



**Institut Universitaire de Technologie**

**Département Informatique**  
Site de Bourg-en-Bresse

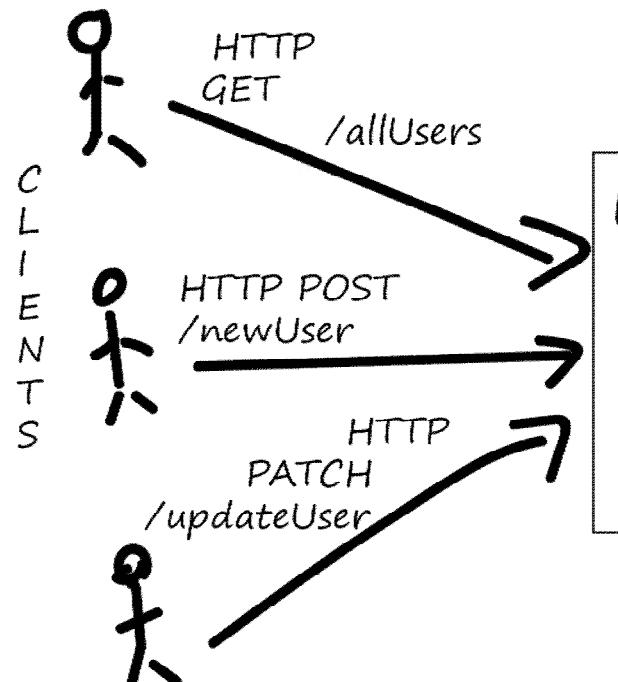


ANDROID

## Les Services Web

[lionel.buathier@univ-lyon1.fr](mailto:lionel.buathier@univ-lyon1.fr)

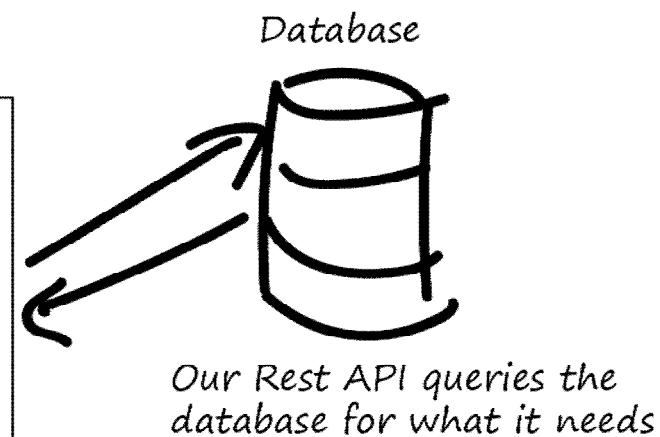
# Rest API Basics



Our Clients, send HTTP Requests  
and wait for responses

**Rest API**  
Receives HTTP  
requests from  
Clients and does  
whatever request  
needs. i.e create  
users

Typical HTTP Verbs:  
GET -> Read from Database  
PUT -> Update/Replace row in Database  
PATCH -> Update/Modify row in Database  
POST -> Create a new record in the database  
DELETE -> Delete from the database



Response: When the Rest API has what it  
needs, it sends back a response to the  
clients. This would typically be in JSON  
or XML format.

<https://tutorialedge.net/general/what-is-a-rest-api/>



# Introduction aux WebServices RESTful

- ▶ REST est une architecture de services Web, à la manière de [SOAP](#) et de [XML-RPC](#). C'est l'acronyme de *REpresentational State Transfer*.
- ▶ [Elaboré en l'an 2000 par Roy Fielding](#), un des créateurs du prot. HTTP
- ▶ Principe : Internet est composé de ressources accessibles à partir d'une URL. Par exemple, <http://www.meteo.fr/paris/> où Paris est une ressource définie par Météo France.
- ▶ La requête de cet URL renvoie une représentation de la ressource demandée (paris.php, par exemple) qui place l'application cliente dans un état (*state*) donné.
- ▶ Un appel différent renvoie une autre représentation. Ainsi, l'application cliente change d'état (*state transfer*) pour chaque représentation de ressource. Utilisé par les SPA (Single Page Appli)



# REST s'appuie sur le protocole http

Requête :

<VERB>	<URI>	<HTTP Version>
<Request Header>		
<Request Body>		

Réponse :

<HTTP Version>	<Response Code>
<Response Header>	
<Response Body>	

Un des objectifs fondamentaux d'une API REST est d'utiliser HTTP comme protocole applicatif, pour homogénéiser et faciliter les interactions entre SI et pour ne pas avoir à façonner un protocole « maison » de type *SOAP/RPC* ou *EJB*, qui a l'inconvénient de “réinventer la roue” à chaque fois.

Il convient donc d'utiliser systématiquement les verbes HTTP pour décrire les actions réalisées sur les ressources ([CRUD](#)).



# Correspondance Verbes / CRUD

Verbe HTTP	Correspondance CRUD	Collection : /orders	Instance : /orders/{id}
GET	READ	Read a list orders. 200 OK.	Read the detail of a single order. 200 OK.
POST	CREATE	Create a new order. 201 Created.	–
PUT	UPDATE/CREATE	–	Full Update. 200 OK. Create a specific order. 201 Created.
PATCH	UPDATE	–	Partial Update. 200 OK.
DELETE	DELETE	–	Delete order. 200 OK.

