Développement Android – TP2
Messagerie Instantanée - Novembre 2015

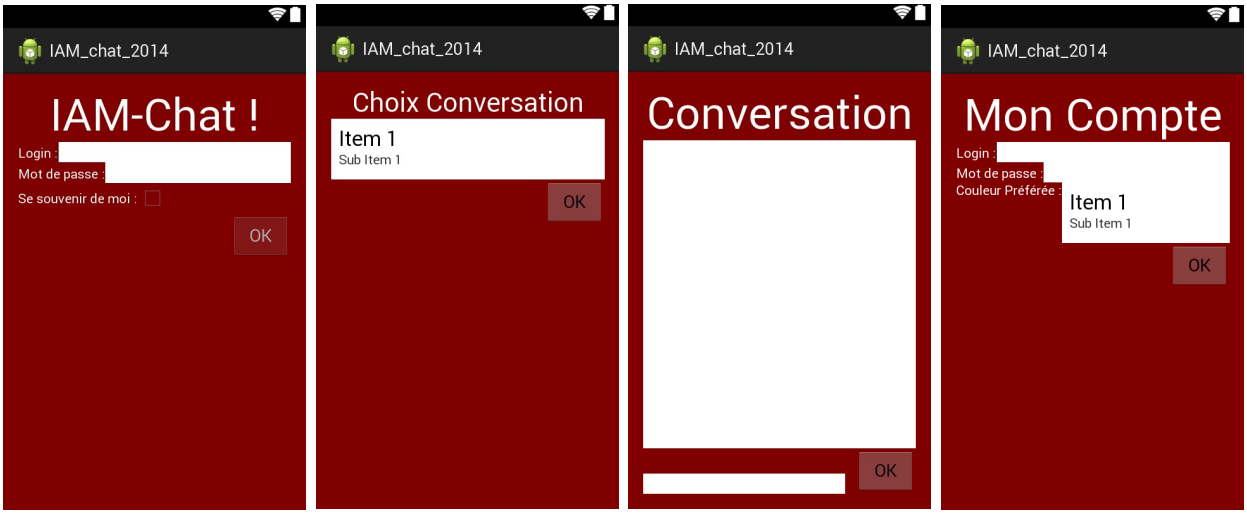
Objectif : Utiliser les connexions réseau du téléphone pour réaliser une application de clavardage envoyant des données encodées au format JSON, générer dynamiquement des interfaces à partir des données récupérées
Organisation : A chaque étape, ou en cas de problème, appelez l'enseignant pour lui montrer votre travail.

Partie 0 : Pré-requis
Un serveur Web, type WAMP, doit être disponible sur votre machine. Vérifier que c'est le cas et appelez votre enseignant sinon. Une architecture de type REST est préparée pour ce TP, à télécharger sur moodle. Copiez l'archive correspondante sous la racine de votre serveur Web. Vous utiliserez l'adresse 10.0.2.2/<repertoireDInstallation> pour effectuer vos requêtes sur ce serveur depuis vos terminaux virtuels.
NB : si vous souhaitez tester votre travail sur votre propre téléphone physique, ce serveur est également installé sur le serveur de la chaire : chaire-ecommerce.ec-lille.fr/ime5

Partie 1 : Design des interfaces (utiliser le code fourni sur moodle)

Nous commençons par designer les layouts de nos activités :

- l'activité de login permet à l'utilisateur de se connecter, l'application vérifie l'accès au réseau
- l'activité des préférences, accessible depuis le menu à tout moment (nous nous servirons d'une préférence Activity)
- l'activité « choix conversation »,
- l'activité « conversation »
- l'activité « compte utilisateur », accessible depuis le menu à tout moment



Récupérer les layouts sur moodle, ainsi que le fichier définissant les chaînes de caractères et les couleurs. Insérez-les dans votre projet. Mettez à jour les fichiers «colors.xml » et « strings.xml » pour modifier éventuellement les couleurs et les textes utilisés dans les layouts. Utiliser le layout de login dans l'activité principale de votre application.

Partie 2 : Menu (Cf. article « Menu Sous Android ») - déjà fait au TP1

Nous implémentons le menu, permettant de donner accès aux préférences utilisateur ainsi qu'à la gestion de son compte.

