Andr0

Zeste de savoir

Communication entre Android et PHP/MySQL

12 février 2021

Table des matières

Le contenu de ce tutoriel est devenu obsolète et devrait être archivée et/ou supprimée.

De nombreux débutants ne savent pas comment faire communiquer une base de données distante MySQL avec une application cliente du type Android. Ce tutoriel tente de répondre à cette question où la réponse est bien loin d'être simple. Des réponses à des questions transverses sont nécessaires et c'est facile de tomber dans certains pièges quand on ne dispose pas de l'expérience requise dans le domaine.

L'expérience requise, parlons-en! Ce tutoriel se veut accessible aux plus débutants mais se concentre en grande partie sur le développement Android et légèrement sur le développement serveur. D'ailleurs, il n'est pas conseillé d'adopter les exemples de la partie serveur de ce tutoriel dans un environnement de production pour des raisons de maintenance et de qualité de code. Cette partie serveur tente simplement de vous expliquer les grandes lignes de son fonctionnement et donne un exemple fonctionnel. Les bonnes pratiques sont mentionnées et expliquées, mais elles ne sont pas enseignées. Cet enseignement demanderait un cours entier sur le sujet où d'autres ressources sur Internet les expliqueront bien mieux.

Les connaissances indispensables sont les suivantes:

— Savoir créer un projet Android et l'exécuter: ressource écrite ou ressource multimédia .
— Savoir créer une base de données MySQL: ressource écrite .
— Savoir exécuter du code PHP: ressource écrite pour les utilisateurs Windows ou ressource écrite pour tous les utilisateurs .

Les connaissances optionnelles sont les suivantes:

— Avoir une connaissance basique des communications entre un serveur et des clients.
— Savoir faire des requêtes HTTP depuis une application Android.
— Lier un serveur applicatif en PHP avec une base de données MySQL.

Ces dernières connaissances sont optionnelles parce qu'elles seront expliquées dans ce tutoriel pour que vous ne soyez pas perdu. Si vous ne disposez pas de ces connaissances, aucun inquiétude à avoir. Vous pourrez quand même suivre ce tutoriel et disposez des même connaissances qu'un autre lecteur disposant de ces connaissances.

Sur ce, ne perdons plus de temps et lançons nous dans cette communication entre une application Android et un serveur applicatif PHP/MySQL!

1. Pourquoi utiliser un serveur ?

Une architecture client/serveur est un mode de communication un peu obscur pour les débutants, à juste titre. En effet, à partir du moment où vos applications ont besoin d'un serveur, elles commencent à prendre de l'importance. Il faut y centraliser les données pour permettre à d'autres clients de les récupérer. Ce genre d'architecture est maintenant de plus en plus courante et donc, de plus en plus rare de fonctionner uniquement avec des bases de données locales.

1.1. Un serveur est nécessaire à partir de quand ?

Le principal objectif d'un serveur est la centralisation des données et permettre à plusieurs clients de les récupérer et d'y contribuer. Plusieurs scénarios existent pour vous poser la question de sa nécessité:

— Développez-vous une application sur plusieurs plateformes (Android, iOS, Windows Phone, site web, etc.) avec les mêmes données? Par exemple, si l'objectif de l'application est la gestion d'une liste de tâches à faire (couramment appelée une *todo list*), il serait dommage de ne pas partager ces listes sur toutes les plateformes à travers les applications, mobiles ou non, que vous avez développées. Si une tâche est rajoutée sur l'application Android, elle doit être rajoutée sur le site web.
— Désirez-vous "ouvrir" certaines de vos données à des services tiers? C'est une pratique courante. Par exemple, la plateforme Zeste de Savoir enregistre chaque jour de plus en plus de données, la plupart sont des données publiques et/ou sous une licence libre. Zeste de Savoir a mit en place une API (expliquée plus loin dans ce cours) qui permet son utilisation dans des applications tierces n'ayant pas forcément un rapport avec Zeste de Savoir.

Il existe d'autant plus de scénarios quand l'on se trouve dans le cadre professionnel. Les entreprises ont des besoins et des contraintes hautement plus complexes avec des architectures précises. Elles sont souvent murement réfléchies par ses dirigeants et/ou ses employés.

1.2. Qu'est-ce qu'une API ?

Une API, ou *Application Programming Interface*, est, comme son nom l'indique, une **interface** d'un programme applicatif informatique. Elle permet d'offrir des **services** et des **ressources** sur base d'un système déjà existant. En fait, ces ressources sont des données exposées selon certaines conditions définies par vos soins. Vous restez maitre de vos données et appliquez la politique souhaitée pour définir les données publiques ou privées.

