Android – Les Services Web

Emmanuel Conchon
(emmanuel.conchon@unilim.fr)

Introduction

- Le système Android est avant tout conçu pour des systèmes mobiles autonomes connectés à Internet
  - Ils peuvent donc tirer partie des fonctionnalités offertes par Internet et notamment des ressources REST
- REST est une architecture orientée vers les données et pas vers les services (ROA vs SOA)
  - Cette architecture a été introduite par Roy FIELDING dans sa thèse en 2000
  - C'est un style d'architecture inspirée du Web
  - Elle propose les fonctionnalités CRUD (Create, Read, Update, Delete)
  - Elle permet le transfert de données directement sur HTTP

L'architecture REST

- Quelques exemples d'entreprises fournissant des services REST

L'architecture REST

- L'objectif de REST est de proposer une architecture légère, simple et performante
- REST repose directement sur HTTP
  - Il ne l'utilise pas comme un protocole de transport comme SOAP
  - Cela permet de disposer d'une interface uniforme (celle d'HTTP)
    - GET
    - POST
    - PUT
    - DELETE
  - Manipulation des données directement via des URIs
  - Services sans états uniquement
    - Toutes les informations nécessaires sont transmises dans les requêtes
  - La sécurité repose intégralement sur HTTPS

L'architecture REST

- Une application RESTfull présente les caractéristiques suivantes
  - Méthode POST pour création de ressources (C)
  - Méthode GET pour la consultation de ressources (R)
  - Méthode PUT pour la mise à jour de ressources (U)
  - Méthode DELETE pour la suppression de ressources (D)
- En pratique les services REST peuvent proposer d'autres méthodes
  - Les services respectant les caractéristiques précédentes sont appelés REST CRUD
  - Ils s'appuient sur tous les messages possibles en HTTP
- Dans une architecture client-serveur
  - La méthode GET est associée au client
  - Les méthodes PUT et POST au serveur

L'architecture REST

- Une ressource est représentée par un document XML
  - La création d'une ressource sera donc la combinaison d'une requête POST et de la représentation XML de cette ressource
  - Cette représentation permet de maintenir un état de la ressource
- Les ressources sont adressées à travers des URIs pour toutes les opérations
  - Une ressource peut posséder plusieurs URIs (mais une URI pour une ressource)
  - Les URIs doivent être le plus explicites possibles
  - Le format standard d'une URI est: http://host:port/path?queryString#fragment
  - Exemple d'URI: https://www.flickr.com/explore/2015/11/28
- La représentation des ressources est libre (XML, JSON, CSV, JPEG, HTML…)
  - Elle est spécifiée dans l'en-tête HTTP (text/xml, text/json, text/plain…)
  - Les formats utilisés peuvent être différents pour les requêtes et les réponses

HATEOAS

- HATEOAS (Hypermedia as the Engine of Application State) est un principe de fonctionnement qui repose sur l'idée qu'une application web est composée d'un ensemble de pages reliés par des liens
  - Ressemble à une machine à états (State machine)
- L'idée d'HATEOAS est d'exposer les différents états de l'application RESTfull à travers des liens
  - A partir de la page d'accueil on obtient dans la réponse, les liens vers les autres pages/ressources
  - Cela consiste à inclure dans chaque réponse du serveur, un lien vers les prochaines requêtes qui peuvent intervenir et vers les autres ressources
    - Cela évite au client de devoir recomposer les URLs
    - Cela permet de changer les URLs coté serveur de manière transparente pour le client

HATEOAS

- Exemple

```
[
    {
        "uid": 1,
        "age": 18,
        "name": "John Doe",
        "order": 1
    },
    {
        "uid": 2,
        "age": 29,
        "name": "Bob Marley",
        "order": 2
    }
]
```

Non HATEOAS

```
[
    {
        "uid": 1,
        "age": 18,
        "name": "John Doe",
        "order": 1,
        "href": "/user/1"
    },
    {
        "uid": 2,
        "age": 29,
        "name": "Bob Marley",
        "order": 2,
        "href" : "/user/2"
    }
]
```

HATEOAS

Le format JSON (Javascript Object Notation)

