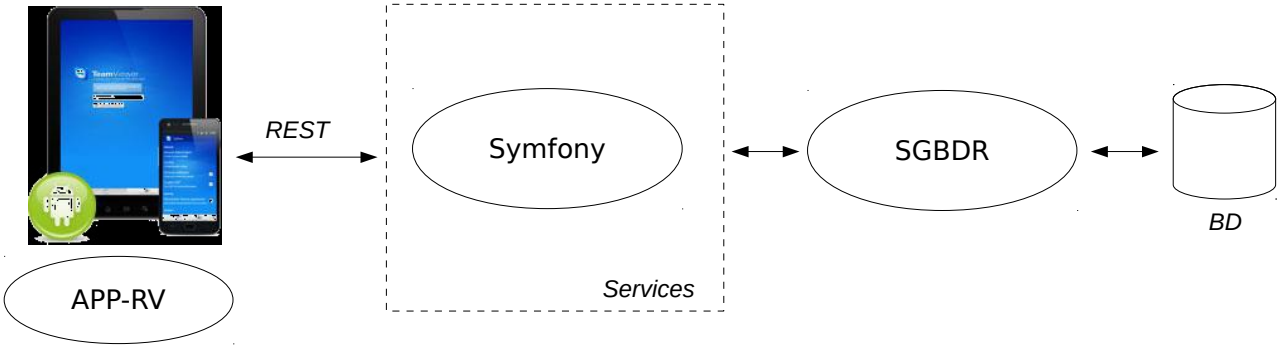
Android – Services Web

1- Contexte GSB - Application GSB-RV/Visiteur



2 – Format JSON : JavaScript Object Notation

2.1 – Principe

JSON (*JavaScript Object Notation*) est un format textuel de données. Ce format s'appuie sur les types de données suivants :

- Les chaînes (string) ;
- Les nombres (number) ;
- Les booléens (true et false) ;
- La valeur nulle (null) ;
- Les tableaux ([ ]) représentés par des listes de valeurs ;
- et les objets ({ }) représentés par des ensembles de paires clé/valeur (*clé*: *valeur*).

Exemple : Notes d'un étudiant

```
{
 "nom": "GENPRI" ,
 "prenom": "Erwan" ,
 "notes": {
                    "slam1": [ 12.5 , 18 , 9 ] ,
                    "slam2": [ 8.5 , 14 , 11 , 15 ]
          }
}
```

2.2- API – JSON

Java

Il existe de nombreuses bibliothèques Java dédiées à la manipulation des données au format JSON. Nous utiliserons la bibliothèque ***gson*** de *Google.*

Bibliothèque : **gson 2.*x***
Site : https://github.com/google/gson
Documentation : https://sites.google.com/site/gson/gsonuserguide

La classe Gson propose plusieurs méthodes dont :

- **fromJson()** dont le rôle est de créer un objet à partir de sa représentation textuelle au format JSON (***dé-sérialisation***).
- **toJson()** dont le rôle est de générer une représentation textuelle au format JSON d'un objet (***sérialisation***).

Pour instancier la classe Gson, il faut passer par une "fabrique" d'objets de la classe Gson. Il est donc nécessaire de commencer par créer cette "fabrique" qui est un objet de la classe GsonBuilder :

```
GsonBuilder fabrique = new GsonBuilder() ;
Gson gson = fabrique.create() ;
```

Symfony :
Pour manipuler le résultat d'une requête doctrine (type entity), il suffit d'implémenter dans la classe concernée l'interface JsonSerializable et d'implémenter la méthode jsonSerialize(). Par exemple, dans l'entité Visiteur, on peut implémenter la méthode de la manière suivante :

```
public function jsonSerialize() {
   return array(
          'visMatricule' => $this→visMatricule,
          'visNom' => $this→visNom,
          ...
   ) ;
}
```

3- Architecture REST (*REpresentational State Transfert*)

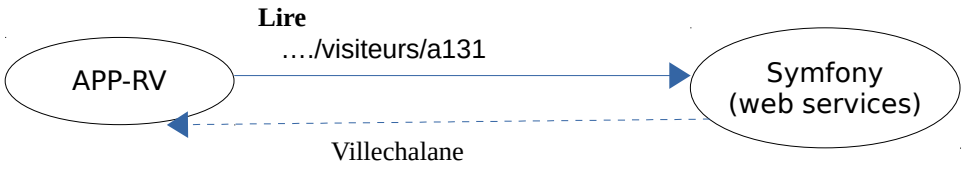
REST est un style d'architecture orientée service. Ce type d'architecture s'appuie sur le protocole HTTP (méthodes et codes d'état) pour accéder aux ressources et les manipuler.