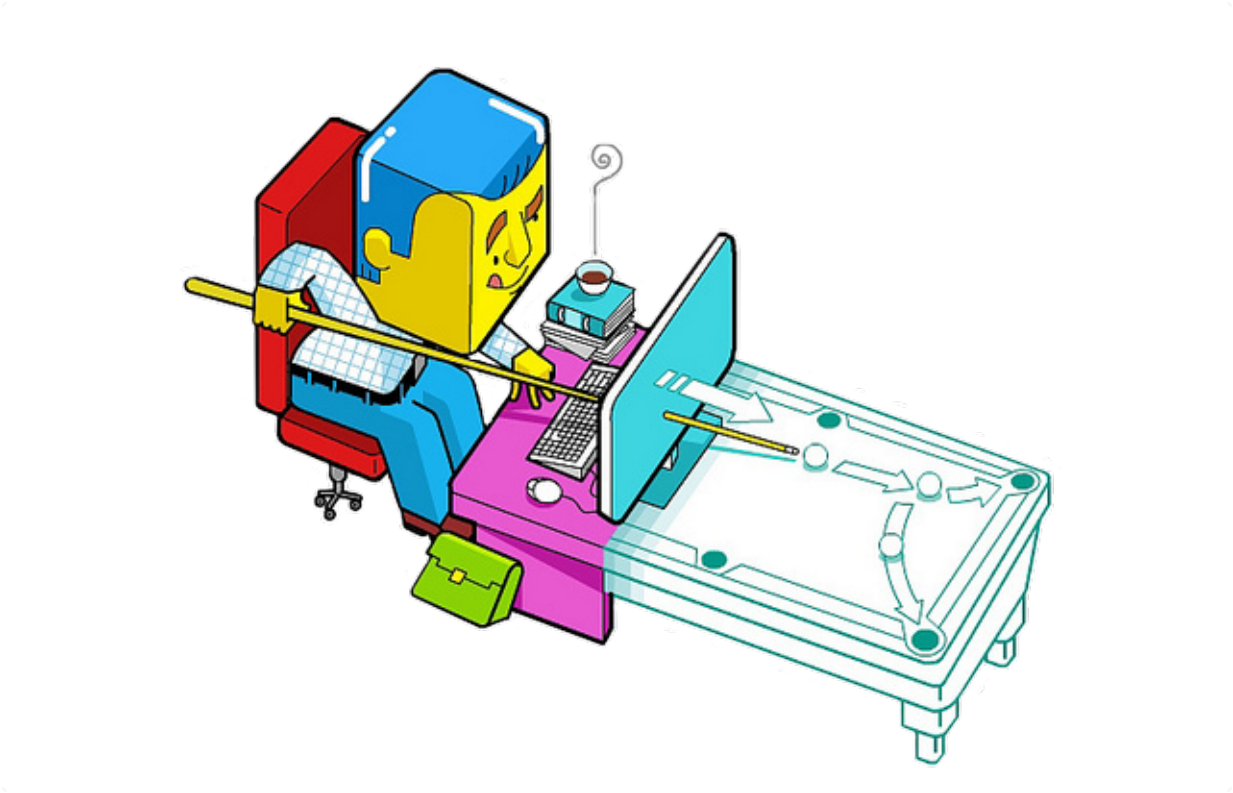

FIGURE 1.1. – Les services sur Internet sont accessibles par des interfaces quand vous utilisez votre ordinateur. Crédit: Eran Mendel, tukinon sur Flickr.

Cette pratique n'est pas obligatoire. Pour le projet illustrant ce tutoriel, il n'y aura pas d'utilisation d'une API, malheureusement, puisque nous resterons simple du côté serveur. Ce n'est pas l'objectif de notre cours mais c'est une pratique sur laquelle je vous encourage à vous renseigner. Sachez tout de même qu'il existe 2 grandes familles dans ce domaine:

— REST : Architecture orientée ressource, une API REST offre une interface simple et uniforme. Elle est particulièrement bien adaptée au web mais elle n'en est pas dépendante. Sa souplesse et sa mise en place rapide est une très grande force par rapport à son concurrent direct.
— SOAP: Permet la transmission de messages entre des objets distants. Il est alors possible à un objet d'appeler directement des méthodes sur un serveur. Cependant, il est bâti sur l'XML et moins sur le JSON, "nouveau" format de données particulièrement utilisé par REST.

Pour un débutant, le REST me semble tout indiqué et les ressources (française et anglaise) regorgent sur le sujet sur la Toile. Soyez curieux!

1.3. Fonctionnement

Les explications qui vont suivre sont largement simplifiées pour être accessibles aux débutants mais elles n'en restent pas moins véridiques et suffisantes pour la compréhension de ce qui va suivre dans ce tutoriel.

FIGURE 1.2. – Fonctionnement d'une architecture client-serveur

Comme vous pouvez le constater, nos clients (les ordinateurs, les smartphones, les tablettes, etc.) font des requêtes HTTP (les flèches pleines) vers des serveur en passant par la Toile, Internet. Ces requêtes n'ont rien de plus spécial qu'une requête HTTP que vous feriez à partir d'un navigateur web. Vous spécifiez une URL et vous vous attendez à recevoir une réponse (les flèches vides).

Une fois que la requête est arrivée à destination, il passe d'abord par l'API (ce n'est pas obligatoire, mais soyons complets dans notre exemple). Cette API possède une procédure qui associera la forme de votre URL à une action à réaliser (gestion d'une ressource dans REST, opération métier dans SOAP). L'API va alors l'exécuter et dialoguer avec le serveur pour récupérer les données. Elle récupère le résultat et le formate dans un format de données comme l'XML, le JSON ou l'HTML.

Notre client reçoit alors la réponse à sa requête et peut la traiter comme bon lui semble. Si nous sommes sur un navigateur, nous obtiendrons une réponse HTML que nous pourrons afficher à l'écran (parce que nous avons spécifié dans la requête que nous désirons une réponse en HTML). Si nous sommes sur des appareils mobiles, nous pourrons obtenir une réponse XML ou JSON pour pouvoir utiliser les données comme nous le désirons. Nous pourrions demander au serveur une réponse HTML mais cela ne serait pas forcément pertinent pour un client mobile.

Grâce à ce procédé, le serveur permet une totale **abstraction de son langage**[1], de son implémentation et n'oblige pas le client à utiliser un langage spécifique pour pouvoir dialoguer avec lui. Ce qui est le cas avec des technologies concurrentes du REST.

1.4. Le choix entre un serveur dédié ou une plateforme de conception

Il existe plusieurs solutions pour développer sa propre application serveur. Il ne sera pas expliqué comment installer et configurer un serveur. Cependant, nous verrons les composants d'un serveur qu'il faut obligatoirement installer pour posséder un serveur simple mais fonctionnel. Cela grâce à différentes technologies possibles.

1. Une abstraction de langage, *language agnostic* en anglais, permet une communication entre deux langages différents qui ne sont pas forcément compatibles, cela grâce à un troisième langage destiné à faire le lien entre les deux.

1.4.1. Une base de données

Toutes vos données sont contenues dans une base de données. Il existe plusieurs types de SGBD (Système de Gestion de Base de Données) et elles ont toutes leurs subtilités. Libre à vous de choisir celle que vous préférez. Vous pouvez donc utiliser SQLite, MySQL, PostgreSQL, Oracle, etc. Cependant, suivant ce que vous allez choisir pour développer votre application serveur, vous devez vous renseigner si le langage supporte le SGBD que vous avez choisi. Pour les plus connus, cela ne devrait pas poser problème mais pour les moins, il est toujours utile de vérifier.

1.4.2. Applicatif

1.4.2.1. PHP Les applications serveur peuvent être développées dans de nombreux langages, y compris par des langages utilisés pour développer des sites web en temps normal comme le PHP. Dans le cadre de ce tutoriel, nous allons utiliser ce langage pour fournir une application serveur fonctionnelle rapidement et facilement compréhensible par les débutants.