- JSON fut initialement crée pour la sérialisation et l'échange d'objet Javascript
  - C'est un langage pour l'échange de données semi structurées
  - C'est un format textuel indépendant du langage de programmation utilisé à la manière de XML
  - Il est définit par l'IETF dans la RFC 7159

- Actuellement, JSON est la notation la plus utilisée pour l'échange de données
  - Dans un environnement Web avec les services REST
  - Pour la persistance et le stockage des données

- JSON repose sur une notation très simple articulée autour de l'association clé-valeur

```
"nom" : "Conchon"
```


Le format JSON (Javascript Object Notation)

- JSON distingue deux types de valeurs
  - Les valeurs atomiques
    - Chaines de caractères ("chaine") comme dans l'exemple précédent
    - Entiers, réels
    - Booléens
  - Les valeurs complexes
    - Permettent de définir des objets `{"nom" : "Conchon", "prenom" : "Emmanuel"}`
      - ★ *Un objet est constitué d'un ensemble d'associations clé-valeur séparées par des virgules*
        - Les clés doivent être uniques
        - Elles font références aux attributs de l'objet
      - ★ *Un objet peut être définit comme une valeur dans une association*

```
"personne":{
    "nom" : "Conchon",
    "prenom" : "Emmanuel"
}
```


Le format JSON (Javascript Object Notation)

- Il est également possible de définir des tableaux
  - ★ *Un tableau est une liste de valeurs pouvant être de types différents ou de même type*

```
"Acteur principaux":["Fisher", "Ford", "Hamill?"]
```

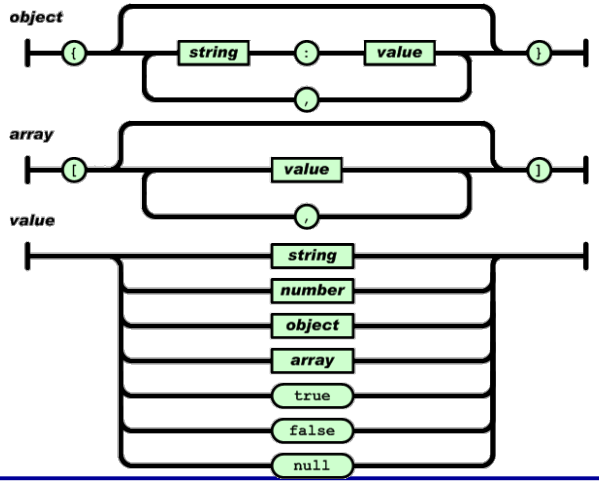
  - ★ *Il est possible de faire des tableaux de tableaux, des tableaux d'objets ….*
    - Imbrication sans limite à la manière de XML !

- Un document JSON fait référence à un objet JSON complexe

```
{
    "Titre" : "Le Réveil de la Force",
    "Synopsis" : "Dans une galaxie lointaine, très lointaine, un nouvel épisode \n
                  de la saga \"Star Wars\", 30 ans après les événements \n
                  du \"Retour du Jedi\".",
    "Date de sortie" : {"année" : 2015, "mois" : 12, "jours" : 16},
    "Réalisateur" : {"nom" : "Abrams", "prenom" : "Jeffrey Jacob"},
    "Acteur principaux":[
        {"nom" : "Fisher", "prenom" : "Carrie"},
        {"nom" : "Ford", "prenom" : "Harrisson"},
        {"nom" : "Hamill", "prenom" : "Mark"}
    ]
}
```


Le format JSON (Javascript Object Notation)

- Résumé graphique sur le langage (source: http://www.json.org)



Le format JSON (Javascript Object Notation)

- Au final JSON est très proche d'un format XML mais
  - Il est plus léger
  - Il est plus intuitif
  - Il est plus simple à analyser (parser)
  - Il ne possède pas de spécification standard pour associer un schéma à un document
    - Mais ça arrive avec JSON-Schema basé sur XSD
      - ★ *Actuellement au stade d'Internet Draft*
  - Il ne possède pas de langage de requête associé

