Elaborer une application Android sécurisée

M214

Support du cours développement mobile

ETTAHERI Nizar
ettaheri.nizar@gmail.com

- **Introduction aux Web services**
- **Présentation de protocole REST**
- **Découverte du protocole SOAP**

- **Introduction aux Web services**

Les **services web** et les API sont des éléments essentiels aux applications web. N'importe quel site sur lequel vous naviguez est un service web, et il fournit une API pour communiquer avec.

Qu'est-ce qu'un service web ?

Un service web est un programme s'exécutant sur un serveur accessible depuis Internet et fournissant un service.

Par exemple, Google est un service web qui vous permet de rechercher des sites web. Une application météo communique avec un service web qui fournit la météo. Un réseau social est un service web qui permet de retrouver ses amis et de communiquer avec.

Le but d'un service web est donc de fournir un service à celui qui le demande. Et pour ce faire, il met à disposition une **API**.

Qu’est-ce qu’une API ?

Une API, ou *Application Programming Interface*, est une interface de communication. Il en existe différents types, mais celle qui nous intéresse est celle qui permet de communiquer avec les services web.

L’API correspond à l’ensemble des demandes que l’on peut faire à un service web. Ces demandes sont appelées des **requêtes**.

Par exemple, demander la météo actuelle est une requête. Faire une demande d’ami sur un réseau social est une requête. Ou encore, envoyer un message via une application de messagerie est une requête.

le protocole HTTP ?

HTTP signifie *HyperText Transfer Protocol.* C'est un protocole qui permet de **communiquer** avec un site Internet. Il va permettre de charger des **pages HTML**, des **styles CSS**, des **polices de caractères**, des **images**, etc. Mais ce n'est pas tout, le protocole HTTP nous permet aussi d'envoyer des formulaires et de récupérer et d'envoyer toutes sortes de données depuis ou vers un serveur implémentant ce protocole grâce à son API !

Tout ce que vous avez besoin de savoir c'est que plusieurs informations se trouvent dans une requête HTTP :

- **La méthode**. Il s'agit de l'action que l'on souhaite faire : récupérer une ressource, la supprimer, etc… Voici les méthodes HTTP les plus courantes :
  - **GET** : permet de **récupérer** des ressources, comme par exemple le temps actuel sur un service de météo ;
  - **POST** : permet de **créer** ou **modifier** une ressource, comme la création d'un nouvel utilisateur sur votre application ;
  - **PUT** : permet de **modifier** une ressource, comme le nom de l'utilisateur que vous venez de créer avec *POST* ;
  - **DELETE** : Permet de **supprimer** une ressource, comme un commentaire dans un fil de discussion.
- **L'URL**. C'est l'adresse sur le service web que vous souhaitez atteindre. Un peu comme un identifiant unique afin que le web service comprenne ce que vous voulez
- **Les données**. Lorsqu'on fait une requête pour enregistrer des données (par exemple un formulaire) il faut pouvoir envoyer ces données au service web.

Une fois votre requête envoyée et traitée par le service web, celui-ci va vous répondre avec, entre autres, les informations suivantes :

- **Les données**. Les données que vous avez demandées : une page HTML, etc…
- **Le code HTTP**. Il s'agit d'un code numérique qui vous indique comment s'est déroulée la requête. Voici les plus courants :
  - **200** : indique que tout s'est bien passé
  - **400** : indique que votre requête n'est pas conforme à ce qui est attendu
  - **401** : indique que vous devez être authentifié pour faire cette requête
  - **403** : indique que vous êtes bien authentifié mais que vous n'êtes pas autorisé à faire cette requête
  - **404** : indique que la ressource demandée n'existe pas
  - **500** : indique une erreur avec le service web

- **Présentation de protocole REST**

Une API, pour *Application Programming Interface*, permet d'**utiliser les ressources**, données ou fonctionnalités, d'une application web depuis une autre application.

Les API REST, en particulier, sont de plus en plus utilisées car elles fonctionnent de la même manière que le reste du web !

REST s'est aujourd'hui imposé comme un standard car c'est une architecture simple d'utilisation et plus facile à maintenir.

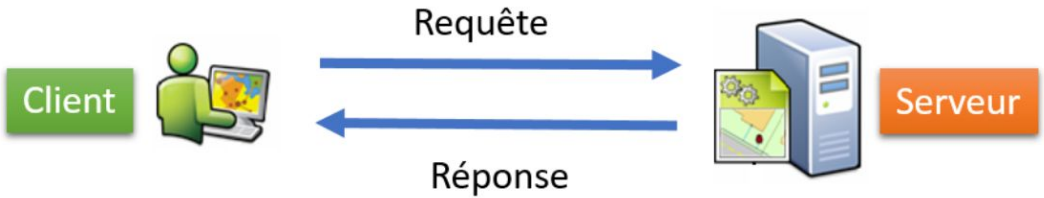
Le REST ne cesse de se démocratiser. Aujourd'hui, des entreprises comme Google, Twitter ou Facebook en font usage pour leur API publique.

Comment fonctionne une API ?

Distinguez client et serveur

Pour fonctionner, une API a besoin d'un **serveur** et d'un **client** :

- un **serveur** est un ordinateur puissant permettant d'exécuter une API. C'est le fournisseur du service, celui qui propose au client d'accéder à ses données ou services ;
- un **client** est un programme qui échange les données avec un serveur à travers l'API. C'est l'utilisateur de l'API. On parle plus couramment **d'architecture client-serveur**.



L'API REST, la toute-puissante !

Les **API REST** sont **les plus répandues** sur le web. Découvrez dans ce chapitre quelles sont les spécificités des API REST par rapport aux autres API et comment elles fonctionnent.

REST est un type d'architecture d'API qui signifie ***RE****presentational* ***S****tate* ***T****ransfer*. Il a été inventé par l'Américain **Roy Fielding** dans les années 2000, période charnière dans la reconnaissance du potentiel des API web, afin de mettre en place des méthodes simples pour accéder à des services web.

Ce type d'API permet à des logiciels qui sont **incompatibles**, qui ne parlent pas le même langage, de **communiquer facilement**. REST peut être considéré comme un langage commun à ces différents logiciels. Par exemple, une API REST peut être réalisée dans le langage Java ou .NET.

Des exemples d'utilisation de l'API REST dans votre quotidien !

Instagram

Vous souhaitez accéder à des comptes utilisateurs, à des photos, à des tags et bien plus encore depuis votre application ? L'API d'Instagram vous le permettra !

Gmail

L'API Gmail vous permet de lire et d'envoyer des messages, gérer les brouillons et les pièces jointes, rechercher des fils de discussion et des messages, utiliser des libellés, configurer les notifications push et gérer les paramètres Gmail.

Exemples d'URL

Imaginons que vous ayez un site qui permet de gérer des timbres en ligne. Le timbre, c'est votre ressource. Il est donc identifié par une URL, et on pourra utiliser des verbes HTTP pour exécuter des actions sur ce timbre :

- pour créer un timbre, pas besoin d'avoir l'URL de la ressource, vous pourrez par exemple utiliser : `POST http://monsiteweb.fr/stamps`. À ce moment-là, le serveur vous répondra sûrement avec l'identifiant du timbre créé. Par exemple 183 ;
- pour afficher le timbre 183, vous pourrez utiliser : `GET http://monsiteweb.fr/stamps/183`
- pour mettre à jour le timbre 183 : `PUT http://monsiteweb.fr/stamps/183`
- enfin, pour supprimer le timbre 183 : `DELETE http://monsiteweb.fr/stamps/183`.

Transformez votre API en API RESTful

Une API est considérée comme RESTful quand elle respecte le principe d'architecture REST qui s'applique aux services web. Les principales contraintes ont été proposées par Roy Fielding et sont les suivantes :

- **client-server** – un mode de communication avec séparation des rôles entre client et serveur ;
- **stateless server** – les requêtes doivent contenir toutes les informations nécessaires au traitement ;
- **cache** – la réponse du serveur doit être mise en mémoire côté client ;
- **uniform interface** – la méthode de communication entre client et serveur doit être sous la forme d'une URL.

Les clés de l'API REST

L'architecture REST s'articule autour de 4 principes clés :

- une ressource distribuée sur un serveur distant (mécanisme client-serveur) ;
- un identifiant de la ressource (les URL et protocole HTTP) ;
- des « verbes » HTTP permettant d’agir sur la ressource (GET, POST, PUT, DELETE non inclus dans l'URI) ;
- une représentation de la ressource (la réponse peut avoir plusieurs représentations possibles : HTML, JSON, XML).

Dans le chapitre précédent, vous avez vu qu'une des clés de l'API REST était la représentation de la ressource (JSON, XML,etc.). Allons voir ce qui se cache derrière les termes JSON et XML.

Découvrez JSON

JSON est l'acronyme de *JavaScript Object Notation*.

Il s'agit d'un format de fichier permettant de stocker les donnée

de manière organisée et lisible.

Intéressez-vous au XML

> XML signifie *e**X**tensible **M**arkup **L**anguage* : en français, c'est un langage de balisage extensible.

Le XML est un langage de balisage, il permet de décrire des informations pour facilement les communiquer entre plusieurs applications. Il est dit *extensible* car il permet de décrire d'autres langages qui *dérivent* du XML et qui auront leurs propres balises et règles.

Ce format de fichier est conçu pour **transmettre des informations** entre applications informatiques.

```
<NOM>ARCGIS</NOM>
<CREATEUR>Jack Dangermond</CREATEUR>
```

Qu’est-ce que l’authentification ?

Vous comprenez de mieux en mieux comment fonctionnent les API. Vous savez :

- qui sont le client et le serveur ;
- qu'ils communiquent grâce au protocole HTTP ;
- et qu'ils utilisent des formats spécifiques de données pour se comprendre.

Plus concrètement, quand vous vous authentifiez auprès d'un serveur, vous prouvez votre identité en lui donnant des informations que vous seul connaissez. Une fois que le serveur sait qui vous êtes, il peut vous faire confiance et vous donner accès aux ressources de votre compte.

C'est le même procédé pour les clients utilisant des API. **Si vous êtes authentifié sur l'API, alors vous pouvez l'utiliser !** Il existe différentes techniques d'authentification :

- authentification **basique** ;
- authentification par **clé** ;
- **authentification O**uverte (**OAuth**).

- **Authentifiez l'API basiquement**

  L'authentification **basique** est la plus simple des authentifications ! Elle nécessite un nom d'utilisateur et un mot de passe.

- **Authentifiez l'API avec une clé**

  Même si l'authentification basique est un système valide, l'identifiant utilisé pour accéder à l'API est le même que le compte utilisateur.

- **Authentifiez l'API avec OAuth2**

  un protocole de délégation d'autorisation, c'est la technique la plus utilisée sur le web actuellement. L’application en question a simplement besoin que vous l’autorisiez à un moment donné.

Quelques recommandations

Vous l'aurez compris, utiliser une API c'est bien, la connaître c'est mieux !

Voici quelques conseils :

- évitez d'utiliser une API publique non sécurisée ;
- préférez une API REST, voire RESTFUL !
- Utilisez une API privilégiant le format d'échange JSON.

Vous avez maintenant toutes les clés pour appréhender la conception d'une API

Définissez les ressources

l'URL GEThttps://myvgburger.com/commands peut renvoyer la liste des commandes en cours. Tandis que cette URL, GET https://myvgburger.com/commands/<id_commande>, renverra une ressource particulière, en l'occurrence une commande spécifique.

> La chaîne "commands/<id_command>" constitue le **endpoint** (point final) supporté par votre API. Pour faire simple, c'est la seule manière d'accéder à une fonctionnalité précise.

Votre API de gestion d'une bibliothèque pourrait ressembler à ceci :

Verbe HTTP	endpoint	Actions
GET	/commands	Lister les commandes
POST	/commands	Passer une commande
GET	/commands/123	Détails de la commande 123
PUT	/commands/123	Modifie la commande 123
DELETE	/commands/123	Annule la commande 123

Chaque ressource peut être accessible en HTML ou en JSON.

Quelques conseils

- Une API est pour un client. Elle doit proposer des fonctions simples et évolutives.
- L'API doit être sécurisée (basique, clé d'API, OAuth).
- Préférez les noms concrets pour décrire vos ressources, évitez les verbes !
- Utilisez des minuscules pour le nommage des endpoints.
- L'API doit être documentée et illustrée par des exemples.

- **Découverte du protocole SOAP**

SOAP n'est pas un style d'architecture, c'est un **protocole de communication** basé exclusivement sur **XML** pour permettre aux applications de s'échanger des informations via HTTP. C'est l'acronyme de "**S**imple **O**bject **A**ccess **P**rotocol".

Ce protocole permet ainsi l'accès aux services web et à l'interopérabilité des applications à travers le web.

Par rapport à REST, SOAP est **plus lent** et requiert **plus de débit Internet**, car les messages sont plus verbeux que ceux de REST.

- **Utilisation de OAUTH**

OAuth 2.0, successeur du protocole OAuth 1.0a, est un framework d’autorisation permettant à une application tierce d’accéder à un service web.

Largement utilisé dans le domaine du web avec notamment Facebook ou encore Google, OAuth est devenu incontournable.

En tant que développeur, nous pouvons être amenés à utiliser un serveur fournissant un accès via OAuth 2.0 ou à implémenter un serveur d’autorisation pour sécuriser une API en utilisant ce framework.

La naissance de OAuth

Formalisé dans la RFC 6749 en octobre 2012, OAuth 2.0 est né d'un besoin simple.

Avec la démocratisation des applications web et la prolifération des applications mobiles, différentes applications sont amenées à interagir entre elles. Ainsi, un site web A pourrait utiliser les données d'un réseau social connu afin d'inscrire un utilisateur.

Cette démarche permet à l'utilisateur de donner accès à ses informations personnelles déjà disponibles sur le réseau social au site web A.

Avant l'arrivée de OAuth, l'utilisateur devait faire confiance au site web A et lui donner ses identifiants.

Cependant, pour éviter de redemander les identifiants à l'utilisateur, les applications tierces avaient tendance à conserver le mot de passe de celui-ci en clair.

et hormis ces problèmes de sécurité, une question d'ordre pratique se pose.

En effet, si l'utilisateur a fourni ses identifiants à plusieurs applications tierces et décide de les changer, il doit le modifier aussi pour toutes ces applications.

Pour répondre à ces problématiques, OAuth 2.0 a été créé afin de permettre des interactions entre applications dans un contexte sécuritaire optimal. OAuth 2.0 a été conçu pour être utilisé avec le protocole HTTPS.

Pour ceux connaissant déjà OAuth 1.0a (standardisé en 2010), OAuth 2.0 se différencie en partie de son prédécesseur par le fait que l'aspect sécuritaire lié à l'intégrité et la confidentialité des données transmises pendant la demande d'autorisation est délégué au protocole TLS.

OAuth 2.0 doit donc toujours s’utiliser avec une connexion sécurisée (HTTPS) pour bien remplir son objectif.

Les rôles

Afin de bien comprendre les mécanismes en jeu lors d’une demande d’autorisation avec OAuth 2.0, nous devons d’abord définir quelques notions utilisées dans les spécifications de ce framework.

Le schéma *classique* d’un processus d’authentification et d’autorisation fait intervenir deux parties :

- un serveur en mesure d’authentifier et d’autoriser l’accès à une ressource ;
- un utilisateur qui fournit ses identifiants pour accéder à la ressource.

- La ressource désigne ici toute information appartenant à l'utilisateur et qui est exposée par le serveur (son identité, ses photos, ses messages, etc.).

Pour OAuth, le processus d'autorisation est un peu plus complexe et peut faire intervenir jusqu'à quatre acteurs.

Propriétaire de la ressource : *Resource owner*

Le propriétaire de la ressource est une entité (par exemple un utilisateur) en mesure de donner l'accès à une ressource protégée.

Client

Le client désigne l'application tierce qui demande l'accès à la ressource au nom de son propriétaire. Le terme client peut parfois induire en erreur car il peut désigner, ici, aussi bien une application mobile qu'un serveur web. Il permet juste d'identifier l'entité qui souhaite accéder à une ressource.

Serveur de ressource : *Resource server*

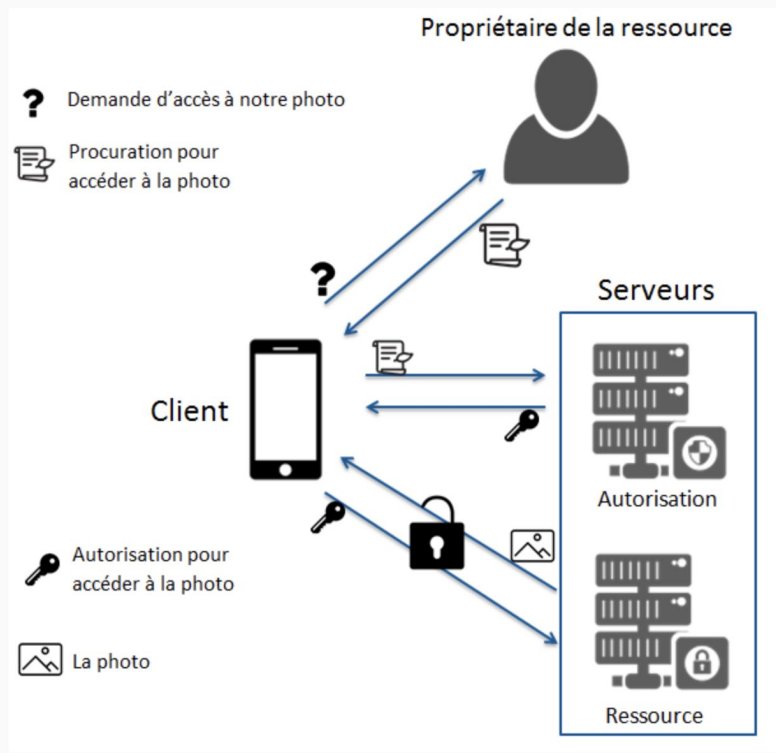
Le serveur de ressource désigne le serveur qui héberge les ressources protégées.

Serveur d'autorisation : *Authorization server*

Le serveur qui délivre le droit d'accès à la ressource protégée au client après avoir authentifier le propriétaire de la ressource.

Dans les faits, le serveur d'autorisation et le serveur de ressource sont souvent confondus.
Ainsi, un même serveur pourra jouer le rôle de serveur d'autorisation et héberger les ressources protégées.

A titre d'exemple, si un site web nous demande notre autorisation pour publier sur notre mur Facebook, nous jouons le rôle de propriétaire de la ressource, le site web est le client et le serveur de Facebook joue le double rôle de serveur d'autorisation et de serveur de ressource


Propriétaire de la ressource
Demande d'accès à notre photo
Procuration pour accéder à la photo
Serveurs
Client
Autorisation
Autorisation pour accéder à la photo
La photo
Ressource

La gestion des clients

Une demande d’autorisation avec OAuth est toujours initiée par un client. Pour tous les clients, il faudra donc les enregistrer auprès du serveur d’autorisation.

L’enregistrement nécessite au moins trois informations :

- l’identifiant du client ;
- le mot de passe ou la paire de clés (publique/privée) pour les clients confidentiels ;
- et une ou plusieurs URL de redirection.

