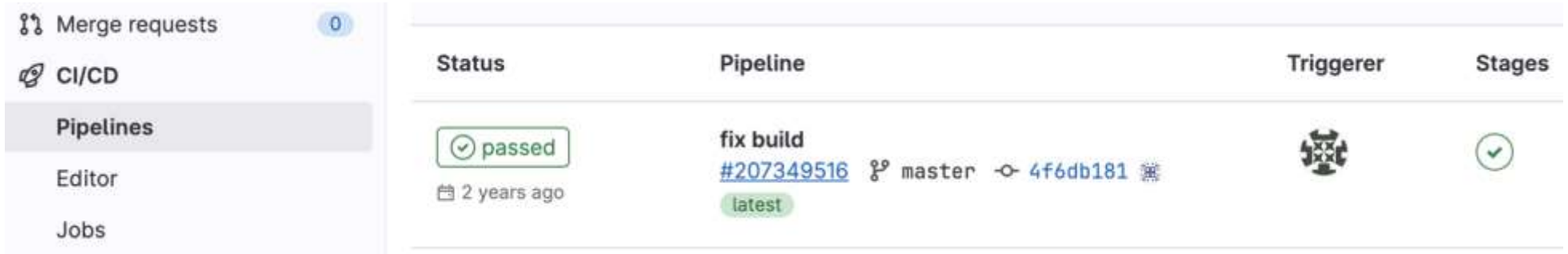
Fastlane clean et assemble l'application, en construisant que la version release.
La signature se fera conformément à la déclaration dans la clôture de la release.

PARTIE 3

Copyright - Tout droit réservé - OFPPT

CI/CD (Pipeline)

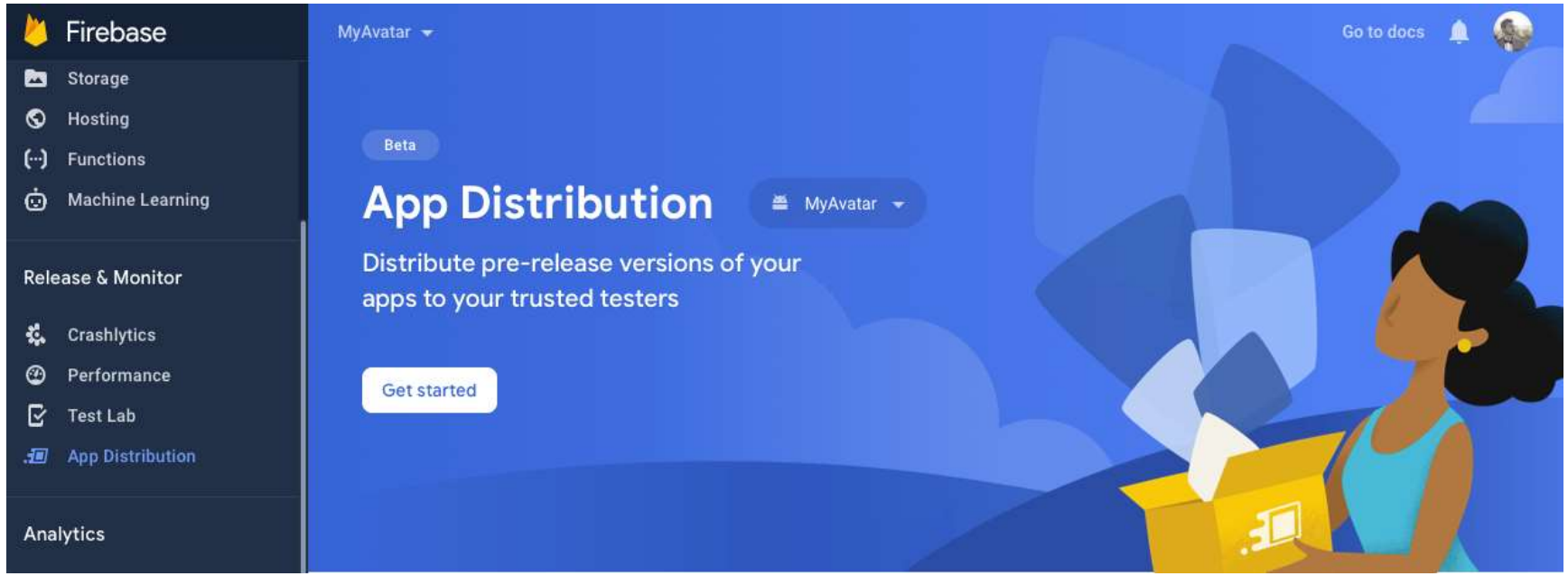
A l'arrivée d'une nouvelle version du code source par un push (Correction d'un bug ou implémentation d'une nouvelle fonctionnalité..), Gitlab exécute le fichier Fastfile pour faire un clean et assemblage de l'application, ce qui permet le démarrage du build d'une nouvelle version exécutable.