REST & Android

- Plusieurs librairies permettent de simplifier les interactions avec les services REST parmi lesquelles
  - GSON
    - Permet de simplifier la manipulation d'objets JSON
  - RetroFit
    - Simplifie les requêtes REST
- Ces deux librairies bien que très utilisées ne font pas partie de la distribution Android, il faut donc les ajouter à votre projet à l'aide de gradle
  - Dans le fichier build.gradle sous Android studio

```
dependencies {
    compile 'com.google.code.gson:gson:2.6.2'
    compile 'com.squareup.retrofit2:retrofit:2.1.0'
    compile 'com.squareup.retrofit2:converter-gson:2.1.0'

}
```

- Elles nécessitent également la permission INTERNET

REST & Android

- La première étape pour pouvoir utiliser un service REST consiste à obtenir les POJOs qui seront fournis par le service
  - Ces objets sont les réponses du service REST
  - Il faut donc les annoter pour faire correspondre les noms de champs avec les noms renvoyés dans le JSON
- Il est également nécessaire d'obtenir une interface de communication avec le service
  - Elle va fournir une gestion des URLs (GET, POST etc.)
- Il faut bien sur également s'appuyer sur un adaptateur REST
  - Dans notre cas, il s'agira de Gson

REST & Android

- Pour construire le POJO le plus simple consiste à passer par jsonschema2pojo http://www.jsonschema2pojo.org/
- Exemple:
  - Prenons le document JSON https://api.github.com/users/fabpot

- L'interface peut être obtenue de manière assez simple à partir de l'API
  - L'url nous donne la structure générale
  - Dans notre exemple:
    - /users/{user} nous permet d'obtenir des informations sur un utilisateur en GET
    - Le user est réutilisable grâce à l'annotation @Path("user") en tant que paramètre dans une fonction

```
public interface GitAPI {

    @GET("/users/{user}")
    Call<GitUserModel> getFeed(@Path("user") String user);

    String BASE_URL = "https://api.github.com";
    Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(BASE_URL)
            .addConverterFactory(GsonConverterFactory.create())
            .build();
}
```


- Dans l'activité principale, il ne reste plus alors qu'à utiliser le service

```
GitAPI gitHubService = GitAPI.retrofit.create(GitAPI.class);
final Call<GitUserModel> call =
        gitHubService.getFeed("fabpot");

call.enqueue(new Callback<GitUserModel>() {
    @Override
    public void onResponse(Call<GitUserModel> call, Response<GitUserModel> response) {
        final TextView textView = (TextView) findViewById(R.id.textView);
        textView.setText(response.body().toString());
    }
    @Override
    public void onFailure(Call<GitUserModel> call, Throwable t) {
        final TextView textView = (TextView) findViewById(R.id.textView);
        textView.setText("Something went wrong: " + t.getMessage());
    }
});
```


- Dans les exemples précédent, le service REST ne nécessite pas d'authentificaiton
- Suivant les APIs plusieurs solutions d'authentification sont en général possible
  - Login/mdp
  - API Key
  - Oauth
  - …
- Retrofit permet de prendre en charge tous ces modes
  - Nécessite d'ajouter des paramètres aux requêtes
  - Passage par un Interceptor

- Le principe d'un Interceptor comme son nom l'indique est d'intercepter toutes les requêtes HTTP et de les modifier le cas échéant
  - Le cas le plus courant dans les API REST est d'ajouter une clé d'API
  - Pour cela, RetroFit s'appuie sur OkHttp

```
OkHttpClient okHttpClient = new OkHttpClient.Builder().addInterceptor(new Interceptor()
{
    @Override
    public Response intercept(Chain chain) throws IOException {
        Request request = chain.request();
        HttpUrl url = request.url().newBuilder().addQueryParameter(
MovieDbApi.PARAM_API_KEY, BuildConfig.MOVIE_DB_API_KEY).build();
        request = request.newBuilder().url(url).build();
        return chain.proceed(request);
    }
}).build();
```

```
Retrofit retrofit = new Retrofit.Builder()
    .client(okHttpClient)
    .baseUrl(MovieDbApi.END_POINT)
    .build();

movieDbApi = retrofit.create(MovieDbApi.class);
```

Source: http://www.fasteque.com/tips-on-updating-to-retrofit-2/