<http://blog.octo.com/designer-une-api-rest/#crud>



## Le bon usage

- ▶ Un service REST devrait respecter les "conventions" suivantes :
  - Bien identifier les ressources devant être exposées au travers du service et de manière unique
  - Chaque ressource devra se voir assigner une URL, de la forme :  
forme `http://www.site.com/contenus/1789`  
(plutôt que `http://www.site.com/contenus.php?id=1789`)
  - catégoriser les ressources selon les possibilités offertes à l'application cliente : lecture seule (GET) ou possibilité de modifier/créer une ressource (POST, PUT, DELETE) ?
  - chaque ressource devrait faire un lien vers les ressources liées
  - documenter l'API



## Sitographie

<http://blog.nicolashachet.com/niveaux/confirmee/larchitecture-rest-expliquee-en-5-regles/>

<http://blog.octo.com/designer-une-api-rest/>

<http://www.drdobbs.com/web-development/restful-web-services-a-tutorial/240169069>

[http://www.journaldunet.com/developpeur/tutoriel/xml/030707xml\\_rest1b.shtml](http://www.journaldunet.com/developpeur/tutoriel/xml/030707xml_rest1b.shtml)





**Institut Universitaire de Technologie**

**Département Informatique**

Site de Bourg-en-Bresse



ANDROID

# **Connexion à l'API Rest avec Retrofit**

**[lionel.buathier@univ-lyon1.fr](mailto:lionel.buathier@univ-lyon1.fr)**

# Présentation de Retrofit

Retrofit est un client REST pour Android créé par Square. Il simplifie grandement la connexion à une API Rest, la récupération de données et leur parsing.

Pour transformer vos réponses WS en données, Retrofit utilise des converters (qui transforment la donnée en objet). Vous pouvez par exemple utiliser Gson pour déserialiser vos réponses JSON ou des converters personnels.

La liste des converters est déjà importante et s'enrichie régulièrement (Gson, Jackson, Moshi, Protobuf...)



# Utilisation de Retrofit

L'utilisation de Retrofit nécessite trois briques :

- Des classes **Models** utilisées pour transformer les réponses en objets.
- Des interfaces contenant toutes les requêtes HTTP supportées par notre API.
- Le builder servant à créer et configurer l'instance de Retrofit à partir de l'interface et l'Url de base.



# Utilisation de Retrofit

L'utilisation de Retrofit nécessite trois briques :

- Des classes **Models** utilisées pour transformer les réponses en objets.
- Des interfaces contenant toutes les requêtes HTTP supportées par notre API.
- Le builder servant à créer et configurer l'instance de Retrofit à partir de l'interface et l'Url de base.



# Exemple d'utilisation

- ▶ Retrofit transforme notre API HTTP en une interface java

```
public interface GitHubService {  
    @GET("users/{user}/repos")  
    Call<List<Repo>> listRepos(@Path("user") String user); }
```

- ▶ La classe Retrofit génère une implémentation de l'interface GitHubService

```
Retrofit retrofit = new Retrofit.Builder()  
    .baseUrl("https://api.github.com/")  
    .build();  
  
GitHubService service = retrofit.create(GitHubService.class);
```

- ▶ Chaque appel 'Call' depuis GitHubService peut être une requête HTTP synchrone ou asynchrone au WS

```
Call<List<Repo>> repos = service.listRepos("octocat");
```



# Définition des requêtes dans l'interface

- ▶ On dispose des méthodes basiques des API Rest (@GET, @POST, @PUT, @DELETE)
- ▶ Ainsi que des annotations pour ajouter des éléments dans les URL :
  - @Path pour insérer des valeurs dans la requête (par ex. le nom de l'utilisateur)
  - @Query pour ajouter des paramètres de recherche en fin d'url (?sort=...)
- ▶ Exemple :

```
@GET( "user/{id} " )  
Call<User> getUser(@Path( "id" ) int userId,  
                      @Query( "sort" ) String sort);
```



## Définition des requêtes dans l'interface

- ▶ On peut ajouter des Body aux requêtes POST pour préciser l'objet à utiliser dans le 'request body' :

```
@POST( "user/new" )  
Call<User> createUser(@Body User user);
```

# Utilisation de convertisseurs (Gson)

- ▶ Par défaut, Retrofit ne peut que déserialiser les HTTP bodies en type OkHttp ResponseBody et n'accepte que le type RequestBody pour @Body.
- ▶ Pour utiliser Gson, il faut :
  - ajouter la dépendance : com.squareup.retrofit2:converter-gson
  - Utiliser la classe GsonConverterFactory pour générer l'implémentation de l'interface
- ▶ L'exemple précédent deviendrait :

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

```
GitHubService service = retrofit.create(GitHubService.class);
```



# Authentification

On peut le faire de 2 manières différentes :

- ▶ soit en manipulant le 'header' :

```
@GET( "user" )  
Call<User> getUser(@Header( "Authorization" ) String  
authorization)
```

*Remarque : on peut utiliser la classe OkHttps credentials pour générer facilement un crédit d'authentification ou un jeton d'API*

- ▶ Soit en utilisant un intercepteur OkHttp :

<https://github.com/square/okhttp/wiki/Interceptors>

# Sitographie – Bibliographie

<http://square.github.io/retrofit/>

<http://www.vogella.com/tutorials/Retrofit/article.html>

<https://futurestud.io/tutorials/retrofit-getting-started-and-android-client>

<http://www.jsonschema2pojo.org/>

Android 7, Les fondamentaux du développement d'applications Java  
Auteur : Nazim BENBOURAHLA, ed. ENI