Copyright - Tout droit réservé - OFPPT

Firebase App Distribution

Avant de faire un déploiement dans le Play Store, le développeur est censé envoyer automatiquement à travers Fastlane une version à Firebase App distribution, pour faire les tests en interne par les bêta-testeurs :



Remarque

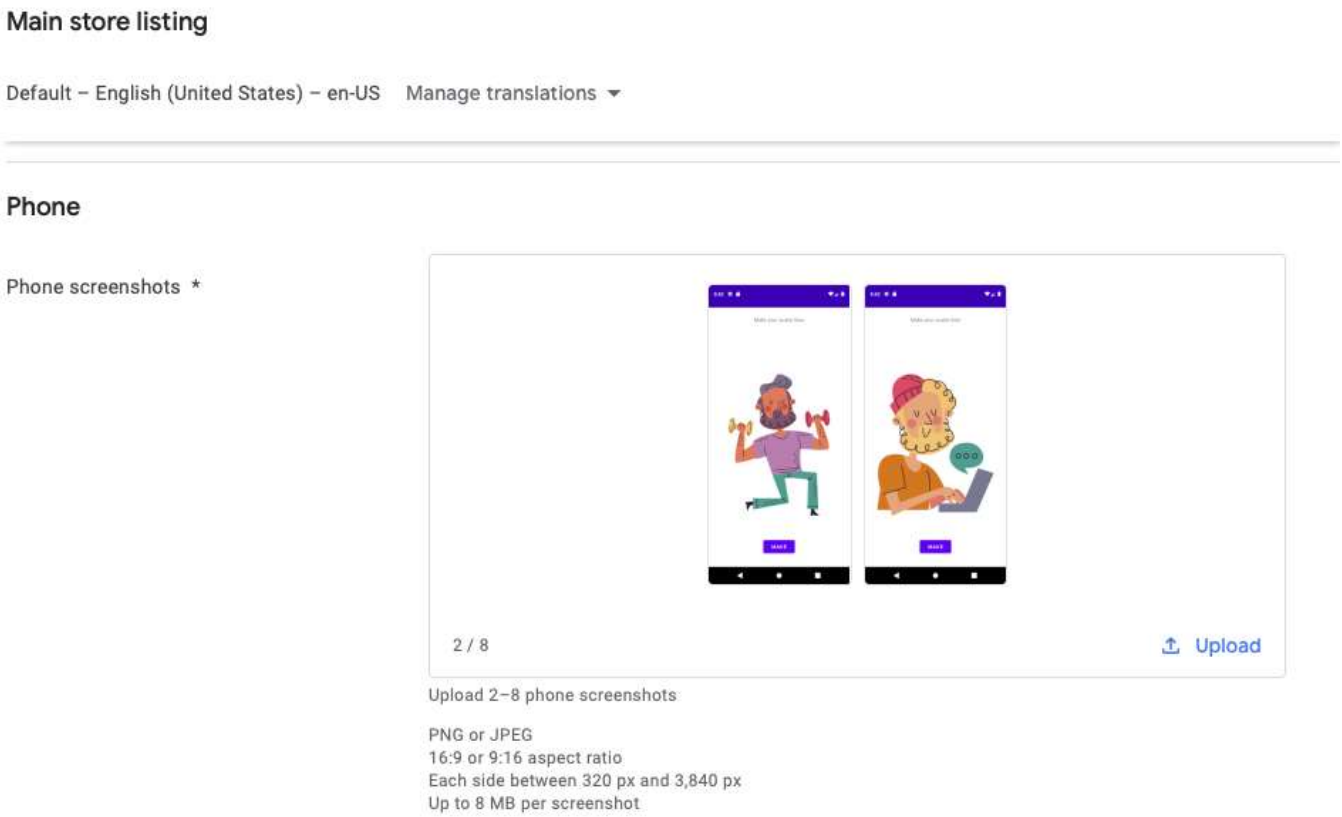
- **Firebase** est un ensemble d'outils pour l'hébergement et le développement d'applications mobiles et web, qui permet l'envoi de notifications et de publicités, la remontée des erreurs et des clics effectués dans l'application.