Bien que les spécifications de OAuth 2.0 n’excluent pas l’utilisation de clients non-enregistrés, leur utilisation est au-delà des spécifications du protocole. Nous n’aborderons donc pas ce cas d’usage.

Les tokens

La demande d'accès à une ressource protégée via OAuth se traduit par la délivrance d'un *token* au client. Le *token* représente juste une chaîne de caractère unique permettant d'identifier le client et les différentes informations utiles durant le processus d'autorisation.

Le serveur d'autorisation est en mesure d'en fournir deux types.

Token d'accès : *Access token*

Le *token* d'accès permet au client d'accéder à la ressource protégée. Ce *token* a une durée de validité limitée et peut avoir une portée limitée.

Cette notion de portée permet d'accorder un accès limité au client. Ainsi, un utilisateur peut autoriser un client à accéder à ses ressources qu'en lecture seule.

Token de rafraîchissement : *Refresh token*

Le *token* de rafraîchissement permet au client d'obtenir un nouveau *token* d'accès une fois que celui-ci a expiré. Sa durée de validité est aussi limitée mais est beaucoup plus élevée que celle du *token* d'accès.

Son utilisation permet au client d'obtenir un nouveau *token* d'accès sans l'intervention du propriétaire de la ressource protégée.

En résumé, OAuth 2.0 formalise un ensemble de mécanismes permettant à une application tierce (client) d'accéder à une ressource protégée au nom de son propriétaire (*resource owner*) ou en son propre nom. Cette autorisation se traduit par la délivrance d'un *token* d'accès (et éventuellement d'un *token* de rafraîchissement) qui permet au client de dialoguer avec le serveur hébergeant les ressources protégées (serveur de ressource).

Les différents scénarios d'autorisation

Contrairement à son prédécesseur qui ne proposait qu’un seul scénario d’autorisation, le framework OAuth 2.0 a été spécifié avec quatre méthodes différentes pour obtenir une autorisation. En plus, le framework est extensible et permet donc de rajouter autant de méthodes que nous le souhaitons.

Selon la nature du client et du niveau d’accès souhaité, le scénario à utiliser n’est pas forcément le même. Le serveur d’autorisation peut ainsi autoriser l’utilisation d’un ou de plusieurs scénarios selon les besoins.

Autorisation avec un code : *Authorization Code Grant*

Ce processus d'autorisation principalement utilisé par les clients confidentiels permet d'obtenir un *token* d'accès (*Access token*) et un *token* de rafraîchissement (*Refresh token*). Il nécessite l'intervention du client, du propriétaire de la ressource protégée et du serveur d'autorisation. C'est d'ailleurs le mode d'utilisation à l'origine du protocole OAuth 1.0.

Les clients dits confidentiels sont les clients en mesure de conserver en toute confidentialité les identifiants qui leurs sont affectés.

Autorisation implicite : *Implicit Grant*

Ce processus est idéal pour les applications clientes dites *publiques*. Comme pour l'autorisation avec un code, il fait intervenir le client, le propriétaire de la ressource protégée et le serveur d'autorisation.

Les clients publics, à l'opposé des clients confidentiels, ne peuvent pas assurer la confidentialité de leurs identifiants.

Autorisation avec les identifiants du propriétaire de la ressource : *Resource Owner Password Credentials Grant*

Ce processus se démarque des deux premières par le fait qu’il ne nécessite pas de redirection du propriétaire de la ressource vers le serveur d’autorisation

Exemple d’utilisation

Un exemple courant pour ce type d’autorisation est l’utilisation de la fonctionnalité Facebook Login.

Considérons que nous sommes en train de développer une application mobile Windows 8 et pour s’authentifier, nous demandons aux utilisateurs de nous donner accès à leurs comptes Facebook.

- **Intégration de SSL**

Un **certificat SSL** est un fichier de données qui lie une clé cryptographique aux informations d'une organisation. Installé sur un serveur, le certificat active le cadenas et le protocole « https », afin d'assurer une connexion sécurisée entre le serveur web et le navigateur. Le SSL est généralement utilisé pour sécuriser les transactions bancaires, le transfert de données et les informations de connexions. Il est récemment devenu la norme pour sécuriser la navigation sur les sites de réseaux sociaux.

Les certificats SSL lient ensemble :

- Un nom de domaine, un nom de serveur et un nom d’hôte.
- L'identité de l'organisation (nom d'entreprise) et le lieu.

Remarque : A partir d'Août 2020, le cadenas et la barre d'adresse verte utilisés auparavant pour symboliser l'utilisation d'un certificat EV n'apparaîtront plus dans les navigateurs .

L'organisation doit installer le certificat SSL sur son serveur web afin d'initialiser des sessions sécurisées avec les navigateurs. Une fois la connexion sécurisée établie, l'ensemble du trafic entre le serveur et le navigateur sera sécurisé.

Une fois le certificat correctement installé sur le serveur, le protocole HTTP devient HTTPs, le ‘S’ signifiant ‘sécurisé’.

Les certificats SSL utilise ce qu’on appelle **la cryptographie à clé publique.**

Ce type de cryptographie exploite la puissance de deux clés qui sont de longues chaînes de nombres générés de manière aléatoire. L'une est appelée clé privée et l'autre clé publique, Une clé publique est connue de votre serveur et disponible dans le domaine public. Elle peut être utilisée pour chiffrer n'importe quel message. Si Alice envoie un message à Bob, elle le verrouillera avec la clé publique de Bob, mais la seule façon de le décrypter est de le déverrouiller avec la clé privée de Bob. Bob est le seul propriétaire de sa clé privée et il est par conséquent le seul à pouvoir l'utiliser pour déverrouiller le message d'Alice. Si un pirate informatique intercepte le message avant que Bob ne le déverrouille, tout ce qu'il obtiendra est un code cryptographique qu'il ne pourra pas déchiffrer, même avec la puissance d'un ordinateur.

Pourquoi installer un certificat SSL ?

Les certificats SSL protègent vos informations confidentielles telles que les informations relatives aux cartes de crédit, les informations de connexion telles que le nom d'utilisateur, le mot de passe, etc. Ils permettent également :

- Sécuriser les données entre les serveurs
- Améliorer votre classement sur Google
- Renforcer la confiance des clients
- Améliorer les taux de conversion

Où puis-je obtenir des certificats SSL ?

Les certificats SSL doivent être émis par une Autorité de Certification de confiance. Les navigateurs, les systèmes d'exploitation et les appareils mobiles tiennent à jour la liste des certificats racine des AC de confiance.

Pour que le certificat SSL soit considéré fiable, le certificat racine doit être présent sur la machine de l'utilisateur final. Si le certificat n'est pas reconnu, le navigateur affichera des messages d'alerte indiquant qu'il ne faut pas lui faire confiance. Sur les sites de e-commerce, de tels messages d'alerte inquiètent les visiteurs. Les organisations risquent alors de perdre la confiance d’un grand nombre de consommateurs, et ainsi de voir leur chiffre d’affaires diminuer.

Dans cette partie, je vais expliquer comment ajouter des certificats à notre application Android lorsque nous avons un `.pfx` fichier, pas seulement le `.crt`, et bien sûr, il inclura une brève explication de ce que sont ces fichiers.

- **Que sont les fichiers `.pfx` et .`.crt`**
- **Comment ajouter des certificats à mon application.**

Ce tutoriel va être basé sur un projet utilisant `Retrofit` pour faire des appels à une API, mais vous pouvez toujours l'adapter à votre cas spécifique.

Parlons des certificats

🖨️ CAs

*Une **autorité de certification (CA)** est une entreprise ou une organisation qui agit pour valider l'identité d'entités (telles que des sites Web, des adresses e-mail, des entreprises ou des personnes physiques) et les lier à des clés cryptographiques par l'émission de documents électroniques appelés **certificats numériques***

Donc, fondamentalement, les autorités de certification nous donnent un moyen de **nous authentifier** en servant d'informations d'identification pour valider notre identité, **crypter** nos données pour une communication sécurisée sur les réseaux non sécurisés tels qu'Internet, et nous donner un moyen de nous assurer que rien n'a été modifié par un tiers partie en transit en raison de la signature du certificat.

En règle générale, un demandeur de certificat numérique génère une **paire de clés** composée d'une **clé privée** et d'une **clé publique** , ainsi qu'une **demande de signature de certificat (CSR)** . Un CSR est un fichier texte codé qui comprend la clé publique et d'autres informations qui seront incluses dans le certificat (par exemple, nom de domaine, organisation, adresse e-mail, etc.). La génération de paires de clés et de CSR est généralement effectuée sur le serveur ou le poste de travail.

Les fichiers `.crt`et sont encodés CSR.`.pfx`

Quelle est la différence ?

Fichiers CER (ou .CRT) : le fichier CER est utilisé pour stocker le certificat X.509. Normalement utilisé pour la certification SSL pour vérifier et identifier la sécurité des serveurs Web. Le fichier contient des informations sur le propriétaire du certificat et la clé publique.

Fichiers PFX : Personal Exchange Format, est un fichier PKCS12. Celui-ci contient une variété d'informations cryptographiques, telles que des certificats, des certificats d'autorité racine, des chaînes de certificats et des clés privées. Il est protégé par cryptographie avec des mots de passe pour garder les clés privées privées et préserver l'intégrité des certificats racine.

Fondamentalement, le PFX contient plus d'informations que le CRT.

Commençons !

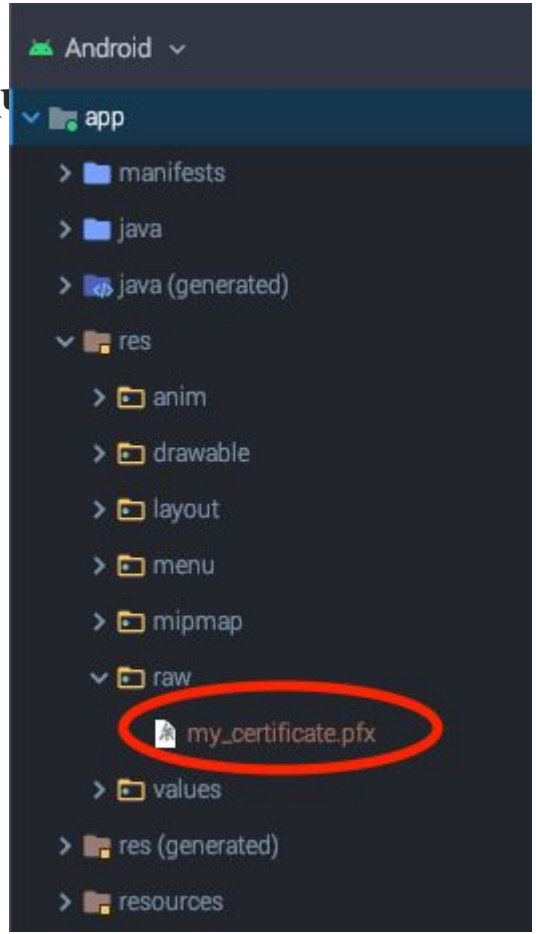
Afin d'ajouter nos certificats nous allons créer une méthode qui génère un OkHttpClient. Nous n'avons qu'à suivre ces étapes simples :

1. Ajoutez notre fichier dans notre projet en tant que ressource brute.
2. Créez une méthode qui renvoie un fichier `OkHttpClient`.
3. Créez un KeyStore contenant nos autorités de certification de confiance.
4. Créez une KeyManagerFactory .
5. Créez un contexte SSL contenant nos autorités de certification de confiance.
6. Ajoutez notre usine de sockets à notre constructeur.
7. Utilisez le généré `OkHttpClient`comme d'habitude.

1. **Ajoutez notre fichier dans notre projet en tant qu**

   (dans le dossier raw).

Nous pouvons simplement faire glisser et déposer le fichier dans le `raw folder`ou aller dans notre répertoire de projet et le créer à l'intérieur du `app/src/main/res` puis coller notre fichier.



2. Créez une méthode qui renvoie un fichier `OkHttpClient`.

Cette méthode est l'endroit où nous allons ajouter notre certificat à notre `OkHttpClient`afin que nous puissions l'utiliser pour effectuer des appels à une API. Dans cet exemple, je vais simplifier, mais vous pouvez toujours ajouter des paramètres personnalisés.

```
fun generateSecureOkHttpClient(context: Context): OkHttpClient {

    // Create a simple builder for our http client
    var httpClientBuilder = Builder()
            .readTimeout(60, TimeUnit.SECONDS)
            .connectTimeout(60, TimeUnit.SECONDS)

  // Here you may wanna add some headers or custom setting for
  // your builder
}
```

3. Créez un KeyStore contenant nos autorités de certification de confiance.

La classe KeyStore va nous aider à stocker nos certificats, mais le type d'instance est très important, cela va faire la différence entre l'utilisation d'un `.crt`fichier ou d' un fichier `.pfx`.

.CRT → le type par défaut va fonctionner pour vous.`KeyStore.getInstance(KeyStore.getDefaultType())`

.PFX → vous devez utiliser PKCS12, c'est un format spécial pour placer le certificat (inclut son "intermédiaire") avec la clé privée.

```
KeyStore.getInstance("PKCS12")
```

Le mot de passe est celui que vous devez utiliser lorsque vous voulez lire votre fichier.

Dans ce code, je le configure en tant que `String`, mais vous devez toujours le garder en sécurité, de sorte que vous feriez mieux d'utiliser un fichier avec lui ou en tant que Build Config Field .

```
// Get the certificates file
var caFileInputStream =
  context.resources.openRawResource(R.raw.my_certificate)

// Set our certificates in a Keystore
val keyStore = KeyStore.getInstance("PKCS12");
keyStore.load(caFileInputStream, "my file password".toCharArray())
```

4. Créez une KeyManagerFactory afin que nous puissions avoir des *keyManagers* avec l'algorithme de notre certificat.

Nous obtenons une instance d'une `X509`usine car c'est la norme de mon certificat de clé publique et c'est la plus utilisée.

```
// Create a KeyManagerFactory with our specific algorithm
// and public keys. Most of the cases is gonna be "X509"
val keyManagerFactory = KeyManagerFactory.getInstance("X509")
keyManagerFactory.init(keyStore, "my file password".toCharArray())
```

5. Créez un contexte SSL contenant nos autorités de certification de confiance.

Dans ce cas, nous utilisons une instance d'un contexte TLS parce que notre serveur l'exige, et comme TLS est fondamentalement une version plus récente de SSL, nous pouvons l'utiliser par défaut.

```
// Create a SSL context with the key managers of
// the KeyManagerFactory
val sslContext = SSLContext.getInstance("TLS")
sslContext.init(keyManagerFactory.keyManagers, null, SecureRandom())
```

6. Enfin, ajoutez notre usine de sockets à notre constructeur.

Utilisez simplement le `socketFactory` de `sslContext` que nous avons créé à la dernière étape et définissez-le sur le constructeur.

```
//Finally set the sslSocketFactory to our builder and build it
return httpClientBuilder
        .sslSocketFactory(sslContext.socketFactory)
        .build()
```

7. Utilisez le généré `OkHttpClient`comme d'habitude.

Maintenant que nous avons ajouté nos autorités de certification de confiance à un OkHttpClient, nous pouvons continuer à l'utiliser avec une `Retrofit`instance comme d'habitude.

```
fun generateService(context: Context): Retrofit {
    return Retrofit.Builder()
            .baseUrl("https://www.myapi.com" + "/")
            .client(generateSecureOkHttpClient(context))
            .build()
}
```

Conclusion

L'ajout de nos autorités de certification de confiance dans notre application Android n'est pas compliqué et peut être très utile lorsque nous avons plusieurs versions ou types de construction avec différents environnements d'API et que certains d'entre eux sont sécurisés avec une autorité de certification.

N'oubliez pas de supprimer votre `Development`certificat lorsque vous créez une `Release`version de votre application ; vous ne voulez pas le donner . Je suggère également de supprimer le fichier de Git pour des raisons de sécurité.

Ce post a été fait pour essayer de simplifier l'explication de l'implémentation que vous allez probablement trouver sur internet.

Code source

https://gist.github.com/kevingamboa17/7da8d8be5497c90b513197b11997b1c0#file-servicegenerator-kt

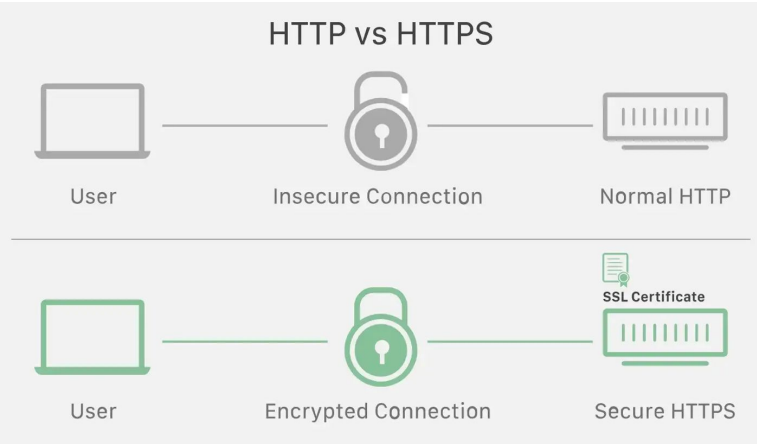
- **Sécurisation avec TLS**

Qu'est-ce que SSL ?

SSL *signifie* ***Secure Sockets Layer*** *. SSL est la technologie de sécurité standard pour établir un lien crypté entre un client et un serveur. Ce lien garantit que toutes les données transmises entre le serveur Web et le navigateur restent privées.*

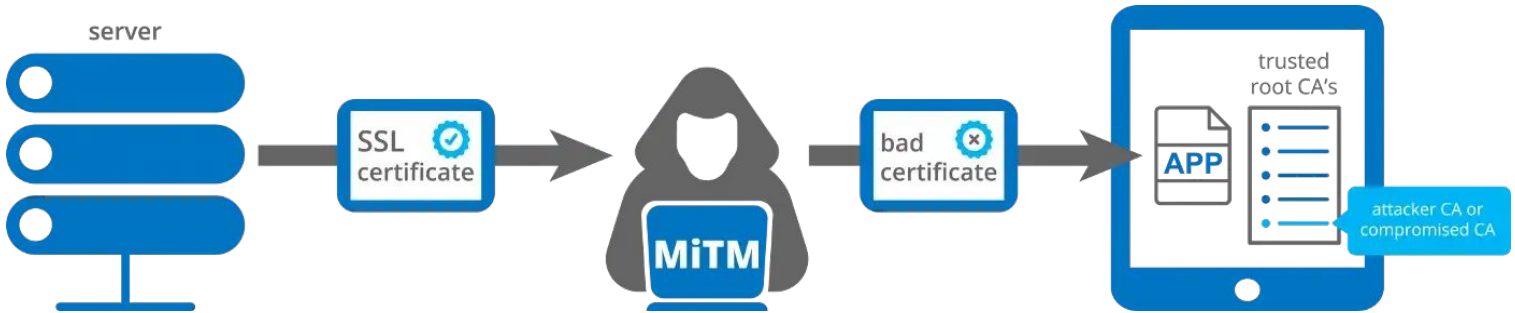
Ces deux systèmes peuvent être un serveur (notre serveur backend) et un client (notre application Android) ou un serveur et un autre serveur (notre serveur backend interagissant avec un autre serveur).

