# Systèmes mobiles
# Laboratoire n°1 : Introduction aux activités *Android*

22.09.2017

## Introduction

Cet exercice est constitué d'une manipulation de laboratoire destinée à implémenter une application élémentaire sur un émulateur *Android* et/ou sur un smartphone ainsi que de vous familiariser avec l'environnement de développement *Android*.

## Installation de l'environnement de développement

Vous pouvez télécharger et suivre les instructions d'installation sur la page du site *Android* : https://developer.android.com/studio/index.html. On prendra garde au point suivant :

- Depuis début 2015, Eclipse n'est plus supporté par Google comme environnement de développement ; il est nécessaire d'utiliser *Android Studio*, qui est basé sur la plate-forme *IntelliJ*. Attention aux anciens tutoriels disponibles sur le web.

## Les outils

On désire ici mentionner quelques-uns des principaux outils à disposition dans l'environnement *Android*. En plus de l'édition de code et de la gestion de projet classiques d'*IntelliJ*, il existe un certain nombre d'outils spécifiques au SDK *Android* qui sont accessibles directement depuis *Android Studio*.

### L'émulateur

L'émulateur est compris dans le "pack" SDK de base. Il s'agit d'une machine *Android* virtuelle relativement complète et raisonnablement fonctionnelle, privée toutefois de certaines possibilités, parmi lesquelles :

- Pas d'accès au hardware du smartphone (USB, Bluetooth, NFC)
- Pas d'accès au GPS (il est possible d'imposer des coordonnées de localisation à la main dans l'émulateur)
- Accès limité aux fonctions de téléphonie (bien sûr ni téléphone ni SMS)
- Accès limité à Internet. Ici, le problème est que l'espace d'adresses est commun avec la machine hôte ; ainsi, par exemple "localhost" désigne logiquement l'émulateur, mais l'adresse 10.0.2.2 pointe sur la machine hôte de développement
- Lire à ce sujet https://developer.android.com/studio/run/emulator.html

## L'exécution sur cible par USB

Pour les possesseurs d'un smartphone *Android*, il est possible et certainement préférable de faire exécuter le code directement sur la cible par la connexion du smartphone au poste de développement via un câble USB. L'installation d'un driver spécifique est sans doute requise pour ce faire ; se référer à la documentation de *Google* ou du constructeur. Vous devrez activer le mode développeur sur les versions *Android* les plus récentes puis autoriser le débugage par USB.

## Editeur d'interfaces utilisateurs, éditeur XML

*Android Studio* contient un petit éditeur d'interfaces utilisateurs qui permet théoriquement de générer des interfaces graphiques XML de manière visuelle. A peu près fonctionnel pour des interfaces très primitives, cet éditeur montre vite ses limites lorsqu'il s'agit de composer des écrans plus sophistiqués. Il est généralement conseillé de faire d'emblée l'effort de comprendre la syntaxe de description d'interfaces XML de *Android*, en sachant qu'on sera probablement tôt ou tard amené à effectuer cet effort ! Il en va de même pour l'éditeur XML inclus dans *Android Studio*, généralement plus gênant qu'utile pour éditer les écrans ou le fichier manifest.

## Logcat

L'outil probablement le plus précieux pour le dépannage de vos applications. Logcat est la contraction de la commande catalog de Linux/Unix et de log file. La machine virtuelle ART (ou dalvik) envoie tous les événements concernant le mobile sur ce flot de sortie, indépendamment de l'émulateur. Ce flot de sortie est donc aussi utilisable en production. Divers librairies permettent de collecter les logs après un crash d'une application en production et de les envoyer au développeur qui pourra se livrer à un diagnostic plus précis. Par exemple : http://www.acra.ch/

En développement, la sortie log est visible en temps réel directement depuis *Android Studio*.

## Log, System.out.println(), System.err.println()

Le complément à Logcat, qui va permettre de tracer l'exécution du code, également en production. Log est une classe permettant de générer des enregistrements de log qui vont apparaître avec un label (par exemple le nom de l'application) et une sévérité donnée dans la sortie Log. Les println() sont généralement redirigés sur la même sortie que les Log, ce qui signifie qu'ils apparaîtront également dans le listing, mais sans identification particulière. Toutefois prenez l'habitude d'utiliser la classe Log plutôt que System.out/err.println, car non seulement cela vous permettra de filtrer les logs par leur sévérité, mais la classe Log permet en plus de lier une exception en troisième paramètre.