Copyright - Tout droit réservé - OFPPT

Déploiement des captures d'écrans

Pour télécharger des captures d'écran, il faut exécuter cette commande pour générer des captures d'écran :

fastlane capture_screen



PARTIE 3

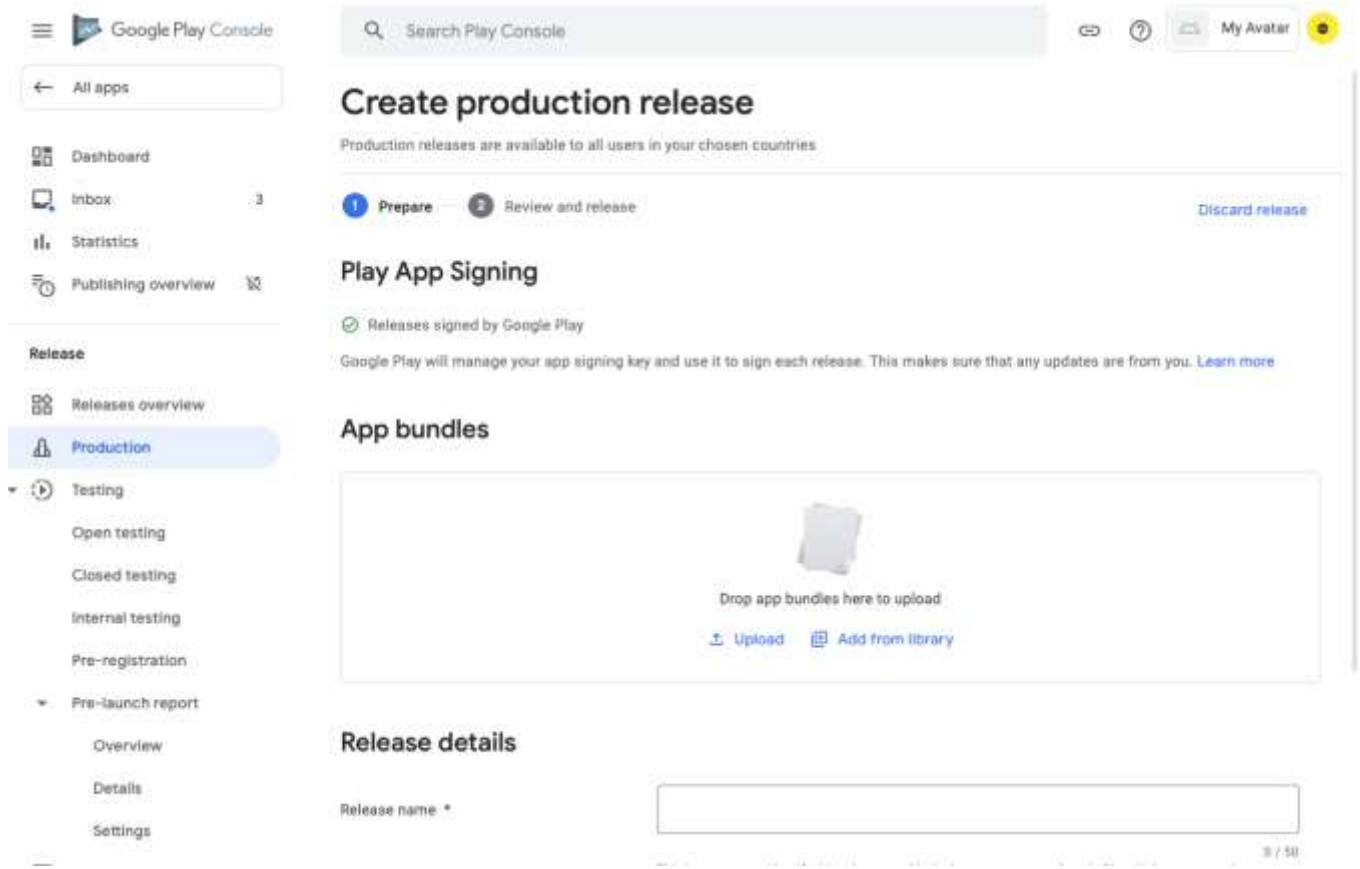
Copyright - Tout droit réservé - OFPPT

Déploiement sur Play Store

Une fois l'exécution de Fastlane terminée, vous pouvez trouver votre app bundle dans le répertoire de construction de l'application (l'emplacement par défaut est **app/build/outputs/bundle/release**).

Vous disposez maintenant d'un fichier de groupe d'applications prêt à être téléchargé sur la console Google Play.

Bien que vous automatisez les versions suivantes avec fastlane, vous devez télécharger **cette première version manuellement.**



PARTIE 3

Copyright - Tout droit réservé - OFPPT