Les webservices

Sébastien NOUET

sebastien.nouet@ynov.com

Objectif

Maîtriser les technologies permettant à des applications de dialoguer à distance via Internet, et ceci indépendamment des plateformes et des langages sur lesquelles elles reposent.

Evaluation

1. Livraison d’un projet technique composé de :
    - un webservice REST écrit en PHP respectant une architecture SOA
    - sa documentation technique
2. Soutenance orale de votre webservice
3. Evaluation des connaissances dispensées en cours

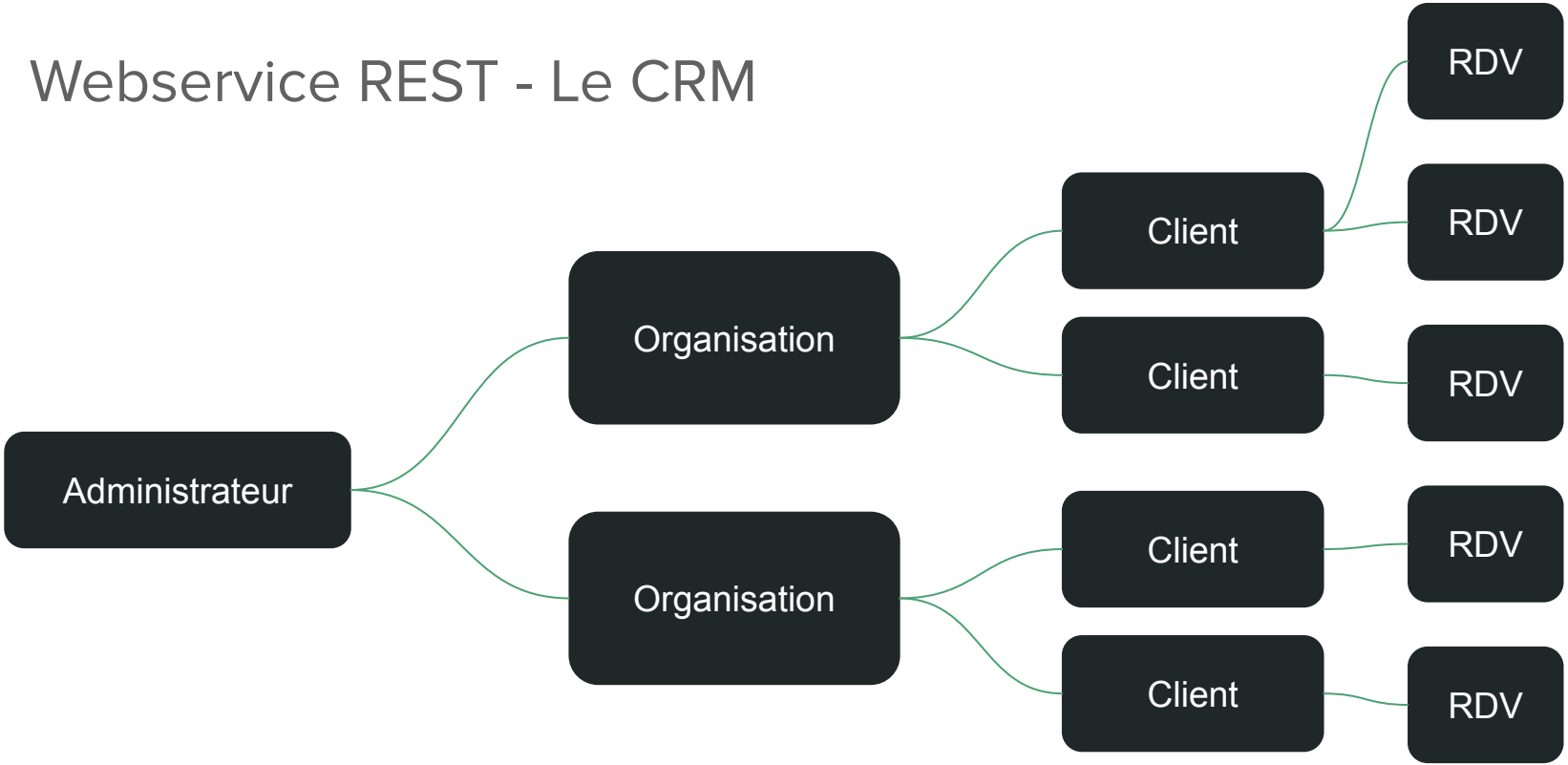
Evaluation - Projet technique

Travail en binôme ou trinôme.