Il s'agit du protocole cryptographique le plus largement déployé pour fournir un canal de communication sécurisé.



Comment SSL assure-t-il la sécurité des données ?

Une attaque Man-in-the-Middle se produit lorsqu'un attaquant se place entre le serveur/hôte et le client, en se faisant passer pour l'un d'eux. En d'autres termes, lorsque le client se connecte au serveur, il traite en fait avec le pirate et vice versa. Ainsi, bien que le client "pense" avoir établi une connexion cryptée avec le serveur, mais en réalité les deux "parlent" à l'attaquant qui peut visualiser et modifier les données. Pour cette raison, tout le monde l'appelle une attaque "Man-in-the-Middle".



SSL crypte les données transmises afin qu'un tiers ou tout « Man-in-the-Middle » ne puisse pas « espionner » la transmission et voir les données transmises. Seuls le client et le serveur sécurisé sont capables de reconnaître et de comprendre les données. Cela signifie que quiconque essaie d'intercepter ces données ne verra qu'un mélange confus de caractères qu'il est presque impossible de déchiffrer.

Cette technique découle du concept de Certificat SSL et de l'infrastructure de l'Autorité de Certification. Il est basé sur l'utilisation de la clé privée, qui établit une connexion valide lorsqu'elle est associée au certificat correspondant. Cela lance un processus d'authentification appelé poignée de main entre deux appareils communicants pour s'assurer que les deux appareils sont bien ceux qu'ils prétendent être. SSL signe également numériquement les données afin d'assurer l'intégrité des données, en vérifiant que les données ne sont pas altérées avant d'atteindre leur destinataire.

Différence entre TLS et SSL

TLS (Transport Layer Security) est le successeur de SSL.

Il s'agit d'une version améliorée et sécurisée de **SSL** . Bien que le protocole SSL ait été obsolète avec la sortie de *TLS 1.0 en 1999* , il est encore courant de faire référence à ces technologies associées en tant que *« SSL » ou « SSL/TLS »*. Par conséquent, nous utilisons le terme ***SSL Pinning*** **.** La version la plus récente est **TLS 1.3** , définie dans **RFC 8446 (août 2018).**

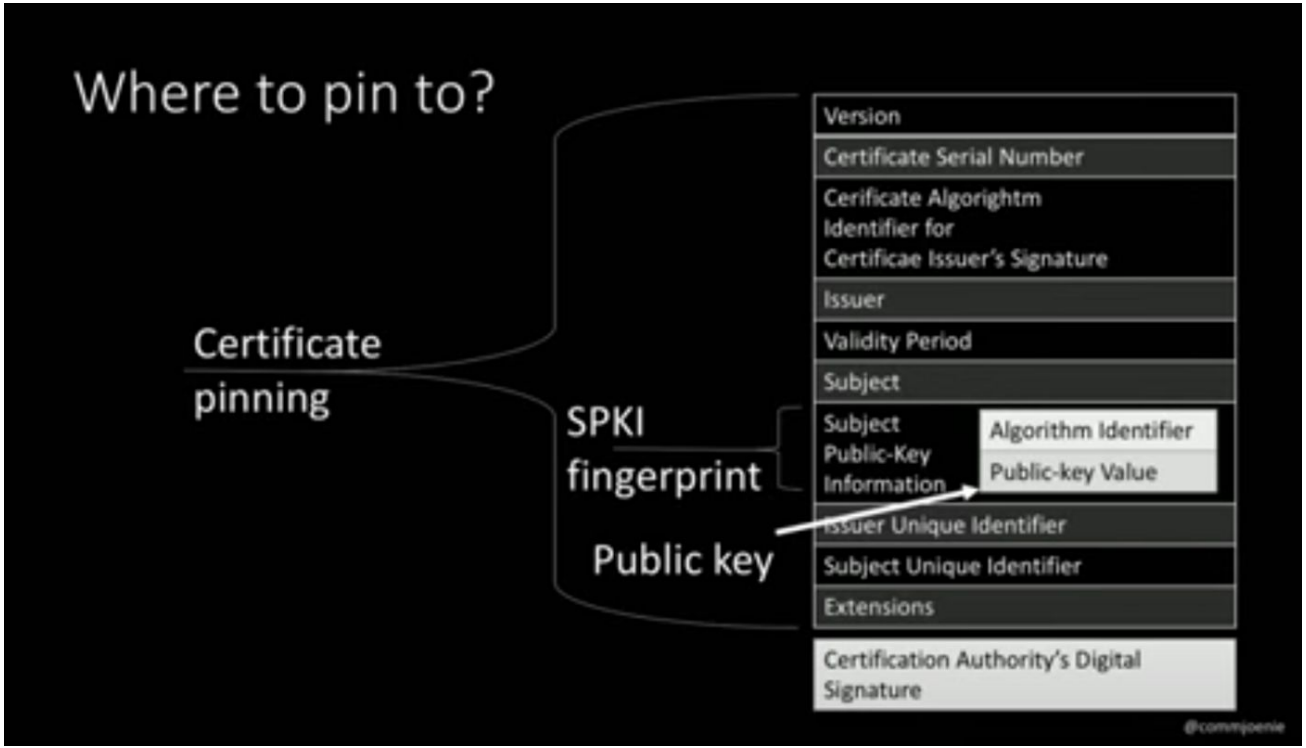
Qu'est-ce que le Pinning SSL et comment y parvenir ?

PInning SSL est un processus d'association d'un hôte avec son *certificat X509* ou sa *clé publique attendus*. Une fois qu'un certificat ou une clé publique est connu ou vu pour un hôte, le certificat ou la clé publique est associé ou *« épinglé »* à l'hôte. Si plusieurs certificats ou clés publiques sont acceptables, l'identité annoncée doit correspondre à l'un des éléments de l'ensemble de chaînes de certificats. Cela permet à l'application de faire confiance uniquement aux certificats valides ou prédéfinis ou aux clés publiques. Nous devons utiliser la technique d' **épinglage SSL** comme couche de sécurité supplémentaire pour le trafic des applications et pour valider l'identité de l'hôte distant. Si nous n'implémentons pas **SSL Pinning**, l'application fait confiance au certificat personnalisé et permet aux outils proxy d'intercepter le trafic.

Cela peut être réalisé de 3 manières : **épinglage de certificat, épinglage de clé publique et épinglage de hachage.**

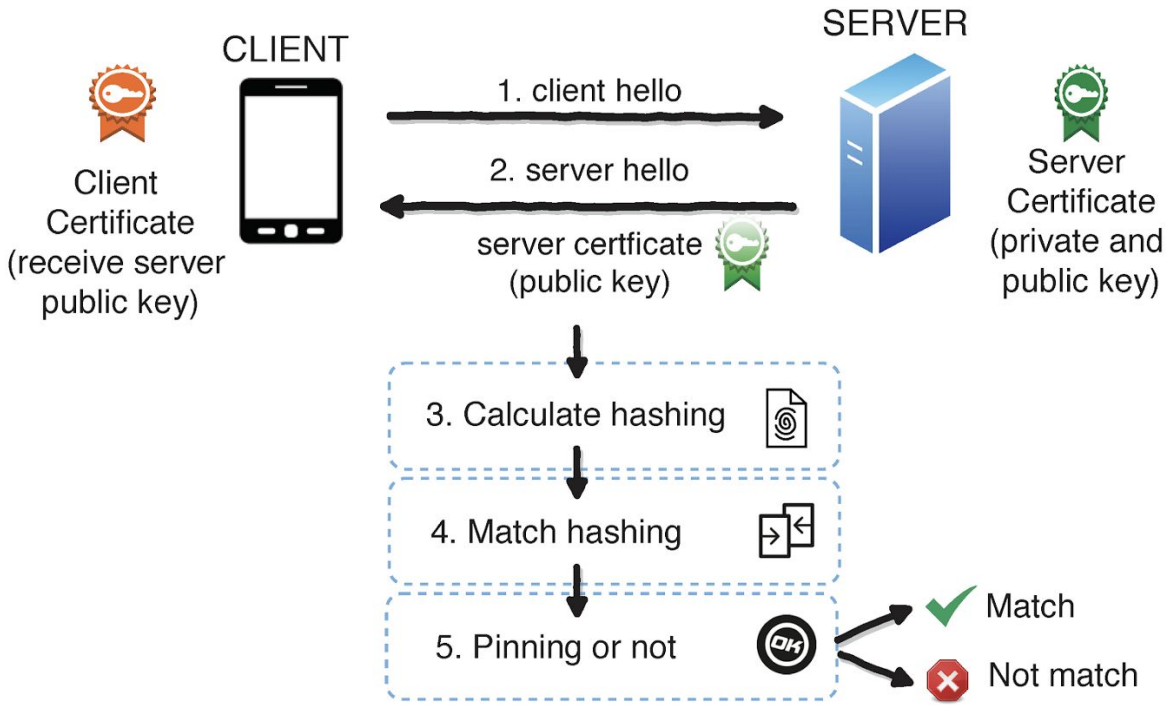
L'épinglage de certificat est le plus facile à réaliser. Nous pouvons stocker le certificat dans notre application et lorsque le certificat expire, nous mettrons à jour notre application avec le nouveau certificat. Lors de l'exécution, nous récupérons le certificat du serveur dans le rappel. Dans le rappel, nous comparons le certificat récupéré avec le certificat intégré dans notre application. Si cela correspond, nous pouvons faire confiance à l'hôte, sinon nous lancerons une erreur de certificat SSL.

Épinglage de clé publique



L'épinglage de clé publique est plus flexible à réaliser mais un peu plus délicat car il nécessite quelques étapes supplémentaires qui sont nécessaires pour extraire la clé publique d'un certificat. Dans cette approche, nous générons une paire de clés, mettons la clé privée dans notre serveur et la clé publique dans notre application. Et tout comme dans l'épinglage de certificat, nous vérifions la clé publique extraite avec sa copie intégrée de la clé publique. Si cela correspond, nous pouvons faire confiance à l'hôte, sinon nous lancerons une erreur de certificat SSL. En utilisant l'épinglage de clé publique, nous pouvons éviter les mises à jour fréquentes de l'application car la clé publique peut rester la même pendant de plus longues périodes.

Épinglage de hachage



Dans Hash Pinning, nous épinglons le hachage de la clé publique du certificat de notre serveur et le faisons correspondre avec le hachage de la clé publique du certificat reçu lors d'une requête réseau. Cette technique est plus complexe que d'autres, mais elle en vaut la chandelle. Après avoir obtenu le certificat, nous pouvons le hacher avec l'algorithme de hachage que nous préférons (assurez-vous seulement qu'il s'agit d'un algorithme sécurisé). Cela donne l'anonymat à un certificat ou à une clé publique. Personnellement, je préfère SHA-256 pour hacher ma clé. Après avoir calculé le hachage, je l'ai simplement encodé avec l' encodage Base64 , pour faciliter le stockage et la lecture.

Façons d'implémenter l'épinglage SSL dans Android

(a) Gestionnaire de confiance

TrustManager est chargé de décider si l'application doit accepter ou non les informations d'identification soumises par l'hôte. Cette interface provient du package *javax.net.ssl*

Les étapes à suivre sont -

1. Ajoutez le fichier de certificat à l'application.

*Nous pouvons l'ajouter dans le répertoire **res** ou sous le répertoire **assets***

2. Chargez KeyStore avec le fichier de certificat en tant qu'InputStream et créez un Keystore. Puisque je l'ai stocké dans le répertoire **res** , je vais le charger à partir de là.

```
val inputStream = resources.openRawResource(R. raw . my_cert )
val keyStoreType = KeyStore.getDefaultType()
val keyStore = KeyStore.getInstance(keyStoreType)
keyStore.load(inputStream, null)
```

3. Créez un TrustManager.

```
val tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm()
val trustManagerFactory = TrustManagerFactory.getInstance(tmfAlgorithm)
trustManagerFactory.init(keyStore)
```

4. Créez SSLContext qui utilise notre TrustManager et dites à l'URLConnection d'utiliser une SocketFactory à partir de notre SSLContext

```
val sslContext = SSLContext.getInstance("TLS")
sslContext.init(null, trustManagerFactory. trustManagers , null)
val url = URL("http://www.yourdomain.com/")
val urlConnection = url.openConnection() as HttpsURLConnection
urlConnection. sslSocketFactory = sslContext. socketFactory
```

Cette technique nécessite d'interagir directement avec l'API du framework. Cela signifie que la mise en œuvre est menée à un niveau assez bas. Par conséquent, nous devons être très prudents.

(b) OkHttp et CertificatePinner

L'épinglage de certificat à l'aide d'OkHttp est facile, car il suffit de créer une instance de CertificatePinner à l' aide d'un générateur dédié avec ses empreintes digitales correspondantes. Les empreintes digitales doivent être codées en dur dans l'application ou nous pouvons injecter de telles clés pendant le processus de construction, en utilisant la méthode buildConfigField. Ensuite, nous devons créer une instance OkHttpClient avec CertificatePinner.

Les étapes à suivre sont -

```
val certificatePinner = CertificatePinner.Builder()     .add
    (
        "www.votredomaine.com", "sha256/ZCOF65ADBWPDK8P2V7+ mqodtvbsTRR     /
        D74FCU+CEEA="
    )     . construire()
```

(b) OkHttp et CertificatePinner

L'épinglage de certificat à l'aide d'OkHttp est facile, car il suffit de créer une instance de CertificatePinner à l' aide d'un générateur dédié avec ses empreintes digitales correspondantes. Les empreintes digitales doivent être codées en dur dans l'application ou nous pouvons injecter de telles clés pendant le processus de construction, en utilisant la méthode buildConfigField. Ensuite, nous devons créer une instance OkHttpClient avec CertificatePinner.

Les étapes à suivre sont -

```
val certificatePinner = CertificatePinner.Builder()     .add
    (
        "www.votredomaine.com", "sha256/ZCOF65ADBWPDK8P2V7+ mqodtvbsTRR     /
        D74FCU+CEEA="
    )     . construire()
```

c) Épinglage avec Retrofit

Avec Retrofit construit sur OkHttp, le configurer pour l'épinglage est aussi simple que de configurer un OkHttpClient comme indiqué ci-dessus et de le fournir à votre *Retrofit.Builder()* .

```
val retrofit = Retrofit.Builder()
   .baseUrl("https://votredomaine.com")
   .addConverterFactory(GsonConverterFactory.create())
   .client(okHttpClient) //créé au-dessus
   de .build()
```

(d) Configuration de la sécurité du réseau

Il est disponible depuis Android 7.0 (niveau API 24 ou supérieur) et constitue le moyen privilégié d'implémenter l'épinglage. Cela nous permet de personnaliser nos paramètres de sécurité réseau dans un fichier de configuration sûr et déclaratif sans modifier le code de l'application. Avec **NSC** , nous pouvons déclarer des méthodes de communication, y compris Certificate Pinning, en utilisant *des fichiers XML* . Pour activer la configuration, nous devons lier un fichier de configuration au fichier *AndroidManifest.xml* .

Les étapes à suivre sont -

1. Créez un fichier de configuration de sécurité réseau sous

   res/xml/network_security_config.xml

2. Configurez le fichier de configuration et ajoutez des empreintes digitales.

```
<?xml version="1.0" encoding="utf-8"?>
<network-security-config>
   <domain-config cleartextTrafficPermitted="false">
       <domain includeSubdomains="true">votredomaine.com</domain>
       < pin-set>
           <pin
digest="SHA-256">9hdyeJFIEmx2Y01oXXXXXXXXXXmmSFZhBXXXXXXXXXX=</pin>
           <pin
digest="SHA-256">9Pacxtmctlq2Y73orFOOXXXXXXXXXXZhBXXXXXXXXXX=</pin>
       </pin-set>
   </domain-config>
</ configuration-sécurité-réseau>
```