Le langage a beaucoup évolué et des frameworks comme Symfony 2 sont apparus. Avec eux, des architectures plus évoluées qui simplifient le développement d'applications serveur actuelles. Pour plus d'information sur Symfony 2 et sur le développement de routes permettant de fournir une API donnant accès aux données, rendez-vous sur ce tutoriel .

1.4.2.2. Java Le java est l'un des langages les plus utilisés pour le développement d'applications serveur via des EJB (Entreprise JavaBeans). EJB est une architecture de composants logiciels côté serveur pour la plateforme de développement J2EE. Grâce à un **serveur d'application** comme JBoss, Glassfish et d'autres, il est possible de déployer des applications serveur définissant des routes pour accéder aux différentes ressources que vous désirez offrir.

Cependant, ces solutions sont plutôt lourdes et ne conviennent pas aux petits projets. C'est pourquoi ces différentes technologies sont très demandés en entreprise pour satisfaire leurs besoins en performance et en maintenance contrairement aux petits sites web amateurs.

1.4.2.3. Autres Bien sûr, il existe beaucoup d'autres langages qui permettent de développer des applications serveur comme NodeJS, C#, Scala, Python, etc.

1.4.3. Utiliser une plateforme de conception

Vous n'êtes pas obligé de partir d'une feuille blanche. Il existe des solutions qui vous permettent de démarrer à partir d'un serveur pré-configuré où toute une série de services d'un serveur sont abstraits. Des outils sont mis à la disposition des développeurs pour une meilleure gestion et une maintenance.

L'un des plus connus est Google App Engine . Grâce à une simple application Web en Python, Java ou Go, vous pouvez confectionner rapidement un serveur que vous pourrez utiliser en développement ou en production. Il faut tout de même savoir que ce genre de solutions sont souvent payantes, Google App Engine n'échappe pas à cette règle.

2. Mise en place du serveur applicatif PHP

Dans un souci de simplicité, le serveur applicatif sera développé par des scripts PHP; c'est-à-dire que nous ne développerons pas avec un framework ou une solution toute faite pour faciliter le développement d'une API. Cela nécessiterait l'écriture d'un tutoriel à part entière. D'autres sont déjà rédigés sur la plateforme Zeste de Savoir. Ce qui sera enseigné ici n'est que le strict minimum pour pouvoir être utilisable depuis une application mobile Android.

2.1. Configuration de la base de données

L'exemple est extrêmement simple. Le plus important ne concerne pas les données ni la manière dont elles sont envoyées mais comment les récupérer et les traiter. C'est pourquoi nous allons limiter notre applicatif à une seule URL qui délivrera une liste de produits dans un format JSON (JSON étant un concept qui sera abordé plus loin dans ce tutoriel).

2.1.1. Pré-requis

Nous devons installer quelques outils et savoir utiliser quelques concepts hors du développement Android. Je n'expliquerais pas comment les installer ni comment les utiliser. Il existe d'autres tutoriels qui vous expliqueront comment faire bien mieux que moi. Je vous renvoie aux chapitres des tutoriels suivants:

- Le chapitre Préparer son ordinateur du tutoriel Concevez votre site web avec PHP et MySQL de Mateo21 pour installer votre serveur PHP.
- Le chapitre Présentation des bases de données du tutoriel Concevez votre site web avec PHP et MySQL de Mateo21 si vous voulez bien comprendre comment lier une application PHP à une base de données MySQL.
- Le chapitre phpMyAdmin du tutoriel Concevez votre site web avec PHP et MySQL de Mateo21 pour utiliser correctement le gestionnaire de bases de données de MySQL.
- Le chapitre Création d'une base de données du tutoriel Administrez vos bases de données avec MySQL de Taguan pour créer une base de données dans PhpMyAdmin.
- Le chapitre Création de tables du tutoriel Administrez vos bases de données avec MySQL de Taguan pour créer facilement une table dans votre base de données.
- Le chapitre Insertion de données du tutoriel Administrez vos bases de données avec MySQL de Taguan pour insérer des valeurs dans votre table.

Ne prenez pas peur! Même si cela fait beaucoup de chapitres, d'une part, je suis certain que la plupart d'entre vous connaissent déjà toutes ces notions (même vaguement) et d'autre part, il n'est pas nécessaire d'être un expert dans ces domaines. Une connaissance basique est suffisante pour ne pas être perdu dans la suite des explications de ce tutoriel.

2.1.2. Configuration de la table

Vous devriez avoir tous les outils et tous les concepts pour installer et configurer correctement votre serveur PHP. Pour pouvoir suivre sereinement et efficacement les exemples qui suivent, voici la spécification de l'unique table de notre base de données:

Libre à vous de choisir une spécification différente mais il faudra la répercuter dans le code de votre applicatif PHP et Android par la suite.

Concept	Nom
Base de données	AdvancedAndroidDevelopment
Table	product
Colonnes	— ID — NAME — TYPE — PRICE

Pour vous faciliter la vie, les requêtes SQL ci-dessous vous permettront de créer la table ”product” et d’insérer quelques données factices que nous allons récupérer dans notre application Android dans la prochaine section.

```sql
--
-- Structure de la table `product`
--

CREATE TABLE IF NOT EXISTS `product` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `NAME` varchar(50) NOT NULL,
  `TYPE` varchar(50) NOT NULL,
  `PRICE` double NOT NULL,
  PRIMARY KEY (`ID`)
) ;

--
-- Contenu de la table `product`
--

INSERT INTO `product` (`ID`, `NAME`, `TYPE`, `PRICE`) VALUES
(1, 'Apple', 'Fruit', 1.2),
(2, 'Pear', 'Fruit', 1.3),
(3, 'Banana', 'Fruit', 1.1),
(4, 'Mushroom', 'Vegetable', 2.7),
(5, 'Pepper', 'Vegetable', 3.6);
```

2.2. Qu’est-ce qu’une réponse en JSON ?

Comme dit précédemment, notre serveur ne sera pas développé avec une API. À la place, l’URL exposée pointera vers un script PHP qui se chargera d’effectuer la requête SQL adéquate pour

récupérer les produits en base de données et de les renvoyer dans le format JSON.