Remis sous la forme d’un répertoire GIT facilement installable et exploitable au plus tard le 15 avril.

Evaluation - Projet technique

Webservice REST - Le CRM



Evaluation - Projet technique

	Administrateur	Organisation	Client	RDV
Description	Utilisateur physique ayant un accès au CRM	Entreprise appartenant à un utilisateur	Personne physique appartenant à une organisation	Événement réalisé avec un client
Champs	Prénom Nom Adresse email	Nom Adresse	Nom Prénom Adresse email Num. de téléphone	Résumé texte Date

Evaluation - Projet technique

Webservice REST - Le CRM

Possibilité pour un administrateur de CRUD* via une API des administrateurs, organisations, des clients et des rendez-vous.

Le webservice doit être documenté, facilement installable et utilisable.

* CRUD = Create Read Update Delete

Evaluation - Soutenance orale

Présenter le webservice du projet REST de CRM à l'oral devant un public de développeurs.

Savoir expliquer clairement les différentes fonctionnalités, leur architecture, les façons de les appeler, les choix techniques.

Présenter sa documentation.

Date prévue : le 18 mars.

Evaluation - Restitution

Evaluation des connaissances dispensées en cours.

Date prévue : le 01 avril


Formez vos équipes

Les webservices

Présentation

Qu’est-ce-qu’un webservice ?

Un peu d’Histoire

Web 1.0

Pages composées de textes, Hypertextes, liens

https://web.archive.org/web/19961229181022/http://www.stanford.edu/

Web 2.0

La page reste dans le navigateur et les informations sont récupérées en arrière-plan

https://www.stanford.edu/

Un peu d’Histoire

EDI - 1960’s

Système d’échange informatique de messages fortement formatés [...].

RPC - 1990’s

Système d’échange utilisée dans les Environnements Informatiques Distribués

Un peu d’Histoire

1998 : la révolution du XML

Remplace le SGML

Non Binaire - composé de texte

Facile à apprendre, il devient vite populaire

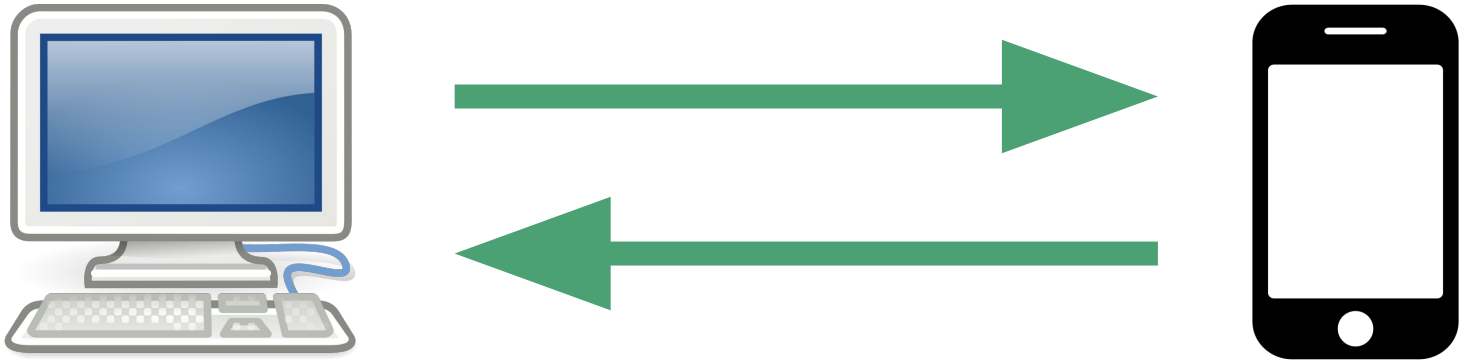
Devient le format standard pour le web

Qu’est-ce-qu’un webservice ?