- Pour cela, définir dans res/menu/menu.xml deux Items nommés « Préférences » et « Mon compte ». N'oubliez pas de leur donner des identifiants pertinents.

- Dans **chaque activité** où ce menu devra apparaître, on complète l'appel onCreateOptionsMenu(Menu menu) de manière à développer la structure de menu choisie : getMenuInflater().inflate(R.menu.menu, menu);
- Une fois le menu créé, il faut réagir aux actions de l'utilisateur, à l'aide de la méthode onOptionsItemSelected(MenuItem item). Il est alors possible de récupérer l'identifiant de l'élément choisi à l'aide de la méthode getItemId(). Pour l'instant, nous déclencherons des alertes permettant de valider que chaque bouton est fonctionnel.

Partie 3 : Préférences de l'application – compte utilisateur (Cf. TP1 & articles « utilisation des preferenceativities » et « stocker des paramètres de configuration ») - Déjà fait au TP1

Nous souhaitons implémenter un mécanisme de persistance de données côté client, pour sauvegarder des préférences utilisateurs. Nous utiliserons la classe « PreferenceActivity », qui permet de stocker des données via des « SharedPreferences » .

- Commencer par définir les préférences souhaitées dans un fichier « preferences.xml » situé dans « res/xml ». Ce fichier contient les items suivants :
  - Un booléen pour savoir si l'on doit se souvenir des identifiants entrés lors d'une connexion (coché par défaut)
  - Une chaîne contenant l'URL du serveur (http://10.0.2.2/android_chat/data.php par défaut)
  - Une catégorie contenant les identifiants (login, passe) de l'utilisateur (vide par défaut)
- Définir une nouvelle activité héritant de la classe PreferenceActivity, qui pointe sur ce fichier xml à l'aide de la méthode addPreferencesFromResource. Ne pas oublier de déclarer cette activité dans votre manifeste
- Il ne reste plus qu'à appeler l'activité ainsi définie à l'aide d'un Intent depuis l'activité de login lorsque l'utilisateur choisit le menu correspondant.
- Lorsque l'activité **login** démarre ou redémarre (l'événement onStart suit l'événement onCreate), il faut lire les préférences à l'aide d'un objet SharedPreferences renvoyé par `PreferenceManager.getDefaultSharedPreferences` et s'en servir pour compléter les champs du formulaire si des valeurs existent dans les préférences et que la case est cochée.
- Si on décoche la case, il faut mettre à jour les préférences en conséquence et vider les champs d'identification login et passe.

Les 3 parties proposées ci-dessous sont relativement indépendantes et peuvent être traitées dans n'importe quel ordre : si l'on ne commence pas par les problématiques liées aux accès réseau, il suffit de déclarer une chaîne de caractères contenant la structure JSON que l'on s'attend à recevoir du réseau.

Partie 4 : Activité Login – Accès au Réseau (utiliser le code fourni sur moodle)

Nous souhaitons tester si le réseau est disponible pour rendre le bouton OK actif, et permettre l'envoi de requêtes au serveur. On se servira d'une classe utilitaire pour le réseau, dont le code est fourni.

- Commencer par définir une classe héritant de « Application », nommée « GlobalState », **dont le nom sera porté par notre application** (attribut android:name) dans notre manifeste.
- Dans chaque activité, vous pourrez récupérer une instance de cette classe à l'aide de la méthode **getApplication**.
- Cette classe définit un gestionnaire de cookies qui sera utilisé pour réaliser toutes les requêtes vers le serveur. Comme cette classe permet de stocker des cookies, cela nous permettra de rester identifié vis-à-vis du serveur, car notre application renverra le cookie d'identification au serveur à chaque requête :

```
CookieManager cookieManager = new CookieManager();
    CookieHandler.setDefault(cookieManager);
```

- Elle définit également un objet permettant d'accéder aux préférences partagées
- Cette classe définira les méthodes suivantes :
  - public alerter, qui envoie des toasts et génère des traces d'exécution
  - private convertStreamToString (fourni), utilisée dans la méthode requête
  - public verifReseau (fourni), renvoyant l'état de la connectivité réseau et déclenchant un Toast
  - public requete, envoyant une requête http
- Utiliser la fonction d'alerte fournie par cette classe dans l'activité login
- Ne pas oublier de déclarer dans le manifeste vos intentions d'utiliser INTERNET, ACCESS_NETWORK_STATE
- A chaque démarrage de l'activité login, vérifier l'état du réseau et modifier l'état du bouton OK. Vérifier que la fonction verifReseau est opérationnelle en désactivant le réseau par l'intermédiaire des paramètres du téléphone.