Si vous ne connaissez pas le JSON, c'est une bonne occasion d'en savoir un peu plus. Si vous êtes déjà un développeur Android, vous connaissez bien l'XML puisqu'il s'agit du langage de balisage utilisé pour la confection des interfaces mais le JSON est différent dans sa syntaxe et dans sa philosophie.

2.2.1. Définition

Voici une courte définition que vous retrouvez sur le site officiel de JSON (dont il existe une version française):

> JSON (JavaScript Object Notation – Notation Objet issue de JavaScript) est un format léger d'échange de données. Il est facile à lire ou à écrire pour des humains. Il est aisément analysable ou générable par des machines. Il est basé sur un sous-ensemble du langage de programmation JavaScript (JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999). JSON est un format texte complètement indépendant de tout langage, mais les conventions qu'il utilise seront familières à tout programmeur habitué aux langages descendant du C, comme par exemple: C lui-même, C++, C#, Java, JavaScript, Perl, Python et bien d'autres. Ces propriétés font de JSON un langage d'échange de données idéal.
>
> *json.org*

Comme la définition le mentionne, le principal avantage de ce format texte est sa totale indépendance du langage. C'est pourquoi, même si notre application serveur est développée en PHP, nous pouvons sans problème développer nos clients dans un autre langage (dans notre cas, en Java puisque nous développons sur Android). Ce n'est pas le seul avantage de ce format. Nous pouvons en citer d'autres depuis cet extrait:

- Sa simplicité dans sa mise en oeuvre. Aujourd'hui, tous les langages supportent ce format et proposent des solutions pour construire et l'utiliser.
- Sa lisibilité aussi bien par les humains que par les machines puisque le langage est peu verbeux.
- Sa syntaxe est réduite et non extensible ce qui le rend facile à apprendre pour l'humain.

2.2.2. Les différentes structures

Les structures d'un fichier JSON sont extrêmement simples et se représentent que de 3 manières différentes:

- Objet: Fonctionne sur le principe de clé valeur qu'il est possible de déclarer plusieurs fois les uns à la suite des autres.
- Tableau: Représente un tableau de valeurs qui peuvent être aussi bien des objets, des tableaux ou des valeurs simples.
- Valeur: Représente simplement une valeur qui peut être une chaîne de caractères, un nombre, un booléen, une valeur null ou même un objet ou un tableau. Raison pour laquelle un tableau peut contenir des objets et des tableaux et réciproquement pour un objet.

Ainsi, toutes ses représentations se schématisent de la manière suivante avec cette image issue du site officiel du JSON .

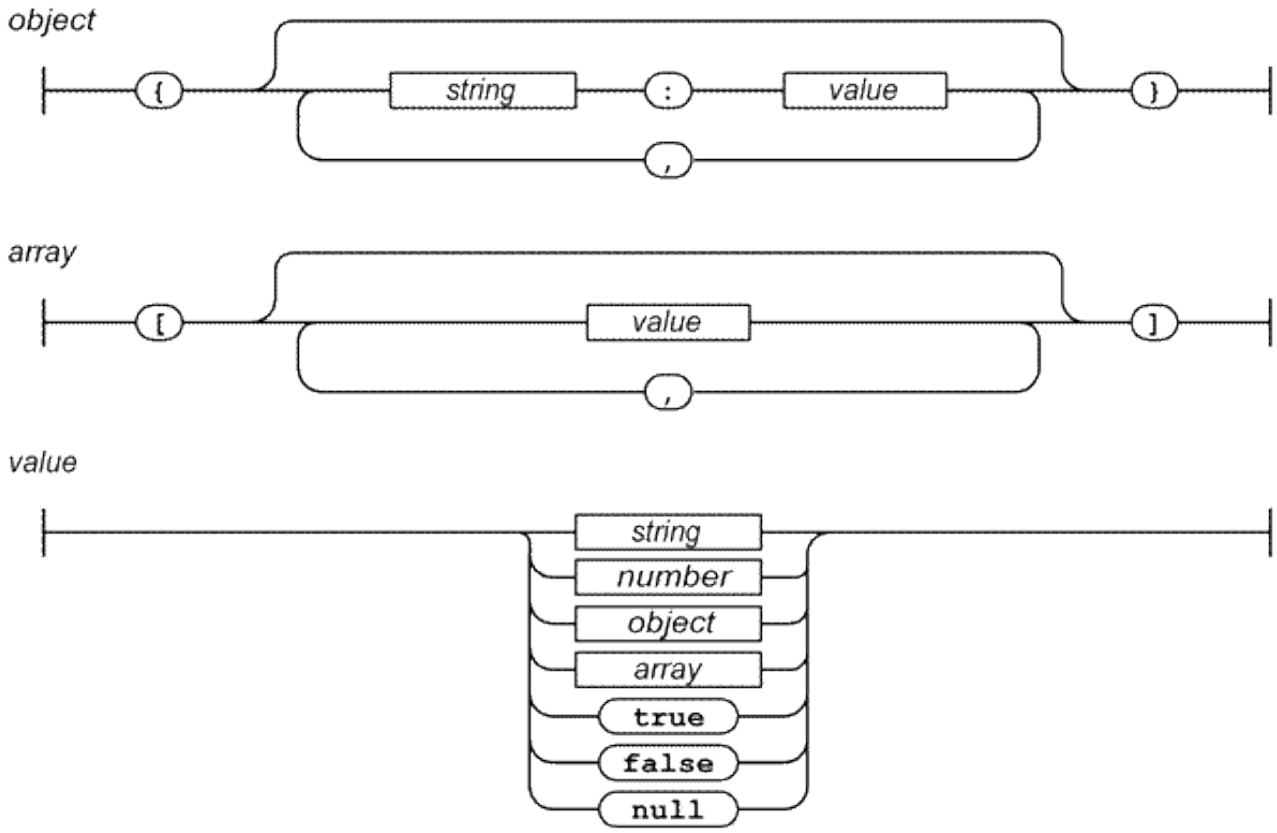

FIGURE 2.3. – Description de son langage dans son intégralité

La lecture de ce genre de représentation est enfantine, il suffit de suivre les branches. Pour vous aider à comprendre, prenons le schéma fléché ci-dessous. Les flèches mettent en évidence les branchements possibles et les arrêts à une valeur obligatoire.