3. Ajoutez l' attribut ***android:networkSecurityConfig*** à la balise d'application.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
   xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.packagename">
   <application
        android:networkSecurityConfig="@xml/network_security_config">
        …
   </application>
</manifest>
```

Conclusion

Sans épinglage de certificat, une application accepte généralement tout certificat correspondant au nom d'hôte demandé et émis par une autorité de certification (autorité de certification) de confiance locale. Comme il y a généralement plus de 100 autorités de certification dans le magasin de confiance local, il y a de fortes chances que l'une d'entre elles soit attaquée avec succès. Il est donc logique de limiter le certificat que vous acceptez à un certificat spécifique. Par conséquent, l'épinglage SSL aide à créer des applications mobiles sécurisées, mais il ne sécurisera pas les connexions si l'hôte épinglé est compromis. Il protège principalement le client, mais il aide également à protéger les serveurs en rendant plus difficile pour les pirates d'espionner le trafic et de comprendre votre API et d'exploiter d'autres failles de sécurité.

Cependant, l'épinglage SSL peut être contourné de plusieurs manières, s'il n'est pas correctement implémenté ou configuré. Certaines des techniques sont :-

1. Utiliser des outils automatisés

2. Par rétro-ingénierie (modification du code Smali)

3. Crochet

Prévention du contournement d'épinglage SSL -

Le contournement de l'épinglage SSL peut être empêché à l'aide de l'authentification SSL bidirectionnelle. En utilisant cette technique, l'application agit en tant que client SSL et envoie son certificat au serveur SSL pour validation après que le serveur SSL se soit validé auprès du client SSL.

La plupart du temps, la mise en œuvre du SSL bidirectionnel est complexe, donc si nous pouvons empêcher la modification ou l'ingénierie inverse de l'application Android, cela éviterait essentiellement le contournement SSL Pinning en utilisant l'ingénierie inverse ou la méthode Hooking ou d'autres outils automatisés.

- **Protection avec Proguard**

Sécurisez vos codes en activant Proguard dans l'application Android

De nos jours, il est assez facile de faire de l'ingénierie inverse et des applications Android. Un fichier .apk peut être décompilé facilement dans son code d'origine en utilisant certains outils, par exemple dex2jar, javadecompilers . Un développeur Android de niveau débutant ou un pirate informatique ou certains méchants peuvent réutiliser votre code en décompilant votre fichier .apk. Il est donc nécessaire d'optimiser votre code. N'est-ce pas ?

N'ayez pas peur, votre solution est ici. Pour empêcher quelqu'un de lire votre code, vous pouvez utiliser un outil qui rendra vos codes plus difficiles à lire. Et l'outil est **Proguard** .

Fonctions de Proguard

La principale fonction de proguard: **Obfuscation** . Il a également deux autres fonctions importantes : **Shrinking** et **Optimization** .

Obfuscation : L'obfuscation est un processus qui complique la vie d'un attaquant potentiel pour lire votre code. Il renomme les classes, champs et méthodes restants en utilisant des noms courts sans signification et ***rend votre APK difficile à désosser, ce qui est particulièrement utile lorsque votre application utilise des fonctionnalités sensibles à la sécurité, telles que la vérification sous licence.***

Shrinking : détecte et supprime les codes et les ressources inutilisés et — ***rend l'APK aussi petit que possible.***

Remarque : Proguard n'empêche pas la rétro-ingénierie. Mais cela rend les choses plus difficiles, "Le code obscurci rend votre APK difficile à désosser" selon la documentation des développeurs Android.

Activer Proguard dans Android Studio

Activer Proguard dans Android Studio est vraiment facile. Parce que **Proguard est ajouté au studio Android en tant qu'outil par défaut, mais par défaut, il reste désactivé** . Il vous suffit de rendre minifyEnabled true à partir du fichier build.gradle dans votre projet et proguard commencera à fonctionner. Vous trouverez ci-dessous le processus d'activation de ProGuard par défaut dans Android Studio.

1. Accédez au *fichier build.gradle(Module : app)* de app. Il s'agit de la valeur par défaut build.gradle(Module:app) où **minifyEnabled false** .

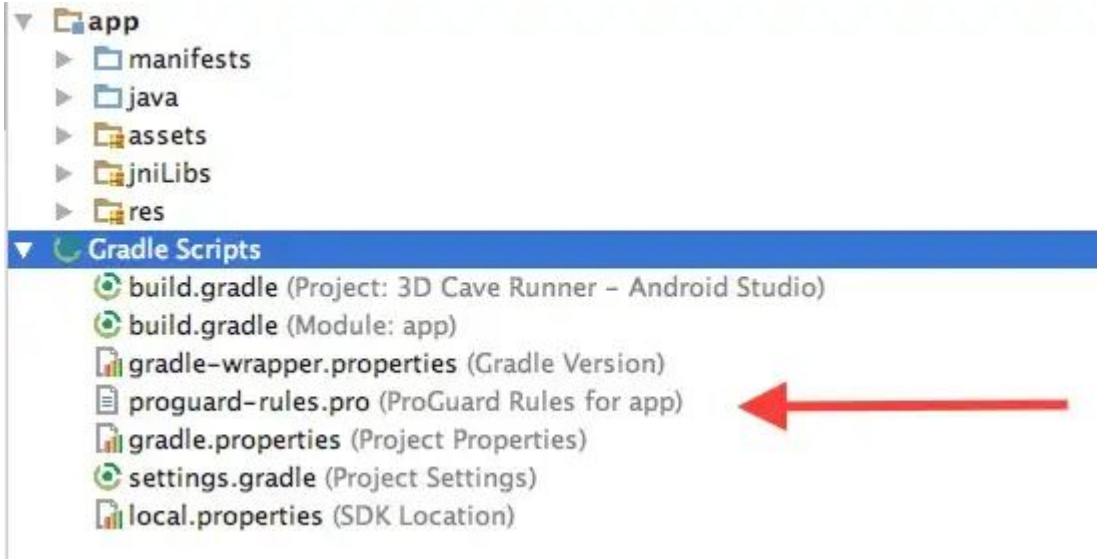
```
1
2  buildTypes {
3      release {
4          minifyEnabled false
5          proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
6      }
7  }
```

2. Faites juste **minifyEnabled true** pour activer proguard. Vous pouvez également activer shrinkResources et useProguard true pour réduire la taille de votre apk en supprimant les codes et les ressources inutilisés.

```
1
2   buildTypes {
3       debug {
4           minifyEnabled true
5           shrinkResources true
6           useProguard true
7           debuggable true
8           proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
9       }
10
11      release {
12          minifyEnabled true   // Obfuscate and minify codes
13          shrinkResources true // Remove unused resources
14          useProguard true     // Remove unused codes
15          debuggable false
16          proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
17      }
18  }
```

Ici, j'ai configuré mes fichiers à la fois pour la version et le mode de débogage. Vous pouvez désactiver Proguard pour le mode débogage. Dans ce cas, faites simplement minifyEnabled false, shrinkResources false, useProguard false dans le débogage.

Maintenant, si votre projet n'a pas de bibliothèque tierce, il fonctionnera fichier. Mais si vous ajoutez une bibliothèque tierce dans votre projet, vous devez spécifier les règles proguard pour cette bibliothèque spécifique dans `proguard-rules.pro`le fichier. Sinon, votre journal sera plein d'erreurs et ce sera très pathétique.



La bibliothèque tierce maximum fournit ses règles proguard dans son référentiel. Si vous n'avez pas réussi à les trouver, vous pouvez rechercher dans Googleou autre. Ici, je ***fournis des règles proguard pour certaines bibliothèques*** que nous utilisons normalement dans nos applications.

Certains fichiers texte générés avec Proguard Enabled

Lorsque vous créez un APK avec ProGuard activé, des fichiers texte de sortie supplémentaires sont créés **.** Les fichiers sont :

dump.txt — décrit la structure interne de tous les fichiers de classe dans l'APK.
mapping.txt — fournit une traduction entre les noms de classe, de méthode et de champ d'origine et masqués. Vous devez le télécharger à différents endroits, comme la console PlayStore, pour voir la trace de pile d'origine des plantages.
seeds.txt — répertorie les classes et les membres qui n'ont pas été obscurcis.
usage.tx t — répertorie le code qui a été supprimé de l'APK.

Ces fichiers sont enregistrés sur —`app/build/outputs/mapping/release/`

Après avoir téléchargé notre apk de version, nous pouvons voir le rapport de plantage contenant le nom de la classe, le nom de la méthode, etc. s'il y a un plantage. Mais si nous obscurcissons nos codes avec proguard, vous connaissez le nom de la classe, le nom de la méthode sera modifié. Il sera donc très difficile/impossible de lire ce crash. Alors, que devrions-nous faire ?

Il existe une solution facile. téléchargez simplement le fichier mapping.text dans Playstore avec votre apk et Google s'occupera de tout. Pour télécharger votre `mapping.text` *fichier, suivez le processus ci-dessous*

1. Connectez-vous à votre Play Console.
2. Sélectionnez votre application.
3. Dans le menu de gauche, cliquez sur **Android Vitals** > **Fichiers** de désobscurcissement .
4. À côté d'une version de votre application, cliquez sur **Télécharger** .
5. **Téléchargez** le fichier ProGuard `mapping.txt` pour la version de votre application.

Quelques points importants lors de l'utilisation de Proguard.

- Si vous souhaitez utiliser proguard dans votre application. Ensuite, essayez d'utiliser depuis le début.
- Lors de l'utilisation de proguard, la création de votre application prendra plus de temps. C'est pourquoi vous pouvez désactiver le (faux) proguard en mode débogage. Mais dans ce cas, vous devez l'activer pour vérifier l'ensemble de votre projet avant de générer l'apk publié.
- N'oubliez pas d'ajouter les règles proguard dans le fichier pour toute bibliothèque que vous avez incluse dans votre projet. `proguard-rules.pro`

- N'oubliez pas de télécharger le fichier dans Playstore. `mapping.text`
- Supposons que votre application a commencé à planter pour une classe spécifique après avoir activé proguard. Mais vous ne trouvez aucune raison à cela. Dans ce cas, vous devez exclure cette classe à masquer en spécifiant des règles dans le fichier proguard-rules.pro.
- Si vous souhaitez simplement obscurcir vos codes mais que vous ne souhaitez pas supprimer les classes inutilisées, ajoutez deux lignes dans `proguard-rules.pro` le fichier.

```
-dontshrink
-dontoptimize
```

- Pendant le processus de construction, proguard vérifie `AndroidManifest` et conserve toutes les classes d'activité. Ceci est nécessaire pour que votre application fonctionne. Vous ne devez pas obscurcir les classes qui étendent android.app.Activity.

Syntaxe d'écriture des règles proguard (Manuel Proguard)

Il existe différentes façons d' `keep` option que vous pouvez utiliser pour configurer Proguard :

-keep : conserve les classes et les membres de la classe.

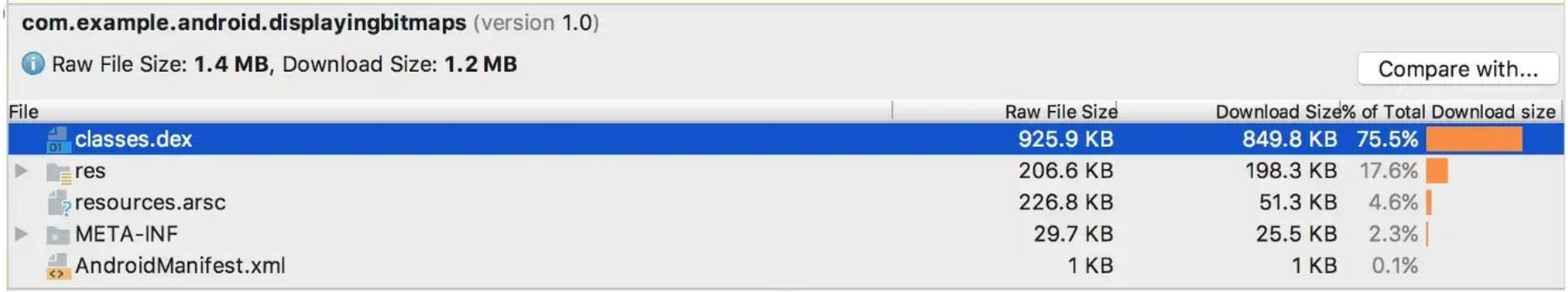
-keepclassmembers : ne conserve que les membres de la classe.

-keepclasseswithmembers : garde les classes et les membres de la classe, si les membres de la classe sont présents.

Analyseur d'APK et ProGuard

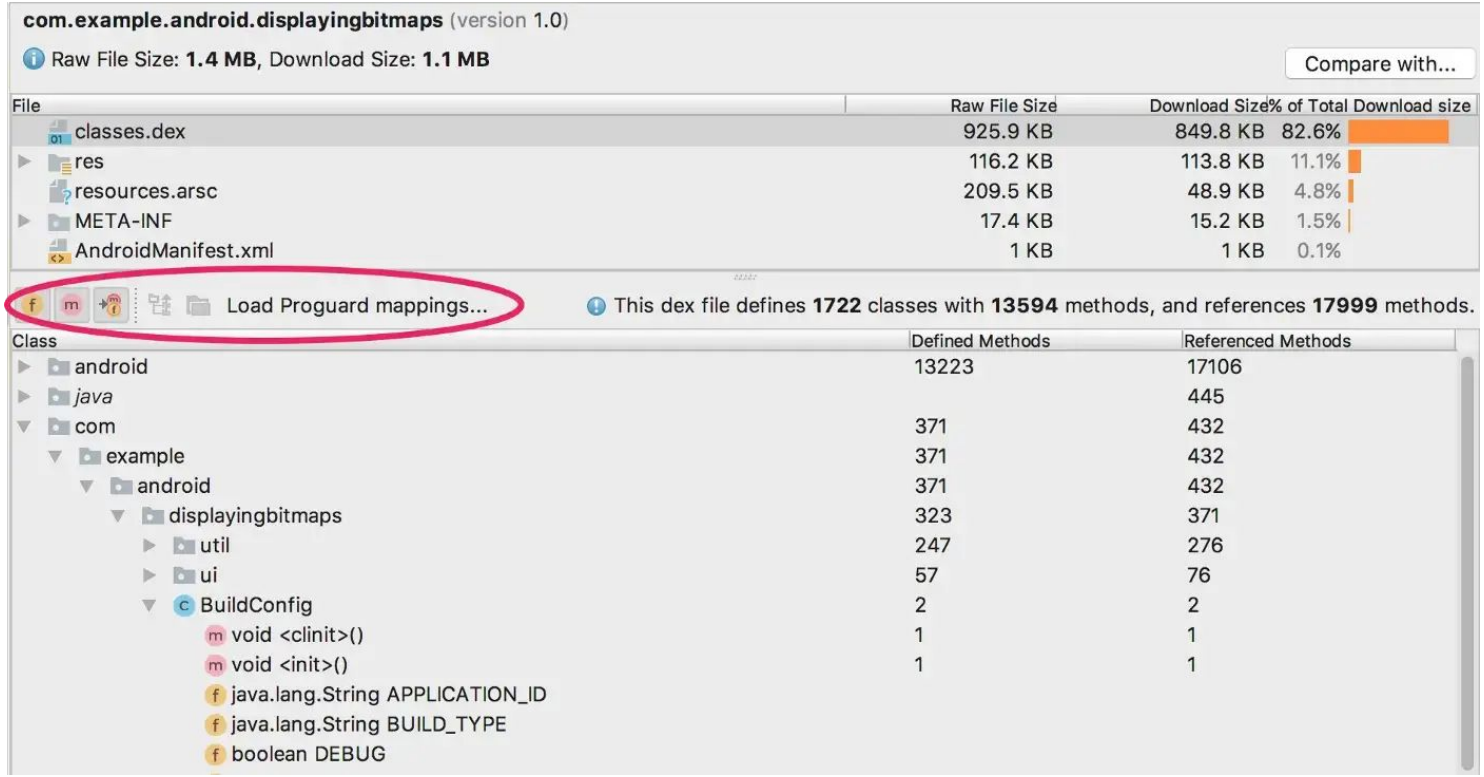
Android Studio comprend un analyseur d'APK qui fournit un aperçu immédiat de la composition de votre APK une fois le processus de construction terminé. L'utilisation de l'analyseur d'APK peut réduire le temps que vous passez à déboguer les problèmes avec les fichiers et les ressources DEX au sein de votre application, et vous aide à réduire la taille de votre APK. (De developer.android.com)

- APK Analyzer affiche également la taille du fichier brut et les valeurs de taille du fichier téléchargé pour chaque entité.



- L' analyseur d'APK dans Android Studio peut vous aider à voir les codes obscurcis ainsi que les classes qui ont été supprimées par ProGuard et à générer des règles de conservation pour eux.

- En chargeant des fichiers texte de mappage dans l'analyseur APK (à l'aide du bouton *"Charger les mappages Proguard… "*), vous obtenez des fonctionnalités supplémentaires dans l'arborescence dex. (Faites google pour cela).


com.example.android.displayingbitmaps (version 1.0)
Raw File Size: 1.4 MB, Download Size: 1.1 MB
Compare with...
File
Raw File Size
Download Size
% of Total Download size
classes.dex 925.9 KB 849.8 KB 82.6%
res 116.2 KB 113.8 KB 11.1%
resources.arsc 209.5 KB 48.9 KB 4.8%
META-INF 17.4 KB 15.2 KB 1.5%
AndroidManifest.xml 1 KB 1 KB 0.1%
Load Proguard mappings...
This dex file defines 1722 classes with 13594 methods, and references 17999 methods.
Class
Defined Methods
Referenced Methods
android 13223 17106
java 445
com 371 432
example 371 432
android 371 432
displayingbitmaps 323 371
util 247 276
ui 57 76
BuildConfig 2 2
void <clinit>() 1 1
void <init>() 1 1
java.lang.String APPLICATION_ID
java.lang.String BUILD_TYPE
boolean DEBUG

- **Manipulation de SOAP/Rest API**

Quand on aborde la gestion des APIs, le format standard de communication est JSON, appelé via des RESTFul. Il existe bien d'autres standards, comme SOAP, OData et XML mais JSON est devenu le standard par défaut. Une API JSON peut être facilement consommée par des projets d'applications Web, iOS, Windows ou encore Android.

Se connecter à une API JSON

La bonne nouvelle est que Google a inclus directement le support de JSON à Android.

Cela est rendu possible avec l'objet android.util.JsonReader. Google explique clairement le fonctionnement de son framework sur son site Web.

Voici 2 méthodes pour manipuler JSON :

- **Avec Array**: beginArray(), endArray(), hasNext()
- **Avec Objects**: beginObject(), endObject(), hasNext()

L'objectif est de charger un paquet JSON, d'examiner le contenu de l'objet puis de manipuler les données.

- **Retrofit**

Un client HTTP de type sécurisé pour Android et Java.

https://square.github.io/retrofit/

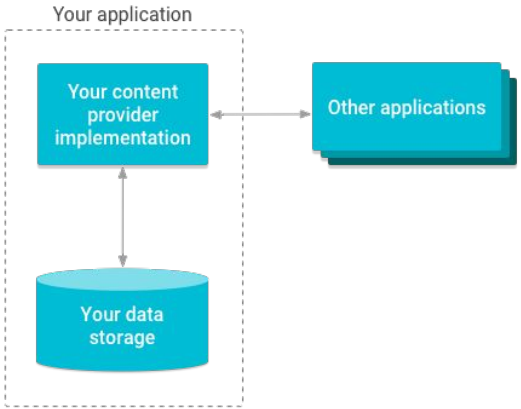
- **Volley**

Volley est une bibliothèque HTTP qui facilite et surtout accélère la mise en réseau des applications Android. Volley est disponible sur GitHub .

- **ContentProvider**

une application Android peut proposer, **publiquement** et de **manière sécurisée**, l'accès à certaines de ses données.

Cette exposition est possible sur Android grâce à un Content Provider, permettant de partager avec d'autres applications que la vôtre un contenu que vous aurez préalablement défini.



Afin **d'exposer** notre base de données SQLite, nous allons utiliser directement la classe ContentProvider…Une fois ce dernier configuré, les applications externes devront, dans le but d'accéder aux données exposées, **lui fournir une URI** que ce dernier **analysera** pour retourner les données appropriées.
insi, dans un premier temps, créez le package provider/ dans lequel vous créerez la classe **ItemContentProvider**.

Classe provider/ItemContentProvider.java :

https://gist.github.com/NizarETH/dbce78d420db695890cb3537d71f4799

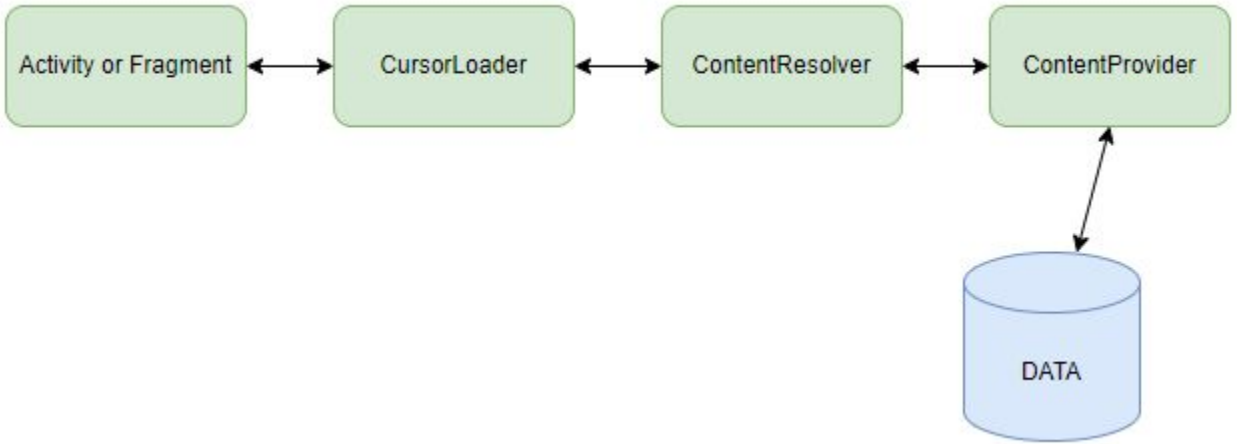
Comment fonctionne un fournisseur de contenu ?

Lorsque l'application veut accéder à ces données, c'est la même chose que nous voulons récupérer certaines données d'une API externe, le fournisseur de contenu, comme son nom l'indique, fournit une interface sur laquelle nous pourrions effectuer des opérations **CRUD** pour **C** reate, **Read** , **U** **pdate** et Supprimer les informations.

Alors… Comment pourrais-je accéder à ces données ?

En tant que composant Android, nous pouvons accéder à ce fournisseur via un autre composant d'application, par exemple une activité.

Ce serait le flux pour faire toute demande à partir de notre interface utilisateur :



1. Faites la demande depuis l'interface utilisateur.
2. Utilisez *CursorLoader* pour effectuer une requête asynchrone.
3. Le *ContentResolver* recevra cette demande et demandera au *ContentProvider* certaines données.
4. Le fournisseur de contenu répondra avec les données demandées.

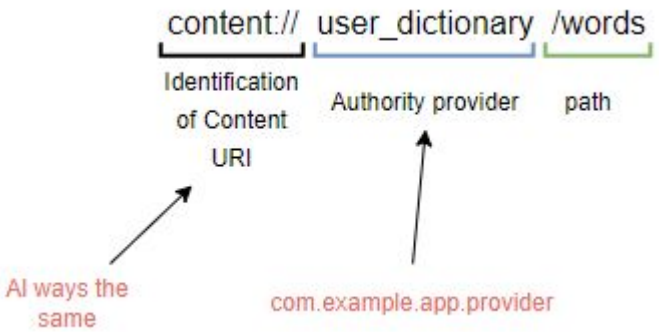
Le *ContentProvider* aura une instance de la base de données afin de pouvoir accéder directement aux données.

Avantages

- Vous pouvez **restreindre** l' **accès** à ces données uniquement pour qu'elles soient accessibles à partir de l'application en cours ou uniquement pour qu'elles soient accessibles en lecture ou en écriture.
- Facilite l' **abstraction** de cette couche.
- Le **ContentResolver** effectue toutes les requêtes de **manière asynchrone** afin que l'interface utilisateur ne soit pas bloquée tant que l'opération n'est pas terminée.
- **URI de contenu** faciles à partager .*

- **Qu'est-ce qu'un URI de contenu ?**
- Un URI de contenu est une identification de données, il pointe donc

  directement vers l'endroit où ces données sont stockées.



C'est le moyen d'identifier le sens des *mots du tableau*. Donc, si nous voulons partager ces informations, c'est aussi simple que de partager cette chaîne.

Une classe héritant de ContentProvider doit absolument implémenter les 6 méthodes suivantes :

- `onCreate()`: représente le point d'entrée du Content Provider. Vous pourrez donc y initialiser différentes variables qui vous serviront par la suite.
- `query()`: permet, à partir d'une URI renseignée, de récupérer les données (via un Cursor) depuis la destination de votre choix (dans notre cas, notre base de données SQLite).
- `getType()`: permet de retourner le type MIME associé à l'URI permettant d'identifier plus précisément le type des données qui seront retournées.
- `insert()`: permet, à partir d'une URI renseignée, d'**insérer** des données au format ContentValues dans la destination de notre choix (dans notre cas, notre base de données SQLite).
- `delete()`: permet, à partir d'une URI renseignée, de **supprimer** des données au format ContentValues de la destination de notre choix (dans notre cas, notre base de données SQLite).
- `update()`: permet, à partir d'une URI renseignée, de **mettre à jour** des données au format ContentValues dans la destination de notre choix (dans notre cas, notre base de données SQLite).

Pour le moment, ces méthodes sont vides... Mais ne vous inquiétez pas, nous les remplirons tout à l'heure