3.1- Les ressources

Les ressources sont identifiées de façon unique par leur URI (*Uniform Resource Identifier*). Dans le *web*, les URI prennent la forme d'URL (*Uniform Resource Locator*).

Exemple : Le visiteur identifié par le matricule 'a131' a pour URI :
http://gsb.fr/VolleyGsb/web/app.php/visiteurs/a131 (à partir de maintenant, je n'indiquerai pas le chemin complet mais uniquement la route)

Les opérations de base sur les ressources sont au nombre de quatre (CRUD) :
1. Créer (**C**reate)
2. Lire (**R**ead)
3. Modifier (**U**pdate)
4. Supprimer (Delete)



3.2- La représentation des données

Plusieurs formats sont possibles pour représenter les données échangées entre le client et le serveur :
- CSV ;
- XML ;
- YAML ;
- JSON ;
- **…**

Le format des données utilisé dans les réponses peut être différent de celui utilisé dans les requêtes.

3.3- Les opérations CRUD

REST s'appuie sur les méthodes HTTP ("*fonctions*") pour spécifier les opérations à appliquer sur les ressources et sur les codes d'état HTTP (un code d'état se trouve dans la ligne d'état d'une réponse HTTP) pour s'assurer du bon déroulement des opérations.

Méthode HTTP	CRUD	Code d'état (réponse HTTP)	SQL
POST	Create	(201) : created : ressource créée (409) : ressource existe déjà	INSERT

GET	Read	(200) : OK – ressource trouvée (404) : Not Found – ressource non trouvée	SELECT
PUT	Update	200 (OK) : ressource modifiée 404 (Not Found) : ressource non trouvée 204 (No Content) : pas de données pour modification	UPDATE
DELETE	Delete	200 (OK) : ressource supprimée 404 (Not Found) : ressource non trouvée	DELETE

4 - Configuration

4.1- Permissions

Pour utiliser les services relatifs au réseau et à Internet, il faut accorder certaines permissions à l'application.

Extrait du fichier AndroidManifest.xml :

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

4.2- Bibliothèques

Deux bibliothèques seront exploitées :

- Volley pour la communication avec le serveur HTTP ;
- Gson pour la manipulation d'objets JSON.

Sous *Android Studio* les applications sont construites au moyen du moteur de production *Gradle*. Cet outil permet d'automatiser la production (dépendances, compilation, tests...) dont les différentes tâches sont décrites dans un fichier.

Pour intégrer les deux bibliothèques au projet, il faut ajouter deux dépendances (section dependencies du fichier build.gradle).

Extrait du fichier build.gradle (Module: app) :

```
compile 'com.android.volley:volley:1.0.0'
compile 'com.google.code.gson:gson:1.7.2'
```

5 - Requêtes et réponses

Toutes les requêtes sont traitées au niveau de Symfony dans le fichier DefaultController.php.

5.1- Réponse : Texte

Côté serveur Symfony :
L'action getVisiteurNomAction($matricule) a pour rôle de retourner le nom du visiteur dont le matricule est envoyé dans l'URL : http://<serveur>/<NomProjet>/web/app_dev.php/visiteurs/<matricule>

Exemple : http://gsb.fr/VolleyGsb/web/app.php/visiteurs/a131 devra retourner Villechalane

Exercice 1 : Ecrire le code de l'action getVisiteurNomAction($matricule)

Côté client Android :

Source : Le matricule sera saisi dans l'activité principale. La validation sur le bouton affichera un message Toast avec le nom du visiteur.

```
...

String matricule = URLEncoder.encode( etMatricule.getText().toString() ) ;
String url = "http://gsb.fr/VolleyGsb/web/app.php/visiteurs/"  + matricule ;

// Pour appeler le web service , on définit une instance de la classe android.Volley.Response/Listener pour traiter la réponse

Response.Listener<String> ecouteurReponse = new Response.Listener<String>() {

    @Override
    public void onResponse(String response) {
            ...
    }
} ;

// et une variable de type Response.ErrorListener
Response.ErrorListener ecouteurErreur = new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
            Log.e( "APP-RV" , "Erreur HTTP : " + error.getMessage() ) ;
            ...
    }
} ;

// instancier l'objet de type StringRequest (type du résultat retourné)
// ici on utilise la méthode HTTP GET
StringRequest requete = new StringRequest( Request.Method.GET , url , ecouteurReponse ,
                                                                                    ecouteurErreur ) ;

RequestQueue fileReq = Volley.newRequestQueue( contexte ) ;
fileReq.add( requete ) ;
```

5.2- Réponse : Objet JSON

Service REST : Retourne la ville dont le code postal est indiqué, sous la forme d'un objet JSON composé de deux éléments dont les clés sont 'nom' et 'codePostal'.