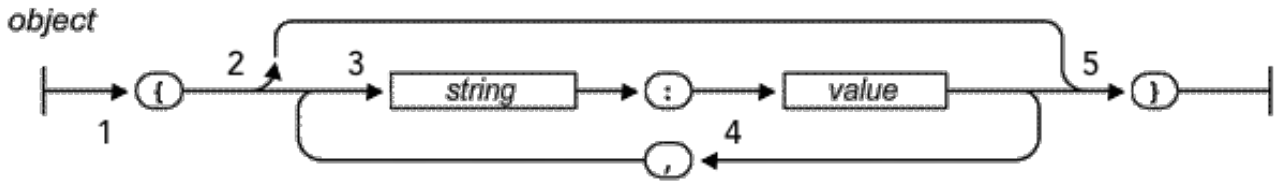

FIGURE 2.4. – Ecriture d'un objet à partir de la représentation du JSON

1. Nous commençons notre objet et nous sommes obligés d'écrire une accolade. C'est cette accolade qui va marquer le début d'un objet JSON.

2. Si nous prenons cette branche et que nous la suivons, nous évitons de renseigner des clés/valeurs dans notre objet pour directement le fermer. Cela veut donc dire qu'il est possible de créer des objets vides mais que à la suite d'une déclaration d'un objet. Dès que vous aurez renseigné au moins une clé/valeur, votre objet JSON ne pourra plus être vide, chose assez logique.
3. Nous décidons de renseigner une clé/valeur dans notre objet. Nous rencontrons un encadré qui représentera notre clé, nous devons renseigner ":" pour faire correspondre la clé à la valeur et nous renseignons la valeur.
4. Nous pouvons réitérer la branche 3 autant de fois que nous le désirons. Pour ce faire, vous prenez la branche 4 qui vous oblige à renseigner une virgule avant de retourner sur la branche précédente.
5. Dès que vous avez fini de renseigner toutes les clés/valeurs de votre objet, vous sortez avec la branche 5 en renseignant une accolade fermante.

Voyons un exemple concret avec le développement du serveur applicatif.

2.3. Développement du serveur applicatif

2.3.1. Script de l'application serveur

Vous avez toutes les cartes en main pour développer votre application serveur. Il ne vous reste plus qu'à développer votre script qui va:

— Se connecter à votre base de données, AdvancedAndroidDevelopment;
— Exécuter la requête SQL de sélection de votre choix pour récupérer les produits;
— Former un tableau d'objets pour le préparer au format JSON;
— Utiliser la méthode `string json_encode ( mixed $value [, int $options = 0 ] )` pour encoder votre tableau en JSON (documentation officielle);
— Utiliser la méthode `void echo ( string $arg1 [, string $... ] )` pour envoyer le fichier JSON au client (documentation officielle).

Vous savez quoi faire et je vais vous laisser faire. Il est important que vous y parveniez seul pour pouvoir développer vos propres scripts. Notre exemple est assez simple et ne devrait pas poser problème mais si vous éprouvez des difficultés, rendez-vous sur des tutoriels PHP pour vous débloquer. Essayez de trouver la solution par vous-même avant de regarder la solution proposée.

À l'exécution du script, le JSON que vous devez obtenir est un tableau avec pour chaque indice, un objet qui représentera un produit:

```
[
        {
                "id" : "1",
                "name" : "Apple",
                "type" : "Fruit",
                "price" : "1.2"
        } ,
```

```
        {
                "id" : "2",
                "name" : "Pear",
                "type" : "Fruit",
                "price" : "1.3"
        } ,
        {
                "id" : "3",
                "name" : "Banana",
                "type" : "Fruit",
                "price" : "1.1"
        } ,
        {
                "id" : "4",
                "name" : "Mushroom",
                "type" : "Vegetable",
                "price" : "2.7"
        } ,
        {
                "id" : "5",
                "name" : "Pepper",
                "type" : "Vegetable",
                "price" : "3.6"
        }
]
```

2.3.2. Solution

```php
<?php
    try {
        // connection to the database.
        $pdo_options[PDO::ATTR_ERRMODE] = PDO::ERRMODE_EXCEPTION;
        $bdd = new
            PDO('mysql:host=localhost;dbname=AdvancedAndroidDevelopment',
            'root', 'root', $pdo_options);

        // Execute SQL request on the database.
        $sql = 'SELECT id, name, type, price FROM product;';
        $response = $bdd->query($sql);
        $output = $response->fetchAll(PDO::FETCH_ASSOC));
    } catch (Exception $e) {
        die('Erreur : ' . $e->getMessage());
    }

    // Print JSON encode of the array.
    echo(json_encode($output));
```

```
17 ?>
```

3. Récupérer les données distantes

3.1. Une requête HTTP standard

Parmi toutes les API Java intégrées dans le framework Android, il existe le paquetage `java.net` pour effectuer des requêtes HTTP standard; c'est-à-dire sans l'intervention d'une bibliothèque externe.

L'usage des classes de ce paquetage implique la configuration totale de vos requêtes HTTP de la méthode utilisée, aux différents *timeout* et nécessite la conversion manuelle d'un flux de données à une chaine de caractère.

Pour résumé, si nous désirons effectuer une requête HTTP du type `GET`, avec une limite de 10000 millisecondes pour lire la requête et de 15000 pour se connecter au serveur, vous devez initialiser la requête de la manière suivante:

```
1  public String get(String url) throws IOException {
2    InputStream is = null;
3    try {
4      final HttpURLConnection conn = (HttpURLConnection) new
           URL(url).openConnection();
5      conn.setReadTimeout(10000 /* milliseconds */);
6      conn.setConnectTimeout(15000 /* milliseconds */);
7      conn.setRequestMethod("GET");
8      conn.setDoInput(true);
9      // Starts the query
10     conn.connect();
11     is = conn.getInputStream();
12     // Read the InputStream and save it in a string
13     return readIt(is);
14   } finally {
15     // Makes sure that the InputStream is closed after the app is
16     // finished using it.
17     if (is != null) {
18       is.close();
19     }
20   }
21 }
```

Code: Requête HTTP vers un serveur à une URL donnée en paramètre.