```
<?xml version="1.0" encoding="utf-8"?>

<manifest

   xmlns:android="http://schemas.android.com/apk/res/android"

   package="com.openclassrooms.savemytrip">

   <application>

      <provider

         android:name=".provider.ItemContentProvider"

         android:authorities="com.openclassrooms.savemytrip.provider"

         android:exported="true"/>

   </application>

</manifest>
```

- **Protection contre les entrées malveillantes**

L'objectif est de savoir s'il existe des fournisseurs de contenu implémentés dans cette application et si OUI, nous devons vérifier et exploiter s'ils sont vulnérables aux fuites de données.

Sujets concernés

- La collecte d'informations
- Attaquer les fournisseurs de contenu vulnérables
- Sécuriser les applications

La collecte d'informations

Comme tout autre pentest, commençons par la collecte d'informations. Nous supposons que nous avons le fichier APK avec nous. Alors, décompilez le fichier APK et vérifiez le fichier **AndroidManifest.xml** pour tous les fournisseurs de contenu enregistrés. Nous devrions également vérifier les fichiers smali pour tous les URI utilisés dans l'application.

Les fournisseurs de contenu sont généralement enregistrés dans le fichier AndroidManifest.xml au format suivant.

```
<provider
      android:name=".MyProvider"
      android:authorities="com.isi.contentprovider.MyProvider">
</provider>
```

Alors allons-y et examinons le fichier manifeste.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.isi.contentprovider"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="18" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name="com.isi.contentprovider.MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <provider android:name=".MyProvider"
                  android:authorities="com.isi.contentprovider.MyProvider"
                  android:exported="true">
        </provider>
    </application>

</manifest>
```

Content Provider

Nous avons un fournisseur de contenu enregistré dans le fichier AndroidManifest.xml et la bonne nouvelle est qu'il est exporté pour être accessible par toutes les autres applications.

Attaquer les fournisseurs de contenu vulnérables

C'est la partie la plus intéressante. Essayons maintenant d'interroger le fournisseur de contenu que nous avons trouvé. S'il renvoie des données, il est vulnérable. Cela peut être fait de plusieurs façons.

- Utiliser le shell adb
- Utiliser une application malveillante pour interroger
- Utilisation du cadre Mercury

Utiliser adb

Pour interroger le fournisseur de contenu à partir d'adb, l'application doit être installée sur l'appareil.

Obtenez un shell adb sur l'appareil et tapez la commande suivante pour interroger le fournisseur de contenu. Dans mon cas, je vais interroger l'URI que j'ai trouvé dans le fichier MyProvider.smali qui est extrait par l'outil APK.

Content –query –uri content://com.isi.contentprovider.MyProvider/udetails

Nous devrions maintenant voir tous les détails stockés dans la base de données de l'application, comme indiqué dans la figure ci-dessous.

```
shell@android:/ $ content query --uri content://com.isi.contentprovider.MyProvider/udetails
com.isi.contentprovider.MyProvider/udetails                                    <
Row: 0 id=2, name=harsha
Row: 1 id=3, name=infosec institute
Row: 2 id=1, name=srini
```

Utiliser une application malveillante pour interroger

Nous pouvons même écrire une application malveillante pour interroger les données de son fournisseur de contenu. Voici l'extrait de code pour interroger la boîte de réception à partir d'un appareil mobile.

```
Uri myURI = Uri.parse("content://sms/inbox");
String[] cols = new String[] { "_id", "address", "body" };
ContentResolver cr = getContentResolver();
Cursor c = cr.query(myURI, cols, null, null, null);
```

Utilisation du cadre Mercury

L'ensemble du processus peut être effectué à l'aide de Mercury Framework de manière encore plus efficace et simple.

Si nous essayons d'interroger le fournisseur de contenu dont la valeur android:exported est définie sur false, il lèvera une exception comme indiqué ci-dessous.

Remarque : La valeur par défaut de android:exported est true pour toutes les applications utilisant un niveau d'API inférieur à 17 .

- Limiter l'accès avec des autorisations personnalisées

Nous pouvons également imposer des restrictions basées sur les autorisations en définissant des autorisations personnalisées pour une activité. Ceci est utile si le développeur souhaite limiter l'accès aux composants de son application aux applications disposant d'autorisations.

Autres problèmes avec les fournisseurs de contenu

Injection SQL : Si les contrôles de sécurité ne sont pas correctement mis en œuvre, les fournisseurs de contenu peuvent mener à des attaques côté client comme l'injection SQL . Cela fonctionne de la même manière que les attaques par injection SQL traditionnelles.

Sécuriser les applications

- **Définition de la valeur de l'attribut android:exported sur false :**

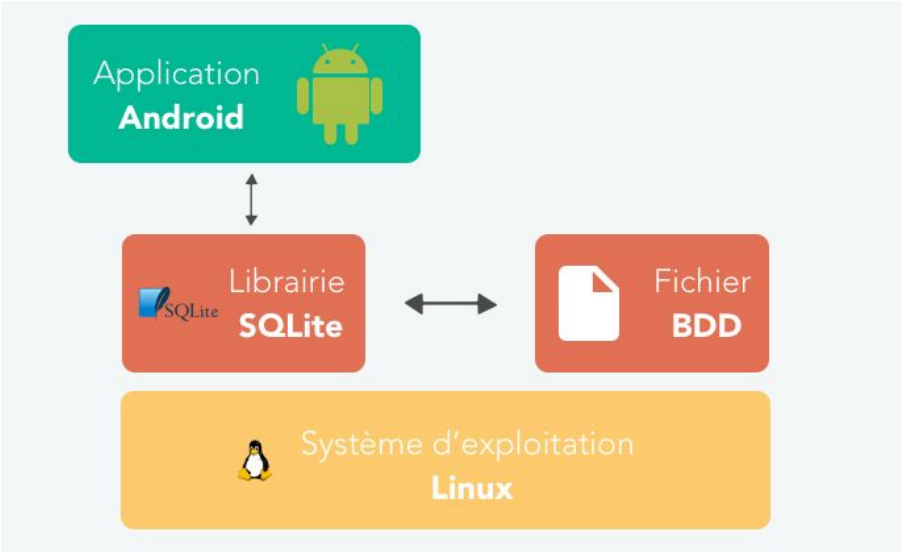
Dans le fichier AndroidManifest.xml de notre application, nous devons ajouter l'attribut suivant au fournisseur de contenu à sécuriser. Dans notre cas, com.isi.contentprovider.MyProvider est le fournisseur de contenu.

```
<provider
      android:name=".MyProvider"
      android:authorities="com.isi.contentprovider.MyProvider"
      android:exported="false">
</provider>
```

- **Utilisation de SqlLite**
- **Découverte de ROOM**
- **Utilisation des SharePreferences**
- **Manipulation des fichiers**

- **Utilisation de SqlLite/ROOM**

SQLite. Ce dernier est un **moteur de base de données relationnelle** entièrement manipulable grâce au langage SQL. Contrairement aux serveurs de base de données traditionnels comme MySQL, PostgreSQL ou encore Microsoft SQL Server, l'intégralité de la base de données est **stockée dans un fichier** (et non sur un serveur distant).



Sur Android, la manipulation de la base de données SQLite se faisait d'une manière... comment dire... assez compliquée !

Premièrement, car les requêtes SQL étaient **écrites "en dur"** dans des variables String statiques...

```
1 private static final String SQL_CREATE_ENTRIES =
2
3     "CREATE TABLE " + FeedEntry.TABLE_NAME + " (" +
4
5     FeedEntry._ID + " INTEGER PRIMARY KEY," +
6
7     FeedEntry.COLUMN_NAME_TITLE + " TEXT," +
8
9     FeedEntry.COLUMN_NAME_SUBTITLE + " TEXT)";
```

L'inconvénient de cette méthode c'est qu'il n'y a **aucune vérification possible** de la part de votre IDE (Android Studio) au **moment de la compilation**. Cela veut dire que, si une erreur s'est glissée, vous ne la verrez QUE lorsque votre application plantera.

Deuxièmement, car vous aviez besoin nativement de beaucoup de code pour transformer le résultat d'une requête en un objet Java. Du coup, des ORM ont commencé à apparaître pour pallier ces différents problèmes.

ORM

Alors en fait, un **Object-Relation mapping** (ou mapping objet-relationnel) est une technique de programmation qui permet de créer volontairement **l'illusion que l'on manipule une base de données orientée objet**, alors qu'en réalité nous manipulons une base de données relationnelle. En fait, l'ORM va permettre de créer une **couche d'abstraction** supplémentaire, afin que nous puissions manipuler une base de données relationnelle (comme SQLite !) avec des objets.

Dans cette partie, nous allons donc nous servir de Room, afin de faciliter le stockage de nos données structurées dans la base de données SQLite d'Android.

Pour cela, nous allons installer Room, en modifiant comme d'habitude notre fichier build.gradle.

Extrait de build.gradle :

```
dependencies {

    //ROOM

implementation "androidx.room:room-runtime:2.3.0"

annotationProcessor "androidx.room:room-compiler:2.3.0"


    //VIEW MODEL & LIVE DATA

implementation 'androidx.lifecycle:lifecycle-extensions:2.2.0'

}
```

Maintenant que Room (notre ORM) est installé, nous allons pouvoir commencer à représenter nos données sous forme de modèles (via des simples classes POJO).

Dans notre cas, j'ai choisi de modéliser cette fonctionnalité de la manière suivante :

- La classe `User`

```
@Entity(tableName = "User")
data class User (@PrimaryKey val id :Int,
                 val name :String,
                 val username :String,
                 val email :String,
                 val phone :String,
                 val website :String,
                 val lat : Double,
                 val lng : Double,

)
```

Maintenant que nos entités sont définies, il serait intéressant de pouvoir les **manipuler** à travers notre base de données SQLite. Par exemple, nous aimerions pouvoir **ajouter** une chose à faire dans notre base, la **mettre à jour** ou encore la **supprimer**... Bref, réaliser les différentes actions CRUD

c'est ce que nous allons faire immédiatement grâce à un patron de conception (ou Design Pattern en anglais) appelé Data Access Object (DAO), en français, un objet d'accès aux données.

```kotlin
@Dao
interface UserDao {

    @Insert
    fun insertUser(user: User)

    @Query("Select * from User")
    fun gelAllUsers(): List<User>

    @Update
    fun updateUser(user: User)

    @Delete
    fun deleteUser(user: User)

}
```

il nous reste tout de même à configurer une classe très importante, dont le rôle sera de **lier** toutes les classes/interfaces que nous avons précédemment créées ensemble, et surtout de **configurer** notre base de données

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

    abstract fun userDao() : UserDao

    companion object {
        private var INSTANCE: AppDatabase? = null

        fun getInstance(context: Context): AppDatabase? {
            if (INSTANCE == null) {
                synchronized(AppDatabase::class) {
                    INSTANCE = Room.databaseBuilder(context.applicationContext,
                        AppDatabase::class.java, name: "data.db").allowMainThreadQueries()
                        .build()
                }
            }
            return INSTANCE
        }

        fun destroyInstance() {
            INSTANCE = null
        }

    }
}
```

Explications : Nous créons ici une classe abstraite, héritant de `RoomDatabase` et étant définie par l'annotation `@Database`. Cette même annotation répertoriera les différentes tables (appelées "**entities**").

À l'intérieur de cette classe, nous avons déclaré nos deux **interfaces de DAO**. Puis, nous avons créé une méthode, `getInstance()` , permettant de créer un **singleton** de notre classe

- **SharedPreferneces**

La gestion des préférences

Description

L'API mise à disposition par Android pour sauvegarder des informations locales à l'application s'appelle SharedPreferences. C'est une couche d'abstraction qui vous facilite la vie : elle récupère les données que vous lui passez, et les stocke dans un fichier XML.

L'API offre la possibilité d'utiliser différents fichiers pour stocker les informations. C'est très pratique dans une application importante avec des données très variées.

Pour accéder à l'instance de **SharedPreferences**, il suffit d'appeler la méthode **getSharedPreferences()** sur l'activité courante, en lui fournissant le nom du fichier dans lequel sauvegarder les données, comme ceci :

```
SharedPreferences preferences = getSharedPreferences(SHARED_PREF_USER_INFO, MODE_PRIVATE);
```

N'oubliez pas de créer la constante qui représente le nom de fichier en haut de la classe :

```
private static final String SHARED_PREF_USER_INFO = "SHARED_PREF_USER_INFO";
```

Le paramètre **MODE_PRIVATE** permet de préciser que les données du fichier sont accessibles seulement par notre application. Encore heureux ! Eh oui, historiquement il existait d'autres modes qui permettaient à des applications tierces de venir lire et écrire dans votre fichier. C'est scandaleux, n'est-ce pas ? Je suis bien d'accord avec vous ! C'est pourquoi Google a décidé de rendre ces modes obsolètes, pour des raisons évidentes de sécurité. Donc partez du principe que vous utiliserez toujours le mode **MODE_PRIVATE**.

Écriture

Pour pouvoir modifier les informations stockées dans les **SharedPreferences**, il est obligatoire d'utiliser l'API **SharedPreferences.Editor**.

Tout d'abord, une information stockée est toujours associée à une clé, permettant ainsi de récupérer précisément une valeur souhaitée sans avoir à récupérer l'ensemble des données.

Ensuite, il est obligatoire de préciser le type de donnée stockée, en utilisant la méthode correspondante. Par exemple, pour stocker une chaîne de caractères, il faudra utiliser une méthode spécifique appelée **putString()**. Et pour stocker un entier, il faudra utiliser la méthode **putInt()**.

Enfin, pour valider la modification, il est nécessaire d'appeler la méthode **apply()**.

Par exemple, pour stocker le prénom de l'utilisateur, le code serait donc le suivant :

```
SharedPreferences preferences = getSharedPreferences(SHARED_PREF_USER_INFO, MODE_PRIVATE);

SharedPreferences.Editor editor = preferences.edit();

editor.putString(SHARED_PREF_USER_INFO_NAME, mUser.getFirstName());

    editor.apply();
```

vous préférerez probablement utiliser la version courte :

```
getSharedPreferences(SHARED_PREF_USER_INFO, MODE_PRIVATE)

  .edit()

  .putString(SHARED_PREF_USER_INFO_NAME, mUser.getFirstName())

      .apply();
```

1. La clé (constante) utilisée pour stocker le prénom de l'utilisateur est *SHARED_PREF_USER_INFO_NAME*. Vous pouvez la créer comme vous l'avez fait pour le nom du fichier des *SharedPreferences*.
2. Le prénom de l'utilisateur est récupéré du modèle.

Lecture

Comme pour l'écriture, la lecture d'une valeur requiert l'appel à une méthode spécifique suivant le type de donnée à récupérer. Par exemple, pour récupérer une chaîne de caractères, il faudra utiliser **getString()**, alors qu'un entier sera récupéré en utilisant **getInt()**. Par exemple, pour récupérer le prénom de l'utilisateur, le code suivant sera utilisé :