## *Google*

La plus abondante source de renseignements pour le développement sur *Android* est sans conteste le net. En cas de problème, poser sa question (si possible en anglais, les forums francophones étant bien moins abondamment pourvus que les anglophones) directement sur la ligne de requête du moteur de recherche, éventuellement en la précédant du mot "android ». Par exemple, on peut imaginer une question sous la forme suivante :

- android popup dialog example

qui devrait vous donner des indications sur la manière de réaliser un des points de la manipulation ci-dessous. Le site sans conteste le plus souvent référencé, et le mieux pourvu en renseignements précieux est le bien connu www.stackoverflow.com (vérifiez que la question est similaire à la votre et cherchez en priorité la réponse marquée d'un vu vert). Mais le site mit sur pied par *Google* pour les développeurs *Android* est aussi bien fournis en renseignements précieux. L'important est de

comprendre et d'être assez critique vis-à-vis du code que l'on trouve sur le net, bien souvent sa qualité laisse à désirer et il n'est plus forcément d'actualité. Vous n'oublierez pas en revanche de citer la source !

Les plateformes mobiles évoluant très rapidement, vous apprendrez vite à jeter un œil à la date de la réponse ou du tutoriel.

De surcroît, même si votre collègue est bien meilleur codeur que vous, cela n'empêche pas d'essayer soi-même... C'est en effet le seul moyen pour qu'il ne reste pas définitivement meilleur que vous.

# Manipulation

La manipulation que nous vous proposons de réaliser est très simple ; il s'agit de réaliser une application qui demande à l'utilisateur par l'intermédiaire de deux champs textuels, un e-mail et un mot de passe, avec un bouton de validation (Ok). Cette application n'est pas forcément pertinente du point de vue de la logique de l'interface utilisateur, mais est plutôt destinée à vous faire prendre contact avec certains points élémentaires de la logique des activités et de certains composants sur *Android* (spécialement la logique des layouts). Vous utiliserez pour mettre en place l'application le modèle de projet (Template) fourni ci-dessous. Bien sûr, l'utilisation de ce modèle est optionnelle ; ceux qui le désirent peuvent en découdre avec les multiples options de développement de projet *Android* proposées par le SDK ; il se trouve simplement que dans le cadre de ce cours, nous n'en avons nul besoin ; un projet très basique nous suffira dans tous les cas.

Si la syntaxe de l'e-mail paraît incorrecte (essentiellement absence du signe "@"), on affichera un *Toast* (message temporaire au bas de l'écran) indiquant qu'il est nécessaire de mettre un email valide, et rester dans le même dialogue en conservant les entrées de l'utilisateur.

Si l'e-mail n'est pas connu de l'application (on aura codé en "dur" une série de couples email/mot de passe dans l'application), afficher un dialogue pop-up à quittancer affichant le message "Utilisateur ou mot de passe inconnu", et sur quittance de l'utilisateur, revenir à l'application en effaçant préalablement les entrées précédentes de l'utilisateur. Pour la création du pop-up, utiliser un *DialogBuilder* qui permet de se passer du fichier XML de définition d'interfaces. Le message affiché dans le dialogue popup ne sera pas codé en dur, mais extrait du fichier XML `res/values/strings.xml` (ressources texte), de manière à ce que l'application puisse ultérieurement être traduite en plusieurs langues.

En cas de succès (les entrées correspondent à ce qui a été prévu), on lancera une activité séparée permettant d'afficher l'e-mail introduit précédemment, l'IMEI du terminal (pour l'émulateur, ce sera "000000..." ou même `null` sur les versions d'*Android* 8.0 et plus[1]) ainsi qu'une photographie, censée représenter l'utilisateur qui vient de réussir son login, résidente sur la carte SD (utilisez par exemple : `Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS);` pour accéder au dossier des téléchargements).

Info : La commande suivante vous permettra de copier un fichier sur l'émulateur

```
adb push     <file-source-local>       <destination-path-remote>
adb push     perso.jpg                 /sdcard/Download/perso.jpg
```

[1] Depuis la version 26 du SDK, l'obtention de l'IMEI a été modifiée, la question 5 couvre cette modification

### Permissions, version simple

Afin de pouvoir lire le numéro IMEI du téléphone ou alors accéder au stockage externe, votre application nécessitera des permissions spécifiques à ajouter au fichier `AndroidManifest.xml` :

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

Ces deux permissions nécessitent depuis *Android* 6.0 (Api 23) une validation par l'utilisateur lors de l'exécution, c'est une procédure relativement complexe à mettre en place pour un débutant sur cette plateforme. Dans un premier temps, pour éviter cette difficulté, nous vous conseillons d'aller dans le fichier `build.gradle` (module app) et de mettre la valeur 22 pour la `targetSdkVersion`. Nous forçons donc l'utilisation de l'ancien système de permissions pour lequel il suffisait de lister les permissions utilisées dans le `manifest` pour qu'elles soient automatiquement accordées.