À la lecture de ce code, cela peut vous sembler pas bien complexe à utiliser mais sachez qu'il s'agit ici du minimum syndical (et encore, il n'est pas nécessaire de renseigner des *timeout*). Dès lors que vous voudrez effectuer des requêtes dans un projet sérieux, il vous faudra réagir

à des codes HTTP et traiter son contenu en fonction de ce même code. Chose qui n'est pas traitée dans cet exemple, ni même dans ce tutoriel puisque nous verrons plus loin des usages plus courants et plus simples pour effectuer des requêtes.

Vous remarquez aussi l'usage de la méthode `readIt(InputStream)` qui permet de lire un `InputStream` et d'en sauvegarder son contenu dans une `String`. Cette méthode n'existe pas dans Android mais il existe plusieurs façons de mettre en oeuvre cette conversion. Voici la mienne:

```
1 private String readIt(InputStream is) throws IOException {
2   BufferedReader r = new BufferedReader(new InputStreamReader(is));
3   StringBuilder response = new StringBuilder();
4   String line;
5   while ((line = r.readLine()) != null) {
6     response.append(line).append('\n');
7   }
8   return response.toString();
9 }
```

Code: Lit un flux de données et place son contenu dans une chaine de caractères.

Cette section n'est ici qu'à titre informatif et parce que vous pourriez rencontrer ce genre de code dans d'autres projets puisqu'il s'agit de la version la plus basique possible pour faire une requête.

Au risque de gêler votre interface graphique ou de faire survenir un crash, il faut **obligatoirement** faire une requête HTTP dans un thread secondaire. Vous disposez de nombreuses possibilités pour exécuter du traitement dans des threads secondaires comme les `AsyncTask`, les `Handler`, les `HandlerThread`, les `Loader` ou les services. Ce sujet fera l'objet d'un tutoriel séparé vu la diversité des possibilités et leurs nombreux avantages/inconvénients.

3.2. Extraire les données du JSON

Rappelez-vous, le serveur applicatif va chercher en base de données une liste de produits et renvoyer le résultat dans un fichier JSON. Il faut donc pouvoir extraire les données de ce fichier et les placer dans une classe pour pouvoir les afficher ou les manipuler dans l'application. Pour y parvenir, nous allons déléguer la responsabilité d'extraire les données à une classe `Product`. Cela permettra de ne plus se préoccuper de l'extraction des "objets" d'un produit dans le fichier JSON.

Avant cela, nous allons en apprendre plus sur les classes principales, à savoir `JSONArray` et `JSONObject`.

3.2.1. Extraire un tableau

Pour extraire un tableau d'un fichier JSON, vous devez utiliser la classe `JSONArray` qui possède 4 constructeurs:

- `public JSONArray ()`: Création d'un tableau JSON vide.
- `public JSONArray (Collection copyFrom)`: Création d'un tableau JSON en copiant toutes les valeurs de la collection donnée.
- `public JSONArray (JSONTokener readFrom)`: Création d'un tableau JSON en copiant toutes les valeurs du tableau contenu dans le tokener.
- `public JSONArray (String json)`: Création d'un tableau JSON suivant un fichier JSON.

Puisque notre requête HTTP lit un `InputStream` et sauvegarde son contenu dans une `String`, le dernier constructeur est celui qui convient le mieux à notre exemple. Concrètement, ce constructeur va créer un `JSONArray` et va contenir toutes les valeurs, les objets ou les tableaux contenus dans le tableau racine. Notez que si le serveur applicatif avait renvoyé non pas un tableau mais un objet, il aurait fallu utiliser un des constructeurs de `JSONObject`.

Une fois que vous avez votre tableau JSON, une série de méthodes existe pour en extraire les différentes valeurs. Nous avons déjà vu les différentes valeurs contenues dans un fichier JSON, il existe une méthode d'extraction pour chacun de ces types:

- `public boolean getBoolean (int index)`: Récupérer une valeur booléenne.
- `public double getDouble (int index)`: Récupérer un double.
- `public int getInt (int index)`: Récupérer un entier.
- `public long getLong (int index)`: Récupérer un long.
- `public String getString (int index)`: Récupérer une chaine de caractères.
- `public JSONArray getJSONArray (int index)`: Récupérer un tableau JSON.
- `public JSONObject getJSONObject (int index)`: Récupérer un objet JSON.

Il existe une alternative à chacune de ces méthodes. Le problème avec les méthodes `get*()`, il lancera une exception `org.json.JSONException` si vous tentez de récupérer une valeur qui n'existe pas. Vous pouvez donc utiliser les méthodes `opt*()` qui renverra une valeur par défaut si la valeur n'est pas présente. Je vous encourage donc à utiliser cette version si la valeur n'est pas forcément nécessaire et qu'une valeur par défaut peut convenir.

3.2.2. Extraire un objet

Pour extraite un objet d'un fichier JSON, c'est similaire aux tableaux. Vous devez utiliser la classe `JSONObject` qui possède 5 constructeurs:

- `public JSONObject ()`: Création d'un objet JSON vide.
- `public JSONObject (Map copyFrom)`: Création d'un objet JSON qui copie toutes les clés/valeurs de la map donnée.
- `public JSONObject (JSONTokener readFrom)`: Création d'un objet JSON qui copie toutes les clés/valeurs du prochain objet dans le tokener.

— `public JSONObject (String json)`: Création d'un objet JSON suivant un fichier JSON.
— `public JSONObject (JSONObject copyFrom, String[] names)`: Création d'un objet JSON qui copie tous les noms du tableau avec l'objet JSON.

Tout comme l'instanciation d'un tableau JSON, nous utiliserons principalement le constructeur qui prend une chaine de caractères en paramètre.

Quant aux méthodes d'extraction, elles sont strictement identiques au JSONArray. Vous n'aurez donc aucun mal à les utiliser.

3.2.3. Extraire les données

La théorie est terminée, nous allons passer à la pratique. L'exemple étant assez simple, il n'y aura qu'une seule classe qui aura la responsabilité d'extraire un `JSONObject` et de renseigner les valeurs que contient cet objet dans les attributs de la classe, que nous appellerons une classe du domaine nommée `Product`.

```java
package org.randoomz.androidnetworkingsample.models;

import android.os.Parcel;
import android.os.Parcelable;

import org.json.JSONObject;

public class Product implements Parcelable {
  private final long id;
  private final String name;
  private final String type;
  private final double price;

  public Product(JSONObject jObject) {
    this.id = jObject.optLong("id");
    this.name = jObject.optString("name");
    this.type = jObject.optString("type");
    this.price = jObject.optDouble("price");
  }

  public long id() { return id; }

  public String name() { return name; }

  public String type() { return type; }

  public double price() { return price; }
}
```

Code: Considérée comme une classe du modèle, elle représente un produit.