```
String firstName = getSharedPreferences(SHARED_PREF_USER_INFO,
MODE_PRIVATE).getString(SHARED_PREF_USER_INFO_NAME, null);
```

En résumé

- Vous pouvez sauvegarder localement des informations grâce à l'API SharedPreferences.
- Pour accéder à l'instance de SharedPreferences, appelez la méthode getSharedPreferences() sur l'activité courante.
- Pour modifier les informations stockées dans les SharedPreferences, vous devez utiliser l'API SharedPreferences.Editor.

En résumé

- Vous pouvez sauvegarder localement des informations grâce à l'API SharedPreferences.
- Pour accéder à l'instance de SharedPreferences, appelez la méthode getSharedPreferences() sur l'activité courante.
- Pour modifier les informations stockées dans les SharedPreferences, vous devez utiliser l'API SharedPreferences.Editor.

- **Manipulation des fichiers**

Le but de celle-ci est de permettre à l'utilisateur d'écrire du texte **dans un fichier** qui sera mémorisé dans l'espace de stockage externe ou interne de son téléphone, selon le choix qu'il fera via les boutons radios (appelés aussi "cases d'options") :

- Dans quel stockage souhaitez-vous écrire ?
  - **Externe**
    - Quel niveau de confidentialité souhaitez-vous ?
      - Public : Le fichier sera stocké sur l'espace de *stockage externe* en mode public. Il ne sera donc *pas supprimé* quand l'utilisateur désinstallera son application.
      - Privé : Le fichier sera stocké sur l'espace de *stockage externe* en mode privé. Il sera donc *supprimé* quand l'utilisateur désinstallera son application.
  - **Interne**
    - Souhaitez-vous stocker le fichier en cache ?
      - Oui : Le fichier sera stocké sur l'espace de *stockage interne* dans le répertoire dédié au cache. Il pourra donc être *supprimé à n'importe quel moment*.
      - Non : Le fichier sera stocké sur l'espace de stockage interne.

Nous commencerons dans ce chapitre par gérer la sauvegarde de ce fichier sur l'espace de **stockage externe** : le fichier pourra donc être créé soit en mode **privé**, soit en mode **public**.

Créez une classe utilitaire

Avant toute chose, nous allons créer une classe utilitaire que nous appellerons **StorageUtils.java** et que nous placerons dans le package **utils/** de notre application.

Classe utils/StorageUtils.java :

```
public class StorageUtils {}
```

Explications : Cette classe sera responsable de la *sauvegarde* et de la *récupération* des données inscrites au sein de notre fichier

Créez un chemin d'accès vers un fichier

Afin d'enregistrer le texte écrit par notre utilisateur, il faut dans un premier temps définir un fichier de destination. Créons ainsi la méthode `createOrGetFile()` dans notre classe **StorageUtils.java**.

Classe utils/StorageUtils.java :

```
public class StorageUtils {

  private static File createOrGetFile(File destination, String fileName, String folderName){

    File folder = new File(destination, folderName);

    return new File(folder, fileName);
  }
}
```

La classe File est un peu trompeuse, car on pourrait croire qu'il s'agit d'une représentation d'un fichier, mais ce n'est pas uniquement le cas ! En fait, elle représente plutôt **un chemin d'accès vers un fichier**, qui sera utilisé par la suite pour y enregistrer ou y récupérer un flux de données (caractères, octets, etc.).

Écrivez et lisez des données depuis un fichier

Maintenant que nous avons vu comment **créer** et **récupérer** un fichier à partir d'un chemin d'accès, il serait bien de pouvoir **lire** et **écrire** à l'intérieur. Pour cela, rajoutez les lignes suivantes :

Extrait de StorageUtils.java :

```
public class StorageUtils {

    // --------------------------------
    // READ & WRITE ON STORAGE
    // --------------------------------

    private static String readOnFile(Context context, File file){

        String result = null;
        if (file.exists()) {
            BufferedReader br;
            try {
                br = new BufferedReader(new FileReader(file));
                try {
```

Manipulez les espaces de stockage

Maintenant que nos méthodes d'écriture et de lecture généralistes sont prêtes, nous allons pouvoir manipuler notre espace de stockage **externe** et **interne** avec. Pour cela, ajoutez les méthodes suivantes à votre classe StorageUtils.java.

Extrait de StorageUtils.java :

```
public class StorageUtils {

   public static String getTextFromStorage(File rootDestination, Context context, String fileName, String folderName){


      File file = createOrGetFile(rootDestination, fileName, folderName);


      return readOnFile(context, file);

   }
```

Explications : Nous avons ajouté ici plusieurs méthodes. Les deux **dernières** permettent de vérifier si l'espace de stockage **externe** est bien *disponible* et si l'on peut *lire* (`isExternalStorageReadable`) ou *écrire* (`isExternalStorageWritable`) dessus.

Puis nous avons créé deux autres méthodes permettant d'écrire et de lire du texte dans un fichier **se trouvant dans un espace de stockage défini en paramètre**... en réutilisant les méthodes `writeOnFile` et `createOrGetFile` que nous avons créées précédemment

Demandez l'autorisation

Eh oui ! Comme vous vous en doutez, **sauvegarder** et **lire** un fichier **sur l'espace de stockage externe** du téléphone de vos utilisateurs nécessite des autorisations spéciales. Ainsi, nous devons avant toutes choses déclarer ces autorisations dans notre application.

```
<manifest

  xmlns:android="http://schemas.android.com/apk/res/android"

  package="com.openclassrooms.savemytrip">

  <!-- ENABLE PERMISSIONS ABOUT EXTERNAL STORAGE ACCESS -->

  <uses-permission

  android:name="android.permission.WRITE_EXTERNAL_STORAGE"

  tools:ignore="ScopedStorage" />

</manifest>
```

En résumé

- La class StorageUtils contient tous les outils nécessaires pour écrire et lire un fichier de façon simple.
- Il est obligatoire de demander la permission à l'utilisateur d'utiliser le stockage externe.
- En mode public, le fichier n'est pas supprimé quand l'utilisateur désinstalle l'application.
- En mode privé, le fichier est supprimé quand l'utilisateur désinstalle l'application.

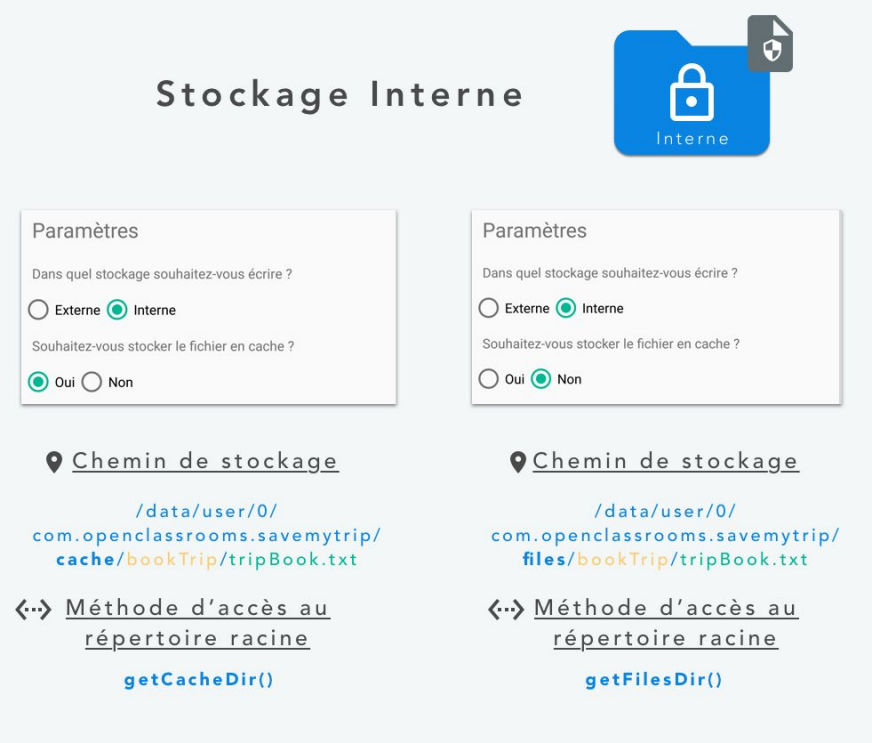
Maintenant que nous savons correctement enregistrer du contenu sur l'espace de stockage externe, nous allons permettre à nos utilisateurs de sauvegarder leur texte sur l'espace de **stockage interne** de leur téléphone, afin de le protéger encore plus..

en **changeant uniquement** le paramètre correspondant au **répertoire racine** pour correspondre à celui de l'espace de stockage **interne** :

- `getFilesDir()` : Cette méthode retournera le chemin d'accès (File) vers la racine de l'espace de stockage **interne** pour votre application.
- `getCacheDir()` : Cette méthode retournera le chemin d'accès (File) vers la racine du **cache** de l'espace de stockage **interne** de votre application.

Lancez maintenant votre application et testez l'enregistrement de texte dans l'**espace de stockage interne.**

Attention, **vous ne pourrez pas voir** les deux fichiers créés, tout simplement car vous n'y avez pas accès en tant qu'utilisateur (sauf si votre téléphone est rooté). C'est tout l'intérêt de l'espace de stockage **interne** !


Stockage Interne
Interne
Paramètres
Dans quel stockage souhaitez-vous écrire ?
Externe
Interne
Souhaitez-vous stocker le fichier en cache ?
Oui
Non
Chemin de stockage
/data/user/0/
com.openclassrooms.savemytrip/
cache/bookTrip/tripBook.txt
Méthode d'accès au répertoire racine
getCacheDir()
Paramètres
Dans quel stockage souhaitez-vous écrire ?
Externe
Interne
Souhaitez-vous stocker le fichier en cache ?
Oui
Non
Chemin de stockage
/data/user/0/
com.openclassrooms.savemytrip/
files/bookTrip/tripBook.txt
Méthode d'accès au répertoire racine
getFilesDir()

En résumé

- Il n’est pas obligatoire de demander à l’utilisateur la permission d’utiliser le stockage interne.
- On peut stocker soit dans un cache soit directement dans le stockage interne.
- On réutilise la classe StorageUtils qui est générique pour le stockage interne et externe.

- **Firebase**

Qu'est-ce que Firebase ?

Firebase est une plateforme mobile créée en 2011

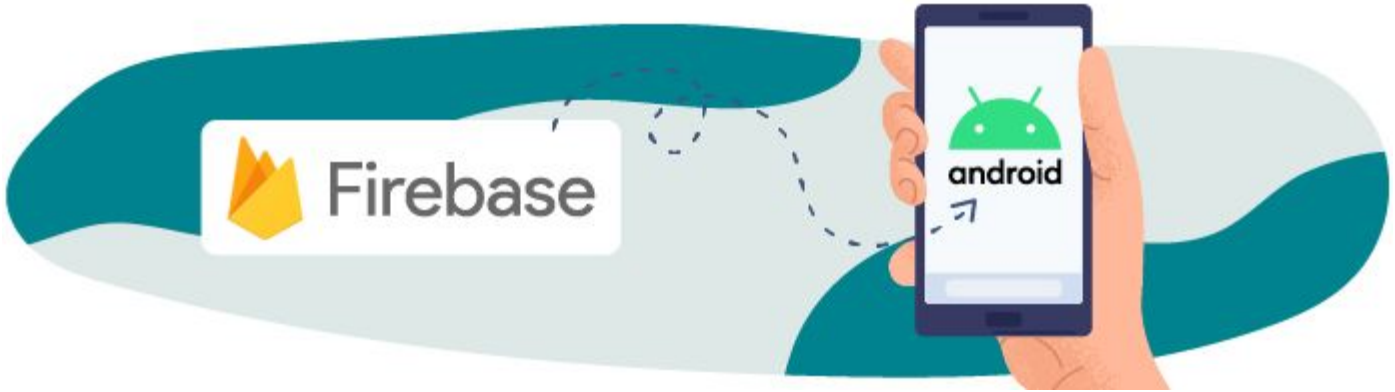
L'objectif premier de Firebase est de vous libérer **de la complexité de création et de la maintenance** d'une architecture serveur, tout en vous garantissant une scalabilité à toute épreuve (plusieurs milliards d'utilisateurs) et une simplicité dans l'utilisation.

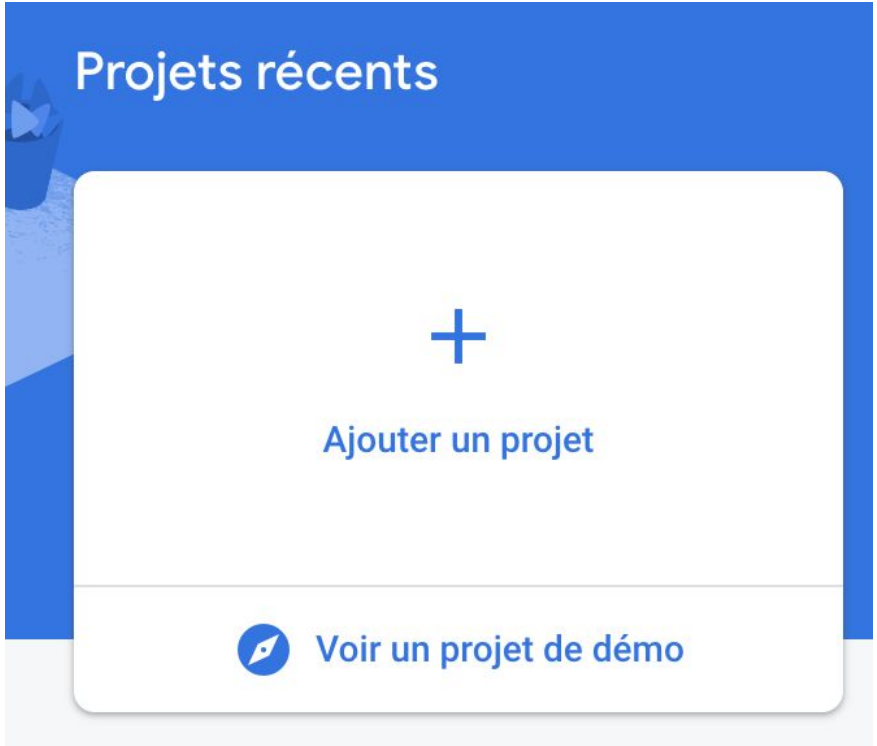
Pour cela, Firebase a été décomposée en plusieurs produits extrêmement riches et adaptés au monde du mobile, dont **la liste est disponible à ce lien**.

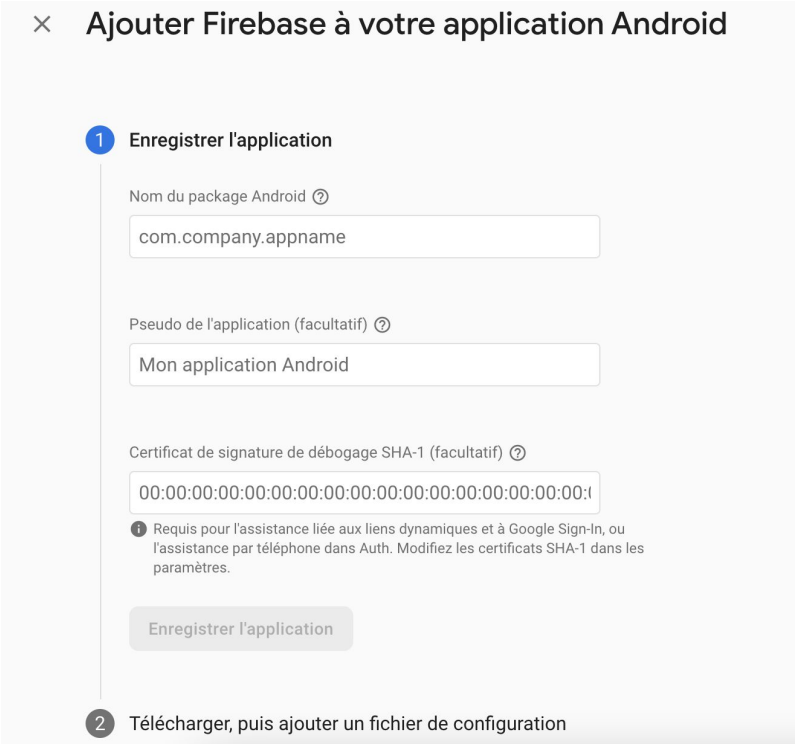
Un seul cours ne suffirait pas pour étudier tous les produits disponibles sur Firebase !

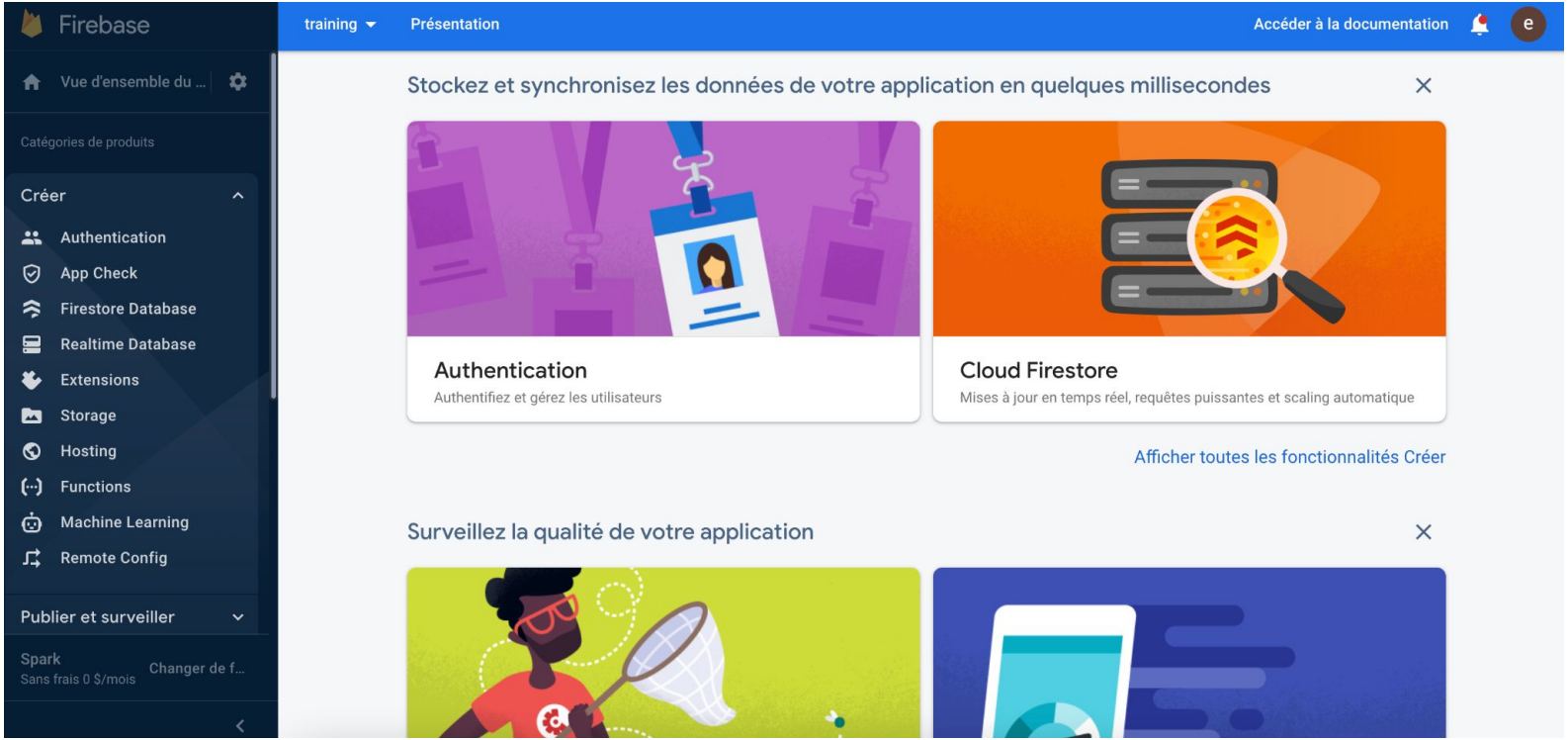
Un autre avantage non négligeable de Firebase est que vous pouvez commencer à l'utiliser **totalement gratuitement**. Firebase utilise un système "**Pay as you go**", vous commencez à payer uniquement lorsque vous atteignez un nombre importants d'utilisateurs, pour les fonctionnalités dont vous avez besoin.

Intégrez Firebase dans une application Android




Projets récents
Ajouter un projet
Voir un projet de démo

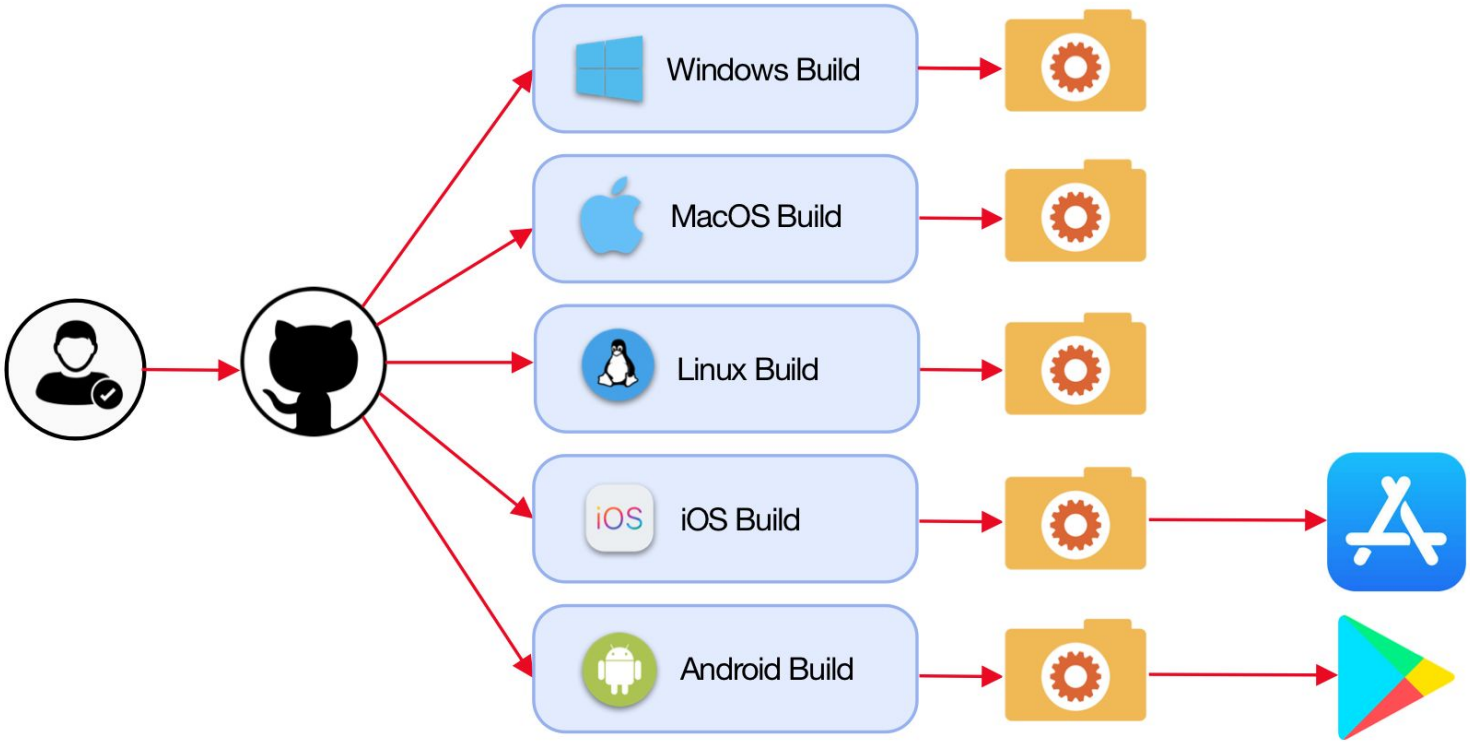

Ajouter Firebase à votre application Android
1
Enregistrer l'application
Nom du package Android
com.company.appname
Pseudo de l'application (facultatif)
Mon application Android
Certificat de signature de débogage SHA-1 (facultatif)
00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
Requis pour l'assistance liée aux liens dynamiques et à Google Sign-In, ou l'assistance par téléphone dans Auth. Modifiez les certificats SHA-1 dans les paramètres.
Enregistrer l'application
2
Télécharger, puis ajouter un fichier de configuration


Firebase
training
Présentation
Accéder à la documentation
e
Vue d'ensemble du ...
Catégories de produits
Créer
Authentication
App Check
Firestore Database
Realtime Database
Extensions
Storage
Hosting
Functions
Machine Learning
Remote Config
Publier et surveiller
Spark
Sans frais 0 $/mois
Changer de f...
Stockez et synchronisez les données de votre application en quelques millisecondes
Authentication
Authentifiez et gérez les utilisateurs
Cloud Firestore
Mises à jour en temps réel, requêtes puissantes et scaling automatique
Afficher toutes les fonctionnalités Créer
Surveillez la qualité de votre application


Firebase
Vue d'ensemble du p...
Catégories de produits
Créer
Publier et surveiller
Analytics
Engager
Tous les produits
Spark
Sans frais 0 $/mois
Changer de f...

- **CI/CD Github Actions**

GitHub Actions est une plate-forme d'intégration et de livraison continues (CI/CD) qui vous permet d'automatiser votre pipeline de construction, de test et de déploiement. Vous pouvez créer des flux de travail qui créent et testent chaque demande d'extraction vers votre référentiel, ou déployer des demandes d'extraction fusionnées en production. GitHub Actions va au-delà de DevOps et vous permet d'exécuter des flux de travail lorsque d'autres événements se produisent dans votre référentiel. Par exemple, vous pouvez exécuter un flux de travail pour ajouter automatiquement les étiquettes appropriées chaque fois que quelqu'un crée un nouveau problème dans votre référentiel. GitHub fournit des machines virtuelles Linux, Windows et macOS pour exécuter vos flux de travail, ou vous pouvez héberger vos propres exécuteurs auto-hébergés dans votre propre centre de données ou infrastructure cloud.

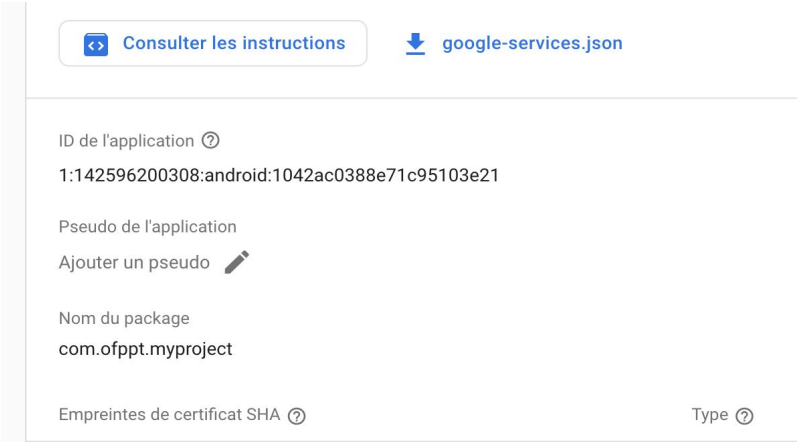

Windows Build
MacOS Build
Linux Build
iOS
iOS Build
Android Build

TP : Github Actions

https://gist.github.com/NizarETH/35caf3f537f66ee8c03c6181bbc3b8e3

NB : FIREBASE_TOKEN => Exécuter la commande dans le terminal : *firebase login:ci*

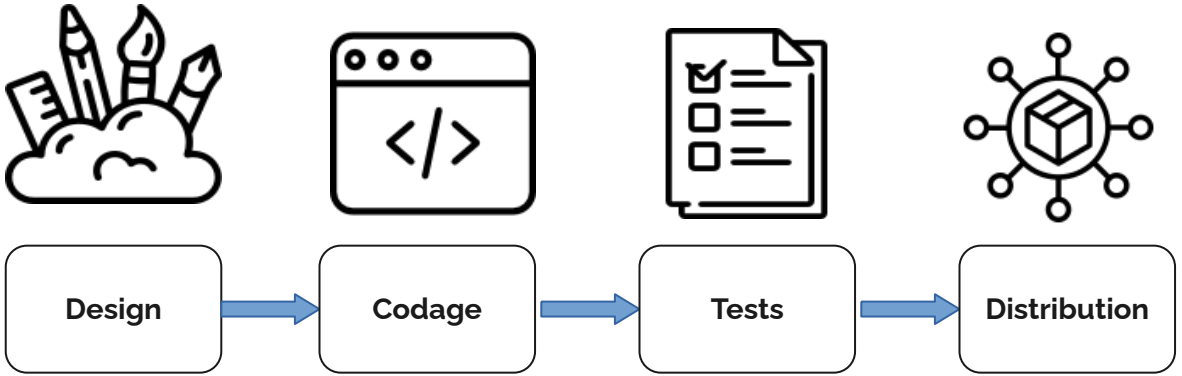
NB : FIREBASE_APP_ID =>



- **Publiez votre application Android sur le PlayStore**

Introduction

- La distribution de l'application est la dernière étape d'une série de processus :



- **Mettre de l'ordre dans votre application.**

  Tests, nettoyage, version, nom, icône, licence, etc.

- **Signature de l’application.**

  Indispensable pour pouvoir la publier.

- **Teste de l’application comme un utilisateur final.**

  Se mettre à la place de l'utilisateur final

- **Publication de l’application.**

  Déployez votre application en ligne.

Mettre de l'ordre dans votre application

- Nettoyez le projet : il faut effacer toutes les traces et les fichiers ou dossiers utilisés pour le débogage et lors des tests.

  Il faut examiner par exemple le contenu des répertoires « **res** » et « **assets** ».

- Arrêtez la journalisation : supprimer tout ce qui est en rapport avec « **Logcat** ». On peut par exemple baliser leurs utilisations comme suit :