Route : /villes/*codePostal*

Exemple : http://gsb.fr/villes/29000

Ville
- nom : String - codePostal : String

Source :

```
String url = "http://gsb.fr/villes/29000";

Response.Listener<JSONObject> ecouteurReponse = new Response.Listener<JSONObject>() {

    @Override
    public void onResponse(JSONObject response) {

            try {
                    ...
                    String nomVille = response.getString( "nom" ) ;
                    ...
            }
            catch( JSONException e){
                    Log.e( "APP-RV" , "Erreur JSON : " + e.getMessage() ) ;
                    ...
            }
    }
} ;
```

```
Response.ErrorListener ecouteurErreur = new Response.ErrorListener() {

    @Override
    public void onErrorResponse(VolleyError error) {
            Log.e( "APP-RV" , "Erreur HTTP : " + error.getMessage() ) ;
            ...
    }
} ;

JsonObjectRequest requete = new JsonObjectRequest( Request.Method.GET , url , null ,
                                                                    ecouteurReponse , ecouteurErreur ) ;
```

```
RequestQueue fileReq = Volley.newRequestQueue( contexte ) ;
fileReq.add( requete ) ;
```

5.3- Réponse : Tableau JSON

Service REST : Retourne la liste des villes localisées dans le département indiqué, sous la forme d'un tableau d'objets JSON dont chaque élément est composé de deux éléments dont les clés sont 'nom' et 'codePostal'.

Route : /dept/*numéroDépartement*/villes

Exemple : http://gsb.fr/dept/29/villes

Source :

```
String url = "http://gsb.fr/dept/29/villes" ;

Response.Listener<JSONArray> ecouteurReponse = new Response.Listener<JSONArray>() {

    @Override
    public void onResponse(JSONArray response) {
            try {
                    for( int i = 0 ; i < response.length() ; i++ ){

                            Log.i( "APP-RV" , response.getJSONObject( i ).getString( "cp" )
                                            + " "
                                            + response.getJSONObject( i ).getString( "nom" )
                            ) ;
                    }
            }
            catch(JSONException e){
                    Log.e( "APP-RV" , "Erreur JSON : " + e.getMessage() ) ;
                    ...
            }
    }
} ;

Response.ErrorListener ecouteurErreur = new Response.ErrorListener(){

    @Override
    public void onErrorResponse(VolleyError error) {
            Log.e( "APP-RV" , "Erreur HTTP : " + error.getMessage() ) ;
            ...
    }
} ;

JsonArrayRequest requete = new JsonArrayRequest( Request.Method.GET , url , null , ecouteurReponse ,
ecouteurErreur ) ;

RequestQueue fileReq = Volley.newRequestQueue( contexte ) ;
fileReq.add( requete ) ;
```

5.4- Méthode POST : Envoi d'un objet JSON

Service REST : Enregistre dans la base de données la ville envoyée sous la forme d'un objet JSON composé de deux éléments dont les clés sont 'nom' et 'codePostal'.

Route : /villes

Exemple : http://gsb.fr/villes

Source :

```
final Ville uneVille = new Ville( "44000" , "Nantes" ) ;

GsonBuilder fabrique = new GsonBuilder() ;
final Gson gson = fabrique.create() ;

String url = "http://gsb.fr/villes" ;

try {
    StringRequest requete = new StringRequest(
                        Request.Method.POST,
                        url,
                        new Response.Listener<String>() {
                                @Override
                                public void onResponse(String response) {
                                        ...
                                }
                        } ,
                        new Response.ErrorListener(){
                                @Override
                                public void onErrorResponse(VolleyError error) {
                                        ...
                                }
                        }
    ){
            @Override
            protected Map<String, String> getParams() throws AuthFailureError {
                    Map<String,String> parametres = new HashMap<String,String>() ;
                    parametres.put( "ville" , gson.toJson( uneVille ) ) ;
                    return parametres ;
            }
    } ;

    RequestQueue fileReq = Volley.newRequestQueue( contexte ) ;
    fileReq.add( requete ) ;

}
catch( Exception e ){
    Log.e( "APP-RV" , e.getMessage() ) ;
}
```

6- Application GSB-RV/Visiteur

L'objectif est de remplacer le modèle "simulé" par des accès à des *web-services* REST implémentés au moyen de *Symfony*.

Démarche :

1. Identifier les *web-services* à proposer ;
2. Écrire et tester les requêtes SQL associées ;
3. Déclarer les routes ;
4. Modifier l'implémentation de l'application mobile.