Cependant, avant d'obtenir les objets des produits, il faut itérer sur le tableau JSON qui se trouve à la racine du fichier JSON. Pour ce faire, nous allons récupérer la chaine de caractères du JSON, nous allons la placer dans un `JSONArray` et nous allons itérer dessus.

```
1  private List<Product> parse(final String json) {
2    try {
3      final List<Product> products = new ArrayList<>();
4      final JSONArray jProductArray = new JSONArray(json);
5      for (int i = 0; i < jProductArray.length(); i++) {
6        products.add(new Product(jProductArray.optJSONObject(i)));
7      }
8      return products;
9    } catch (JSONException e) {
10     Log.v(TAG, "[JSONException] e : " + e.getMessage());
11   }
12   return null;
13 }
```

Code: Traduit une représentation en chaine de caractères d'un fichier JSON en une liste de produits.

Vous disposez maintenant de la liste des produits et vous pouvez l'utiliser n'importe où dans votre application comme dans une liste.

3.3. Facilitons-nous la vie avec des bibliothèques

Vous savez quoi? Oubliez tout ce que nous venons d'apprendre dans cette section, ou plus justement ne l'utilisez plus jamais. L'une des forces d'Android se situe dans sa communauté. Elle est tellement énorme et active qu'il existe des milliers de bibliothèques pour aider les développeurs dans le développement de leurs applications Android.

Dans le cadre de ce tutoriel, nous allons en apprendre 3 qui me semblent indispensables à connaitre pour tout développeur Java. Java parce que ces bibliothèques sont adaptées au développement Android et Java standard.

Dans la suite de cette section, nous allons partir du principe que vous utilisez Android Studio et donc Gradle comme gestionnaire de dépendances. Toutes les dépendances seront à renseigner dans le fichier Gradle `build.gradle`.

3.3.1. OkHttp et Gson

Dans les exemples précédents, nous avons appris à utiliser les classes standards du framework Android mais elles sont loin d'être ergonomique. Pour palier ce problème, nous allons voir deyx bibliothèques qui vont simplifier vos requêtes HTTP et la transformation d'une chaine de caractères dans des objets Java.

OkHttp est une bibliothèque qui tente de proposer une solution moderne pour effectuer des requêtes web. Son utilisation est extrèmement simple et se découpe en 3 étapes:

1. Préparation de la requête HTTP grâce au design pattern Builder .
2. Exécution de la requête HTTP précédemment préparée.
3. Récupère le résultat de la requête.

```
1  private final OkHttpClient client = new OkHttpClient();
2
3  public String get(String url) throws IOException {
4    // Prepare the request.
5    Request request = new Request.Builder().url(url).build();
6    // Execute the request.
7    Response response = client.newCall(request).execute();
8    // Get the result.
9    return response.body().string();
10 }
```

Code: Requête HTTP vers un serveur avec la bibliothèque OkHttp.

C'est tout … Exit la dizaine de ligne de code pour préparer la requête HTTP ou la manipulation des `InputStream` pour récupérer les résultats. Toutes ces choses sont gérées par la bibliothèque. Vous pouvez vous concentrer uniquement sur vos requêtes HTTP. Malheureusement, toute la gestion des codes HTTP restent de votre ressort, mais l'API est bien mieux pensée pour la gérer.

Pour toutes les informations à propos de cette bibliothèque, rendez-vous sur sa documentation officielle et pour l'utiliser, il suffit de renseigner la dépendance dans le fichier build.gradle de votre projet.

```
1  compile 'com.squareup.okhttp3:okhttp:3.3.1'
```

Code: Dépendance Gradle pour OkHttp.

Gson peut être utilisé pour convertir des objets Java dans une représentation JSON ou inversement, d'une chaine de caractères JSON vers des objets Java. L'idée est de se passer d'une API pour placer manuellement les valeurs d'un JSON vers des objets Java, laisser la bibliothèque gérer cette tâche.

Son usage est déconcertant de simplicité. Il vous suffit d'instancier un objet `Gson` et d'utiliser les méthodes `toJson(...)` pour passer d'un objet à une représentation JSON ou aux méthodes `fromJson(...)` pour transformer un fichier JSON en des objets Java.

Par exemple, si le fichier JSON représente directement un objet, vous pouvez placer les valeurs du fichier JSON dans un objet grâce à la ligne suivante:

```
1 Product p = new Gson().fromJson(json, Product.class);
```

Code: Traduit une représentation en chaine de caractères d'un fichier JSON en un objet produit avec Gson.

Dans notre tutoriel, nous avons un fichier JSON qui représente une liste de produits. Nous devons informer Gson et lui fournir le type adéquat de la manière suivante:

```
1 List<Product> list = new Gson().fromJson(json, new
      TypeToken<List<Product>>() {}.getType());
```

Code: Traduit une représentation en chaine de caractères d'un fichier JSON en une liste de produits avec Gson.

Grâce au type générique spécifié dans `TypeToken`, `Gson` est au courant du type attendu et peut convertir le fichier JSON dans une liste appropriée.

Tout ne fonctionne pas par magie non plus. Il existe une limitation à Gson. Pour que Gson puisse placer les valeurs dans un objet JSON, vous devez avoir des attributs dans votre classe qui correspondent aux clés spécifiées dans le fichier JSON. Dans le cas contraire, placez l'annotation `@SerializedName` sur l'attribut qui ne possède pas le bon nom avec comme paramètre à l'annotation, la valeur de la clé dans le fichier JSON. Par exemple, pour un attribut qui se nomme "nom" mais avec une clé "name" dans le fichier JSON, vous devez annoter l'attribut de la manière suivante:

```
1 @SerializedName("name") private String nom;
2 ```
```

Pour toutes les informations à propos de cette bibliothèque, rendez-vous sur sa documentation officielle et pour l'utiliser, il suffit de renseigner la dépendance dans le fichier build.gradle de votre projet.

```
1 compile 'com.google.code.gson:gson:2.6.2'
```

Code: Dépendance Gradle pour Gson.