```
if(BuildConfig.DEBUG) {
        //si on se trouve en version debug, alors on affiche
        //des messages dans le Logcat
        Log.d(...);
    }
```

- ou utiliser **Timber** : https://github.com/JakeWharton/timber

- **Désactivez le débogage** : soit la retirer du fichier « **AndroidManifest.xml** » ou la mettre à « **false** ». android:debuggable="false"

- **Numéro de la version**

  android:versionName

  - Donne une valeur sous la forme d'une chaîne de caractères à la version de votre application (par exemple « **1.0 alpha** » ou « **2.8.1b** »).
  - Cet attribut est visible à l'utilisateur

  android:versionCode

  - Cet attribut n'est pas visible à l'utilisateur.
  - Il ne peut contenir que des nombres entiers.
  - Si votre ancien numéro était à « **1** », en le mettant à une valeur supérieure à « **1** », le Store va conclure qu'il s'agit d'une version plus récente.

- **Nom du package**

  C'est le nom utilisé pour identifier votre application. Vous ne pouvez pas le changer entre deux versions. Ce nom est unique et permanent. Par ailleurs, il faut qu'il se démarque.
  (com.companyName.appName.flavor)

- **Icône**

  C'est un détail pour vous, oui, mais … c'est le premier contact.

https://material.io/design/iconography/product-icons.html#design-principles

https://developer.android.com/distribute/best-practices/develop/use-material-design?_hl=FR
https://www.usabilis.com/material-design-lui-selon-google/

http://kunzisoft.blogspot.com/2017/02/creer-un-icone-materila-design-android.html

- **Licence d'utilisation (facultatif)**
  À vous de décider le type de licence que vous voulez associer avec votre application.
- **La version ciblée**
  Dans le manifeste, vous devez décider de la version minimale de l'API. Un choix permet soit de restreindre le nombre d'utilisateurs ou de l'élargir.

- **Tester, tester et encore tester**
  Ne pas oublier de faire des tests exhaustifs en tant que développeur afin de valider la robustesse de votre application.
  https://developer.android.com/training/testing/fundamentals

  Faire l'exercice pratique (« codelab ») :

  https://codelabs.developers.google.com/codelabs/android-testing/#0

Signature de l'application

- Les applications dans « **Google Play** » sont représentées par un fichier au format « **apk** ».
- On commence par exporter le projet de l'interface de développement sous la forme d'un fichier « **apk** ».
- Cette procédure fait en sorte que le projet est exporté en mode « **release** » et non pas « **debug** ».
- Durant ce processus, nous sommes ramenés à signer l'application.
- Android exige d'une application qu'elle soit préalablement signée avant d'être installée ou mise à jour sur un appareil.

- **Signer une application permet de la sécuriser :**
  - On garantit ainsi son intégrité.
  - On définit l'auteur de l'application.
  - La mise à jour de l'application ne peut avoir lieu que si elle possède une signature provenant du même certificat.

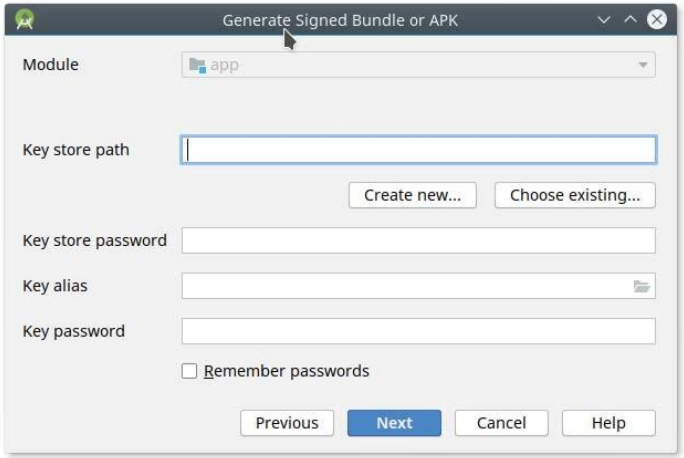
- **Utilisez sa propre clé :**
  - Vous évitez ainsi une signature générique « simple » à trouver.
  - Utilisez la même clé pour signer toutes ses applications. Ces applications vont pouvoir fonctionner dans un même processus (dans « une » seule application) si elles le désirent. Elles peuvent aussi échanger et partager des données de manière sécuritaire.
  - Utilisez un mot de passe abracadabrant.
  - Évitez de perdre votre clé! Sinon impossible de mettre à jour votre application.
  - Évitez aussi de vous la faire voler!

- **Signez l'application en utilisant Android Studio :**

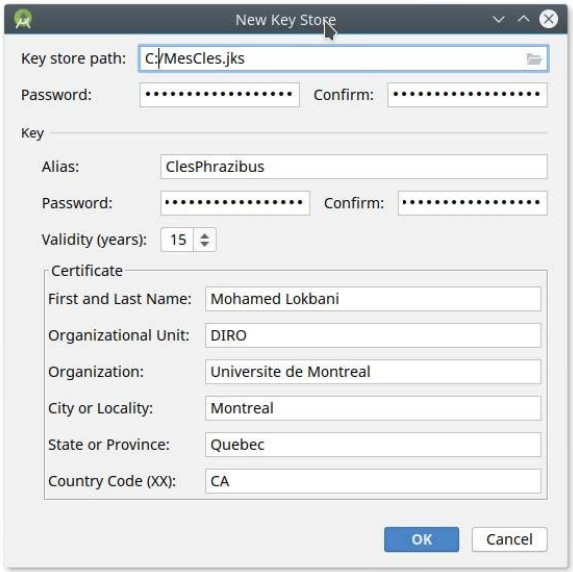
  https://developer.android.com/studio/publish/app-signing

  o Dans le menu, cliquez sur « build » puis « Generate signed Bundle/APK… » , cochez « APK », « Next »

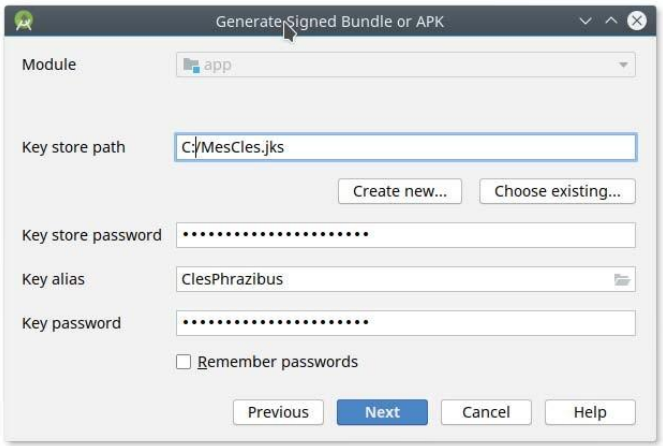
- Nous devons préciser une clé. Pour cela, nous allons commencer par générer un dépôt (« **Key store** ») pour les clés. Cliquez sur « **Create new** » :



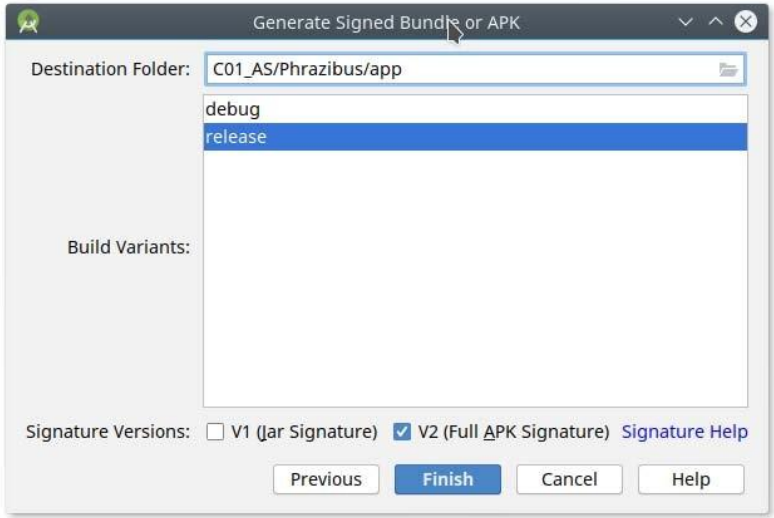
o Complétez les différents champs et cliquez sur « **OK** »



o Vous allez obtenir cette fenêtre complétée, cliquez sur « **Next** » :



- Précisez le répertoire où le fichier « **apk** » sera stocké, la variante « **release** » et la version « **V2** » puis cliquez sur « **Finish** » :



- Si l'opération a réussi, vous allez obtenir ce message :

  Generate Signed APK
  APK(s) generated successfully for 1 module:
  Module 'app': locate or analyze the APK.

- Notez le chemin où le paquetage « APK » sera sauvegardé.

- Il est possible aussi de configurer « Android Studio » pour qu'il signe automatiquement votre application. Voir pour cela les indications décrites sur le lien mentionné au début de cette section.

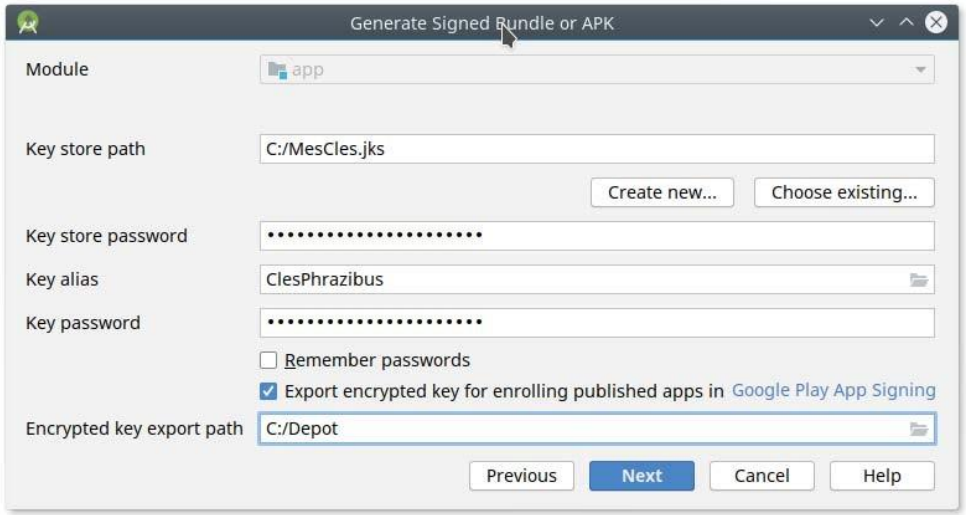
- **Signez un « bundle »**

Avant la version « 3.2 » d'Android Studio, le développeur devait générer une version du fichier « APK » pour les différentes variantes de son application (taille de l'écran, type de l'appareil, etc.). Il pouvait aussi générer un seul fichier « APK » qui permettait de couvrir toutes ces variantes. Or la taille de ce fichier était très large. Ajouter à cela que la création de plusieurs variantes nécessite d'inclure des heures de travail supplémentaires pour réaliser cette tâche. Une étude menée par Google a montré la taille de l'application, a une influence directe sur la décision de l'utilisateur de l'installer ou pas. Pour pallier à tous ces problèmes, Google a proposé une livraison dynamique. Le développeur doit inclure les différentes images pour les différents appareils dans un fichier compressé « bundle ». Au moment de l'installation, Google va générer de manière dynamique à partir de ce « bundle », le fichier « APK » qui correspond aux caractéristiques de l'appareil.

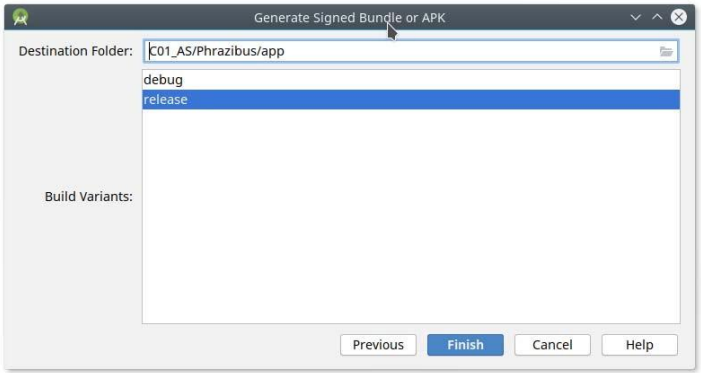
C'est l'approche recommandée par Google.

o Dans le menu, cliquez sur « **build** » puis « **Generate signed Bundle/APK…** » , cochez « **Android App Bundle** », « **Next** » :

o Complétez les différents champs, puis cliquez sur « **Next** »



- Choisissez la destination et la variante, puis cliquez sur « **Finish** ». Attention l'application doit inclure aussi un numéro de version comme mentionné au début de chapitre.



- Si l'opération est un succès, vous allez obtenir cette série de messages :

Generate Signed Bundle

App bundle(s) generated successfully:
Module 'app': locate or analyze the app bundle.
Locate exported key file.

- Signez l'application « **manuellement** »
    - Générez une clé :

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.

