AUTHENTIFICATION

Théorie :

On utilise le Web service pour demander un jeton. Le jeton est transmis et vérifié à chaque interaction client-serveur.

Le jeton a une durée de validité mesurée – à expiration, il faut en demander un autre – donc une re-connexion.

A priori, c’est assez simple à faire en REST. On fait un POST avec l’Id et le mot de passe de l’utilisateur :

* S’il existe, on renvoie un jeton
* Sinon s’inscrire
  + Dans la plupart de vos config, les données utilisateurs sont stockées en local, donc on sait en local si un U existe ou non
  + Donc l’activité de création/transmission de jeton est sans ambiguité
    - Une première inscription – dépendamment des config :
      * On renvoie directement un jeton à l’U
      * Après inscription, le nouvel U doit se connecter pour obtenir un jeton
    - Une connexion
      * On renvoie un jeton dans tous les cas d’identification de l’U sur la base côté serveur

Dans le cadre de ce projet, je vous déconseille de fixer une limite de validité en temps du jeton, sauf si vous êtes très sûrs de vous. Pour ça, il faudrait un valideur, et une routine pour relancer la connexion quand celle-ci est expirée.

OAuth2.0

Il s’agit d’un protocole, plus que d’une méthode – voir :

<http://www.bubblecode.net/fr/2013/03/10/comprendre-oauth2/#HTTPS>

De fait, ce protocole est destiné en théorie à la communication entre client-tierce et serveur externe : l’application fait appel à un serveur non dédié. Ça n’est pas le cas dans le cadre de cet exercice. Mais c’est le protocole en vigueur si on utilise des endpoints.

L’intérêt de OAuth2 est qu’il permet la spécification d’un certains nombres de paramètres associés à la demande/délivrance d’autorisation :

* Exemple : le scope de l’autorisation : quelles applis, quelles fonctionnalités, etc.

Voici un exemple concret de traitement dans lequel l’objet d'authentification est géré de manière "statique".  
  
Quand une activité a besoin de cette authentification, elle regarde si l'authentification existe  
=> c'est le cas, elle l'utilise  
=> c'est pas le cas elle ouvre une boite de dialogue d'authentification.  
  
A la fin de la boite de dialogue (donc sur le onActivityResult()):  
SI CANCEL => fermeture.  
SI OK => utilisation de l'authentification.  
  
L'activité d'authentification va récupérer dans les SharedPreferences le dernier login utilisé, et afficher la boite avec login/password.  
SI CANCEL => on ferme l’appli;  
SI OK:  
... => ouverture d'une progress-dialog,  
... => lancement de l'async-task d'authentification de "token"  
  
A la fin de l'async-task d'authentification:  
... Dismiss de la progress-dialog  
SI Auth. Failure => Toast à l'utilisateur et c'est tout (retour à la boite de login donc)  
SI Auth. Success => Population de l'intent de retour (avec les informations nécessaires à l'objet Authentication), puis finish()  
  
Async-Task d'authentification:  
doBackground() => Appel du web-service d'authentification pour récupération du token.  
onPostExecute() => Appel de onLoginSucceded/onLoginFailed de l'activité (voir ci-dessus)  
  
  
Coté serveur, réception d'un login / password (possiblement encrypté, ex MD5 + Salt) / ID de téléphone, renvoi d'un token (validité variable sur un ID de téléphone, signature complètement privé).  
Et pour les appels MySQL vérification du token... (éventuellement renvoi d'un nouveau token pour éviter les expirations inopinées).