3.3.2. Retrofit

Vous êtes déjà impressionné par OkHttp et Gson? Alors vous ne connaissez pas encore Retrofit. Cette bibliothèque permet de faire des requêtes HTTP avec une plus grande simplicité que OkHttp, s'occupe tout seul de convertir vos JSON dans des objets Java et vous propose une

architecture adaptée pour réagir en conséquence aux codes HTTP. Par contre, elle est un poil plus complexe à utiliser. Il faudrait un tutoriel entier pour couvrir toutes les possibilités. Dans le cadre de ce tutoriel ci-présent, nous verrons que les strictes bases. En attendant un tutoriel français sur le sujet, rendez-vous sur la documentation officielle de la bibliothèque.

Pour utiliser Retrofit, vous devez disposer d'une instance de `Retrofit` (si possible unique dans votre projet), d'une ou plusieurs interfaces Java pour exécuter des requêtes vers votre serveur et d'une instance vers cette interface. Vous vous retrouvez avec le code suivante:

Une interface `ProductService` avec une seule méthode `list()` pour récupérer la liste des produits. Cette méthode est annotée par `@Get("list")` pour informer Retrofit que la requête HTTP doit se faire en GET et que l'URL se termine par "list". La méthode doit retourner `Call<List<Product>`. `Call` est une interface de Retrofit qui permet d'exécuter en synchrone ou asynchrone la requête et le type générique à l'intérieur informe Retrofit du type de retour souhaité.

```
1 public interface ProductService {
2   @GET("list") Call<List<Product>> list();
3 }
```

Code: Interface Retrofit pour faire des requêtes HTTP.

Maintenant que nous avons l'interface, il suffit de l'utiliser avec une instance. Pour récupérer une instance de cette interface, il faut passer par une instance de `Retrofit`. Cette instance s'initialise grâce à un `Builder`. Dans ce builder, vous spécifiez l'URL de base de votre serveur (dans notre exemple "http://www.randoomz.org/").

```
1 Retrofit retrofit = new Retrofit.Builder()
2     .baseUrl("http://www.randoomz.org/")
3     .build();
```

Code: Construction d'un objet Retrofit pour récupérer des instances d'un service et faire des requêtes HTTP.

L'instance `Retrofit` à disposition, vous pouvez construire une instance de votre service grâce à la méthode `create(Class<?>)`. Cette instance n'est pas réellement instanciée par l'interface puisque le langage Java ne le permet pas. Néanmoins, vous n'avez pas à vous soucier de l'implémentation de l'interface. Vous pouvez utiliser les méthodes déclarées dans l'interface et Retrofit se chargera de faire vos requêtes HTTP!

```
1 final ProductService service =
      retrofit.create(ProductService.class);
```

Code: Récupération d'une instance d'un service pour faire des requêtes HTTP.

Pour exécuter une requête HTTP, vous avez 2 solutions:

— Exécuter la requête en synchrone `service.list().execute()`. Il en sera de votre responsabilité d'exécuter la requête dans un thread secondaire.
— Exécuter la requête en asynchrone `service.list().enqueue(Callback)`. Vous n'êtes pas obligé de vous soucier d'exécuter la requête dans un thread secondaire, Retrofit le fait pour vous!

Ainsi, récupérer la liste des produits en asynchrone se fera de la manière suivante:

```
service.list().enqueue(new Callback<List<Product>>() {
  @Override public void onResponse(Call<List<Product>> call,
      Response<List<Product>> response) {
    // Request ok. Get the result.
  }

  @Override public void onFailure(Call<List<Product>> call,
      Throwable t) {
    // Request failed. Check why.
  }
});
```

Code: Requête HTTP vers un serveur avec la bibliothèque Retrofit.

Retrofit s'occupe tout seul de faire la requête HTTP, il vous fournit automatiquement la conversion JSON vers vos classes Java. Concernant les cas d'erreur, évitez de vous faire avoir. Comme vous le voyez dans l'exemple, vous disposez de la méthode `onFailure(Call, Throwable)` mais elle sera appelée uniquement s'il y a eu un problème avec la requête HTTP comme une perte de la connexion internet. Si vous voulez gérer les codes d'erreur HTTP, vous devez le faire dans la méthode `onResponse(Call, Response)` grâce à la vérification `response.isSuccessful()` sur la réponse de la requête. Si `isSuccessful()` renvoie faux, vous pouvez récupérer le code HTTP et son message d'erreur.

Comme d'habitude, ces explications ne sont que des introductions. Rendez-vous à la documentation officielle pour connaitre toutes les possibilités de cette bibliothèque. Quant à la dépendance Gradle, il suffit de spécifier cette ligne dans le bloc `dependencies` de votre fichier build.gradle.

```
compile 'com.squareup.retrofit2:retrofit:2.0.2'
```

Code: Dépendance Gradle pour Retrofit.

A la lecture de ce tutoriel, une architecture client/serveur devrait avoir moins de secret pour vous et vous devriez être capable d'effectuer des requêtes HTTP à partir d'une application Android. D'ailleurs, si vous voulez vous entrainer sur ce dernier point, sachez que Zeste de Savoir met à la disposition des développeurs une API pour récupérer des informations comme les membres et vos conversations privées. Rien ne vous empêche de développer une application Android qui interroge cette API pour vous entrainer, voire même pour proposer votre propre

vue pour consulter ces informations. Tout dépend de vous, de votre motivation et de votre imagination pour proposer des alternatives!

Merci à artragis , Arius et aux membres de la communauté de Zeste de Savoir pour leurs retours sur ce tutoriel.

Ce tutoriel dans sa version écrite ne vous suffit pas? Sachez que des tutoriels vidéos seront rajoutés dans les semaines à venir pour vous proposer une alternative pour apprendre tous les concepts enseignés dans ce tutoriel! Pour être tenu au courant, il vous suffit de suivre ce contenu grâce au bouton "Suivre ce contenu". Je me chargerais personnellement de poster un message dans les commentaires pour vous notifier d'une nouvelle vidéo qui sera intégrée directement dans ce contenu. Vous aurez alors une notification dans votre centre de notifications sur Zeste de Savoir!

L'icône est le logo officiel du projet PHP For Android et disponible en libre téléchargement sur le site du projet.