Partie 5 : Activité Login – JSON & Identification

Définir le comportement de l'activité login lorsque l'utilisateur clique sur le bouton OK :

- Si la case à cocher est cochée, on met à jour les préférences de l'application à l'aide d'un SharedPreferences.Editor (cf. poly). Tester que votre sauvegarde fonctionne correctement.
- Si les identifiants sont non vides, envoyer une requête au serveur avec les données login,passe. Pour cela, on utilise une Async task car les versions récentes de l'API Android interdisent de déclencher des traitements longs dans le thread d'affichage.
  - On pourra optionnellement informer l'utilisateur de l'avancement de la requête à l'aide d'une progress bar
- Afficher la réponse du serveur dans un toast.
- Le server renvoie un état de connexion en JSON dans le champ « connecte ». On utilisera un JSONObject et sa méthode getBoolean pour récupérer la valeur de ce champ (cf. poly).
  - NB : l'interprétation JSON étant également longue, il est recommandé de le placer dans un autre thread. On effectue donc l'interprétation dans l'AsyncTask précédente.
- Si le feedback est correct, on change d'activité en affichant l'activité ChoixConversation. On sauvegarde l'état de connexion de manière à pouvoir afficher l'activité d'affichage du compte depuis le menu de chaque activité. NB : le même menu doit s'afficher sur chaque activité
- Sinon, on affiche un feedback à l'utilisateur et on ne fait rien

Partie 6 : Refactorisation de code

Dans cette partie, on souhaite offrir à certaines activités notre capacité de déclencher des requêtes, tout en évitant de devoir à chaque fois rédiger le code des AsyncTasks... On propose donc de créer une classe RestActivity capable d'effectuer des requêtes de manière native, à l'aide d'une classe RestRequest héritant de `AsyncTask<String, Void, JSONObject>`. Notre classe RestActivity devra permettre d'envoyer une seule requête, ou une requête de façon périodique. Une méthode traiteReponse à implémenter dans les classes concrètes sera appelée lors de la réception d'une réponse et sera destinataire de l'objet JSON de réponse. Les requêtes périodiques pourront être adressées à des URL différentes à chaque fois (de manière à changer les données de la chaîne de requête éventuellement). Proposer une solution et l'implémenter.

Partie 7 : Activité ChoixConversation

Lors de l'affichage de l'activité (i.e. Sur onStart) , on récupère les données du serveur à l'aide de la requête action=getConversations et on s'en sert pour compléter le spinner.

- On pourra parcourir l'objet JSON à l'aide de getJSONArray et getJSONObject.

- On s'inspirera de la page http://www.mkyong.com/android/android-spinner-drop-down-list-example/ pour remplir le menu déroulant.
- On pourra améliorer le code en déclarant un objet conversation contenant des propriétés id et thème et en utilisant un « ArrayAdapter<Conversation> » capable de lire des listes « List<Conversation> ».
- Lorsque le bouton est appuyé, on passe un bundle de données à l'activité conversation pour lui indiquer l'identifiant de la conversation choisie.

Partie 8 : Activité Conversation

On termine en affichant les messages de la conversation choisie lorsque l'utilisateur sélectionne une des conversations.

- Lors du démarrage de l'activité, et après l'envoi de chaque nouveau message, on récupèrera une liste des précédents messages de la conversation à l'aide de la chaîne de requête action=getMessages&idConv=<id>. On affichera cette liste en respectant les couleurs des messages dans un layout de type « scrollView » permettant l'affichage de tout l'historique
- On aura soin de ne pas récupérer TOUS les messages mais uniquement ceux que l'on n'a pas déjà (cf. paramètre idLastMessage)
- Mettre en oeuvre le mécanisme d'envoi d'un nouveau message lors du clic sur le bouton OK, en envoyant action=setMessage&contenu=<message>&idConv=<id>