### Permissions, version avancée

Une fois que vous aurez terminé ce laboratoire, nous vous proposons d'implémenter la demande et gestion des permissions au runtime, comme il faut dorénavant le faire depuis *Android* 6.0.

Documentation officielle :
https://developer.android.com/training/permissions/requesting.html

Vous vérifierez votre solution sur un téléphone (ou émulateur) avec une version *Android* >= 6.0 et aussi idéalement avec une version plus ancienne.

Astuce : il existe certainement des librairies qui faciliteront grandement votre travail…

## Questions

1. Comment organiser les textes pour obtenir une application multi-langues (français, allemand, italien, langue par défaut : anglais) ? Que se passe-t-il si une traduction est manquante ?

2. Dans l'exemple fourni, sur le dialogue pop-up, nous affichons l'icône `android.R.drawable.ic_dialog_alert`, disponible dans le SDK *Android* mais qui n'est pas très bien adaptée visuellement à notre utilisation. Nous souhaitons la remplacer avec notre propre icône, veuillez indiquer comment procéder. Dans quel(s) dossier(s) devons-nous ajouter cette image ? Décrivez brièvement la logique derrière la gestion des ressources de type « image » sur *Android*.

   Info : *Google* met à disposition des icônes open source dans le style « Material Design » utilisé actuellement sur *Android* : https://design.google.com/icons/

3. Lorsque le login est réussi, vous êtes censé chaîner une autre Activity en utilisant un Intent. Si je presse le bouton "Back" de l'interface *Android*, que puis-je constater ? Comment faire pour que l'application se comporte de manière plus logique ?

4. On pourrait imaginer une situation où cette seconde Activity fournit un résultat (par exemple l'IMEI ou une autre chaîne de caractères) que nous voudrions récupérer dans l'Activity de départ. Comment procéder ?

5. Vous noterez que la méthode `getDeviceId()` du `TelephonyManager`, permettant d'obtenir l'IMEI du téléphone, est dépréciée depuis la version 26 de l'API. Veuillez discuter de ce que cela implique lors du développement et de présenter une façon d'en tenir compte avec un exemple de code.

6. Dans l'activité de login, en plaçant le téléphone (ou l'émulateur) en mode paysage (landscape), nous constatons que les 2 champs de saisie ainsi que le bouton s'étendent sur toute la largeur de l'écran. Veuillez réaliser un layout spécifique au mode paysage qui permet un affichage mieux adapté et indiquer comment faire pour qu'il soit utilisé à l'exécution.

7. Le layout de l'interface utilisateur de l'activité de login qui vous a été fourni a été réalisé avec un LinearLayout à la racine. Nous vous demandons de réaliser un layout équivalent utilisant cette fois-ci un RelativeLayout.

8. Implémenter dans votre code les méthodes `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, etc... qui marquent le cycle de vie d'une application *Android*, et tracez leur exécution. Décrivez brièvement à quelles occasions ces méthodes sont invoquées. Si vous aviez (par exemple) une connexion Bluetooth (ou des connexions bases de données, ou des capteurs activés) ouverte dans votre *Activity*, que faudrait-il peut-être faire, à votre avis (nous ne vous demandons pas de code ici) ?

9. Facultatif – Question Bonus - S'il vous reste du temps, nous vous conseillons de le consacrer à mettre en place la résolution des permissions au runtime.

## Durée

- 6 périodes
- A rendre le jeudi **12.10.2017** à **23h55**, au plus tard.

## Donnée

Un template d'application est disponible sur la page *Cyberlearn* du cours :
https://cyberlearn.hes-so.ch/course/view.php?id=10181

## Rendu/Evaluation

Pour rendre votre code, nous vous demandons de bien vouloir zipper votre projet *Android Studio*, vous veillerez à bien supprimer les dossiers `build` (à la racine et dans *app/*) pour limiter la taille du rendu. En plus, vous remettrez un document **pdf** comportant au minimum les réponses aux questions posées. Vous indiquerez aussi, si vous l'avez fait, comment vous gérez le cas des permissions avancées.

Merci de rendre votre travail sur *CyberLearn* dans un zip unique. N'oubliez pas d'indiquer vos noms dans le code, sur vos réponses et de commenter vos solutions.

**Bonne chance !**