```
signingConfigs {
    releaseconfig {
        keyAlias 'release_key'
        keyPassword 'release_pwd'
        //storeFile file('C:\\Users\\lenovo\\AndroidStudioProjects\\app\\Release.jks')
        storeFile file('/Users/lenovo/StudioProjects/app/Release.jks')

        storePassword 'smssurtymar'
    }
}
```

```
buildTypes {

    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-rules.pro'
        signingConfig signingConfigs.releaseconfig

        firebaseCrashlytics {
            mappingFileUploadEnabled false
        }
        manifestPlaceholders = [crashlyticsCollectionEnabled:"true"]
    }
```

- Compilez votre projet en mode « **release** ».
- On obtient ainsi une application en mode « **release** » signée.

Distribuer l'application

- Assurez-vous d'abord d'avoir suivi les recommandations pour déployer l'application.

https://support.google.com/googleplay/android-developer/answer/7159011
Le déploiement peut se faire de plusieurs manières :

http://developer.android.com/distribute/tools/open-distribution.html

- Manuelle : via le web, espace local ou sur le réseau.

- Boutique en ligne : une boutique qui peut héberger des applications gratuites ou payantes, dédiées entre autres à des appareils Android. C'est l'option idéale si l'on veut ratisser large.

Déploiement manuel

- Vous pouvez donner accès à l'application via le web par exemple. Vous n'avez qu'à fournir le lien « URL » vers la page web en question.

  Telecharger App

  - Vous devez activer dans votre appareil l'option « **installation à partir d'une source inconnue** ».

  - Le serveur qui héberge la page doit ajouter ce « MIME » :

    « **application/vnd.android.package-archive** »

Exercice : **Déploiement manuel**

1- il faut créer un projet Android (JAVA ou Kotlin) : avec 2 Activities (MainActivity1 et MainActivity2 qui sont reliées par un Button), ensuite créer un APK

2- Créer une Page HTML (Image, titre et détails) et insérer dedans un lien pour l'APK généré.

3- hébérger la Page HTML pour donner accès à l'application via le web

- Vous pouvez aussi la déposer dans un espace partagé. Dans ce cas, l'utilisateur lambda doit connaître les étapes à suivre pour installer l'application.

    - **adb -s nom_de_appareil install monpaquetage.apk**

- Vous pouviez copier l'application localement sur votre appareil à travers le port USB. Par la suite, vous allez utiliser l'explorateur de fichiers pour vous rendre à l'endroit où le fichier a été sauvegardé. Finalement, il suffit de cliquer sur le fichier pour effectuer l'installation de l'application.

    - Vous devez activer dans votre appareil l'option « **installation à partir d'une source inconnue** ».

Déploiement à travers « Google Play »

- « **Google Play** » store est une place de marché qui a été créée et exploitée par Google qui permet aux Développeurs enregistrés dans certains pays de distribuer des Produits directement aux utilisateurs d'Appareils.

- Créée le 6 mars 2012 de la fusion de « Android Market », « Google Music » et « Google eBookstore ».

- En date du 17 Septembre 2021, la boutique contient 2,705,806 des applications gratuites 96.3 % et 102,850 applications payantes 3.7%.

  http://www.appbrain.com/stats/free-and-paid-android-applications

- La boutique est utilisable par l'intermédiaire d'un compte Google, c'est-à-dire « **Gmail** ».

- Ce compte peut-être associé à une personne physique ou morale.

- La personne gère les accès à la boutique pour tous les appareils dont elle est propriétaire, sans restriction de nombre.

- Ainsi elle peut déployer une application payante ou gratuite acquise sur la boutique pour l'ensemble des appareils dont elle est propriétaire, en ne payant qu'une seule fois l'application (payante).

- Pour rendre disponible une application via la boutique, il faut avoir un compte développeur.

- Un compte développeur est un compte de publication attribué aux développeurs qui permet la distribution de Produits via le Play Store.

- Ce compte requiert des frais d'inscription uniques de 25USD.

- Pour une application payante : 30% des revenus sont pour Google, 70% pour le développeur.

- Le compte développeur nécessite aussi un compte « Gmail ».

https://developer.android.com/distribute/googleplay/start.html

- Avec votre compte « Gmail », commencez par vous connecter à cette adresse :

  https://play.google.com/apps/publish/signup/

- Acceptez les conditions générales associées à la distribution sur « Google Play » pour les développeurs, puis cliquez « Continue to payment ».

- Réglez les frais d'inscription de 25USD.

- L'étape finale consiste à fournir des informations relatives à votre compte (nom du développeur, adresse courriel et numéro de téléphone).

- Google vous redirige par la suite sur la console développeur de « Google Play » :

https://developer.android.com/distribute/googleplay/developer-console.html

- Cette interface va vous permettre d'importer des applications, de définir les prix, d'ajouter des utilisateurs de comptes et gérer les autorisations et finalement, de consulter des rapports, des statistiques et des informations relatives à vos applications.

- L'interface pour ajouter un nouvel utilisateur du compte :

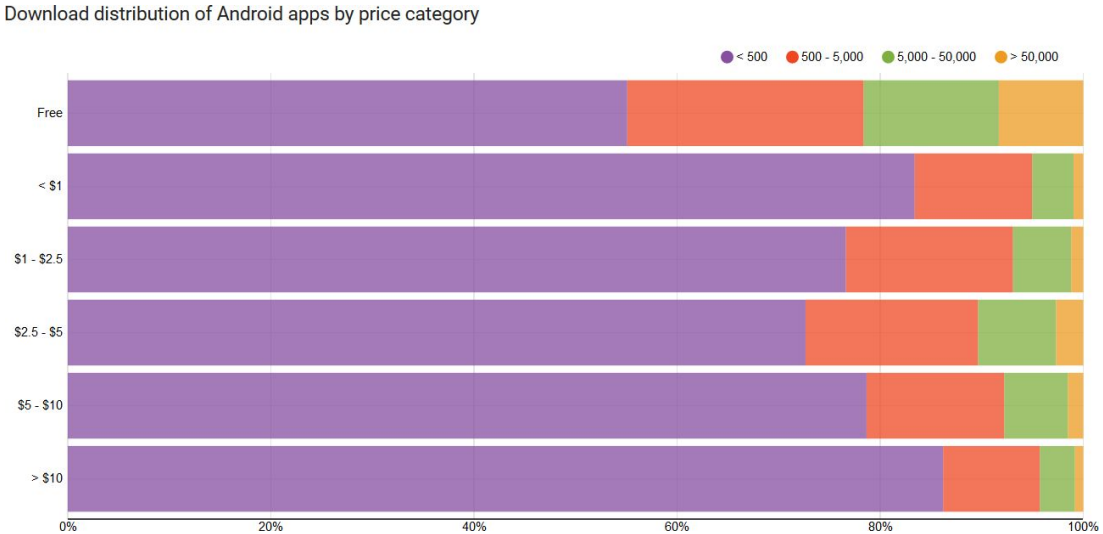
- La publication d'une application nécessite l'envoi du fichier « aab » et de fournir des captures d'écran et des métadonnées relatives à votre application.

- Si une application est proposée gratuitement, elle ne peut pas changer d'état.

- Si une application payante est rendue gratuite, elle ne peut plus changer d'état.

- Une application peut-être :
  - **Payante** : elle est facturée avant qu'elle ne soit téléchargée.
  - **Gratuite** : mais vraiment gratuite sans les « extras » !
  - **Mixte** : offrir une version gratuite basic et une version améliorée payante.
  - **Financée par la Pub** : elle est gratuite, mais vous allez lui inclure de la pub.
  - **Produits intégrés** : elle est gratuite, mais vous allez lui inclure un contenu qui peut-être acheté au fur et à mesure (une arme pour un jeu, un véhicule pour une course de voitures, etc.).
  - **Abonnements** : le contenu nécessite le paiement d'un abonnement périodique. Cette technique est utilisée pour contrer le fait qu'un compte peut servir plusieurs appareils dont il est responsable. L'abonnement proposé est généralement lié à un compte et à un appareil.

- Quel prix fixé?



- 90% des applications dont le prix est supérieur à 10$ ont été téléchargées moins de 500 fois! (http://www.appbrain.com/stats/free-and-paid-android-applications)

- Visibilité

https://support.google.com/googleplay/android-developer/answer/4448378?hl=fr&ref_topic=3450986

« La fonctionnalité de recherche de Google Play tient compte de l'expérience utilisateur générale que votre application procure en se basant sur le comportement et les commentaires des utilisateurs. Les applications sont classées en fonction de plusieurs facteurs tels que les notes, les avis ou le nombre de téléchargements. Bien que le poids et la valeur de chaque facteur soient confidentiels en raison de leur appartenance à l'algorithme de recherche de Google, vous pouvez réaliser les opérations ci-dessous afin d'améliorer la visibilité de votre application :

- Créez une expérience utilisateur durable et enrichissante pour vos utilisateurs.
- Améliorez votre application en y apportant des mises à jour régulières.
- Encouragez vos utilisateurs à laisser un avis et à donner une note à votre application.
- Fournissez un service client de qualité en répondant aux utilisateurs et en résolvant leurs problèmes. »

Déploiement sur d'autres boutiques alternatives

- Noyer dans la masse : 2,5 millions d'applications sur « Google Play »!
- Le marché chinois : il est possible de télécharger de la boutique « Google Play » que les applications gratuites.
- Avoir un autre ratio de partage des gains que (70/30) de « Google Play ».
- Publier ailleurs l'application « rejetée » par « Google Play ».
- Difficile d'ignorer des boutiques alternatives devenues trop grandes.

- L'alternative doit donc permettre de publier aussi des applications payantes, de fournir des statistiques adéquates, de proposer une interface dans une langue couramment utilisée (ou plusieurs langues).

http://en.wikipedia.org/wiki/List_of_mobile_software_distribution_platforms

- 2 exemples :
  - Amazon App Store : il surfe sur la vague du magasin en ligne « Amazon ». Il offre des applications pour Kindle et Android.
  - SlideMe : réseau alternatif #1 à « Google Play ». Il est installé par défaut sur de nombreux terminaux.

Toutes les applications
Boîte de réception 40
Conformité aux règles
Utilisateurs et autorisations
Gestion des commandes
Télécharger des rapports
Détails du compte
Page du développeur
Comptes de développeur associés
Journal d'activité
Configuration

Toutes les applications

Consulter les applications et les jeux auxquels vous avez accès dans votre compte de développeur

Créer une application

Applications épinglées

Épinglez des applications ici pour y accéder rapidement et consulter les statistiques clés

9 applications

Filtrer par

Tout

Faire une recherche par nom d'application...

Application	Audience ayant installé l'application	État de l'application	État de la mise à jour	Dernière mise à jour
Ghazala Horaires & tarifs - غزالة com.stanly.ghazala	10	Application suspendue par Google		27 déc. 2018
My Beautiful Selfie - Filter Cam... com.stanly.selfie	0	Supprimé		27 sept. 2018
My puzzle game com.stanly.puzzle	0	Supprimé par Google		29 déc. 2018

Créer une application

Informations sur l'application

Nom de l'application

Le nom de votre application apparaîtra tel quel sur Google Play. Soyez concis et n'incluez pas de prix, de classement, d'emoji ni de symboles répétitifs. 0/50

Langue par défaut

Anglais (États-Unis) – en-US

Application ou jeu

Vous pouvez modifier la catégorie plus tard dans les paramètres du Play Store

- Application
- Jeu

Application gratuite ou payante

Vous pouvez modifier cette application plus tard sur la page de l'application payante

- Gratuit
- Payant

Tableau de bord

Commencer à configurer votre application

Lors de la configuration, les tâches à accomplir pour rendre votre application opérationnelle sont indiquées dans le tableau de bord. Vous trouverez ainsi des recommandations pour gérer, tester et promouvoir votre application. Dès que vous avez terminé une tâche, revenez ici pour découvrir ce que vous pouvez faire d'autre.

Masquer

Démarrer les tests maintenant

Publier votre application de manière anticipée en vue de réaliser des tests internes sans vérification

Partagez votre application avec un maximum de 100 testeurs internes pour identifier les problèmes et bénéficier de précieux retours

Masquer les tâches

- Sélectionner les testeurs >

CRÉER ET DÉPLOYER UNE RELEASE

- Créer une release >
- Vérifier et déployer la release

Tableau de bord

Fournir des informations sur votre application et configurer sa fiche Play Store

Décrivez-nous le contenu de votre application et gérez sa présentation sur Google Play

Masquer les tâches

DÉCRIRE LE CONTENU DE VOTRE APPLICATION

- Accès aux applications ›
- Annonces ›
- Classification du contenu ›
- Cible ›
- Applications d'actualités ›
- Applis sur le suivi des contacts et le statut COVID-19 ›

GÉRER L'ORGANISATION ET LA PRÉSENTATION DE VOTRE APPLICATION

- Sélectionner la catégorie de votre application et indiquer vos coordonnées ›
- Configurer une fiche Play Store ›

Publier votre application

Tester votre application auprès d'un plus large groupe de testeurs que vous contrôlez

Les tests fermés vous permettent de tester votre application auprès de plus larges groupes de personnes. Vous pouvez contrôler l'accès à l'aide des adresses e-mail ou de Google Groupes.

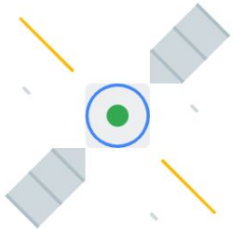
Effectuez d'abord les tâches de configuration initiales Masquer les tâches

CONFIGURER LE CANAL DE TEST FERMÉ

- Sélectionner les pays et les régions
- Sélectionner les testeurs

CRÉER ET DÉPLOYER UNE RELEASE

- Créer une release
- Vérifier et déployer la release



Publier votre application sur Google Play

Publiez votre application afin qu'elle soit disponible pour tous les utilisateurs sur Google Play en la diffusant dans le canal de production

Effectuez d'abord les tâches de configuration initiales Masquer les tâches

Sélectionner les pays et les régions

CRÉER ET DÉPLOYER UNE RELEASE

Créer une release

Vérifier et déployer la release

Bibliographies

- Centre d'aide « Google Play Developer »
  https://support.google.com/googleplay/android-developer/?hl=fr#topic=3452890

- Google Play de A à Z
  https://developer.android.com/distribute/googleplay/about.html

- Publier une app sous Android : petite présentation de Google Dev Console
  http://electricstudio.fr/news/publier-un-app-sous-android-petite-presentation-de- google-dev-console/

- Vidéos (ancien format de Google Play)
  https://www.youtube.com/watch?v=uSemegamBZA
  https://www.youtube.com/watch?v=9lKRiq64kJI

- Conseils pour le marketing d'application sur Google Play
  http://www.apptamin.com/fr/blog/marketing-app-google-play/

- Publier et rentabiliser une application
  http://openclassrooms.com/courses/creez-des-applications-pour-android/publier- et-rentabiliser-une-application

- App Stores Growth Accelerates in 2014
  http://blog.appfigures.com/app-stores-growth-accelerates-in-2014/

C’est la fin du module

Merci !