“C’est un protocole d'interface informatique de la famille des technologies web permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués.”

Wikipedia 27/09/2021

Qu’est-ce-qu’un webservice ?



Appli web
PHP
Serveur Linux
France

Appli mobile
C++
Android
Angleterre

Qu’est-ce-qu’un webservice ?

Un webservice permet à une ressource de communiquer avec d'autres ressources sans connaître les détails de leur mise en œuvre.

Le plus souvent, le protocole HTTP est utilisé. le format d’ échange peut changer (JSON, XML).

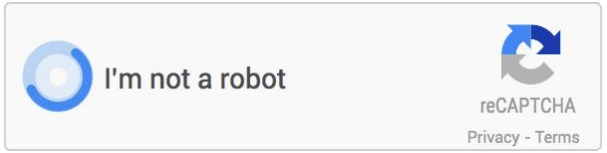
Avantages

Facilités de développement

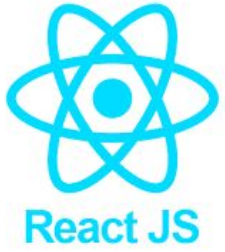
Normalisation des échanges (REST, SOAP)

Accès à de nouveaux métiers (ex Google Maps, reCaptcha, AWS…)

Utilisation fréquentes d’API

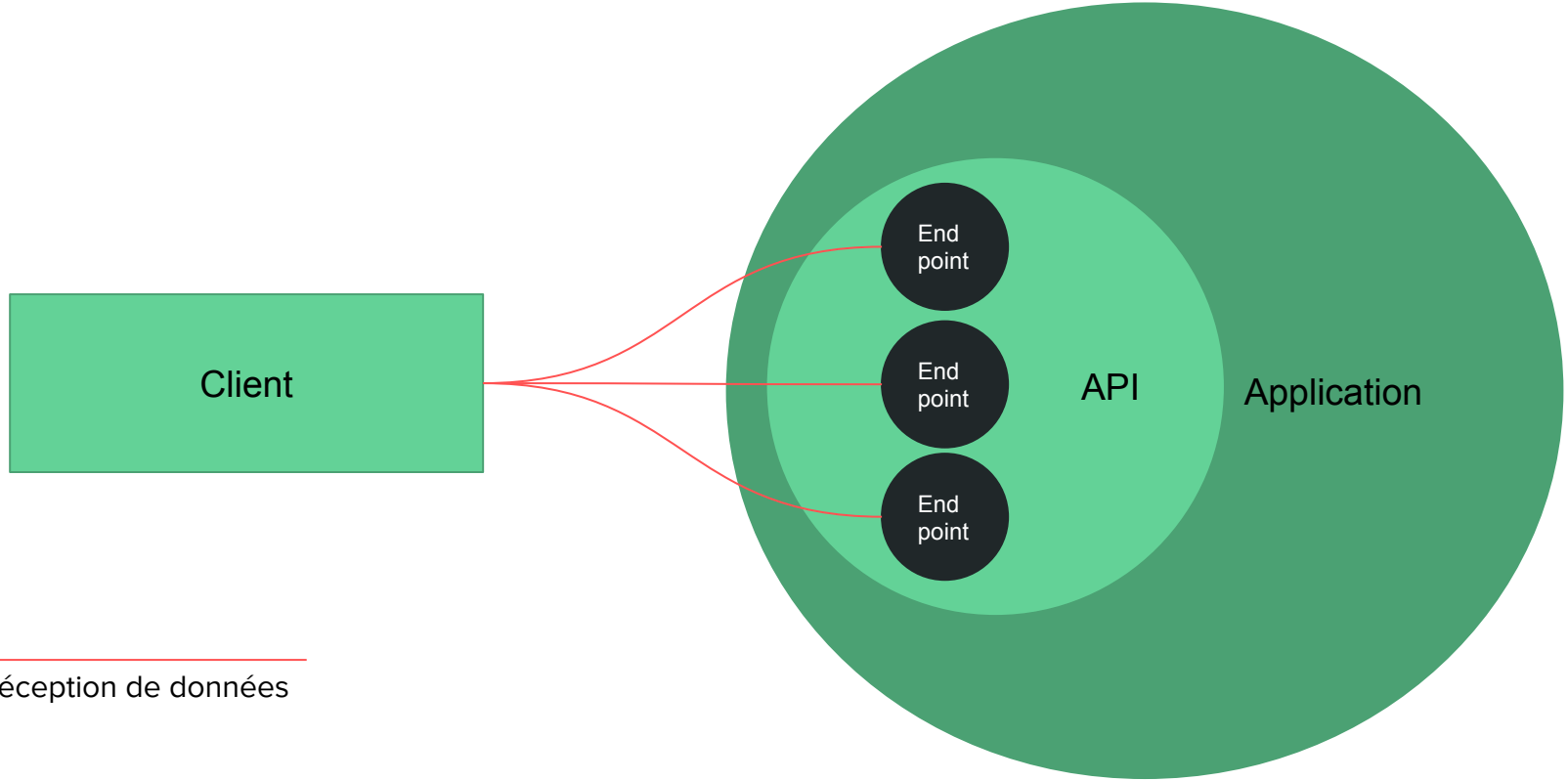


Gmail



Comment fonctionne un webservice ?

Fonctionnement basique d’une API



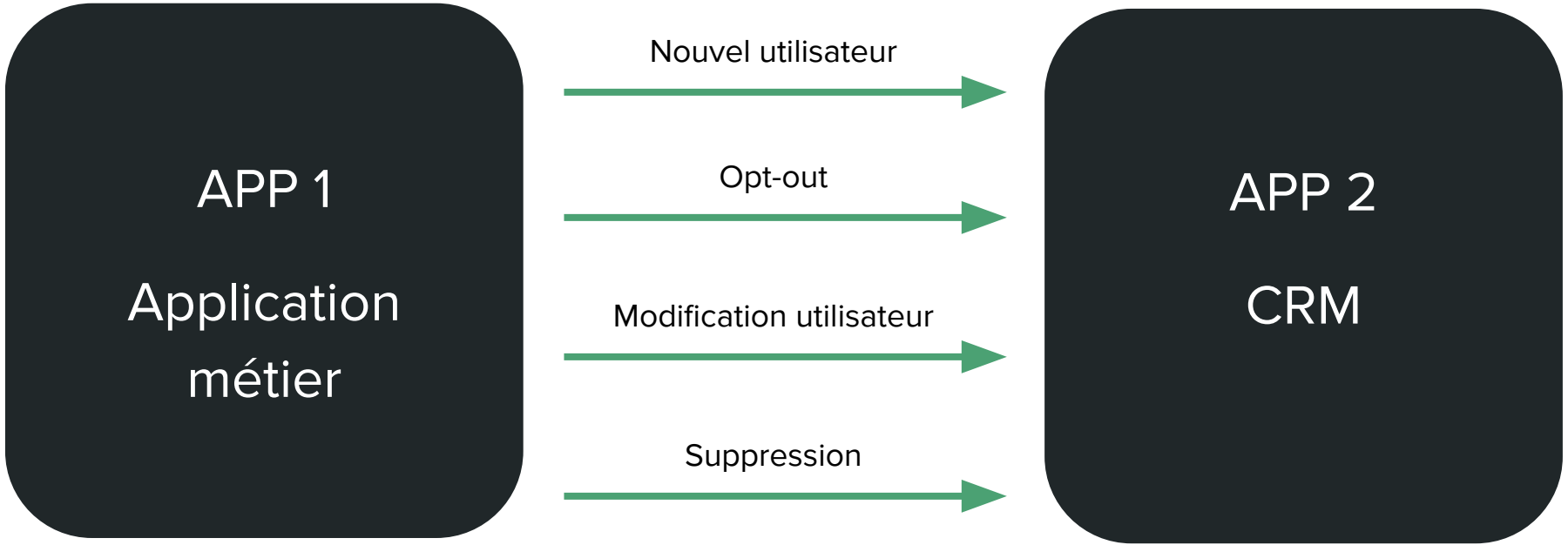
Envoi / réception de données

Les endpoints ?

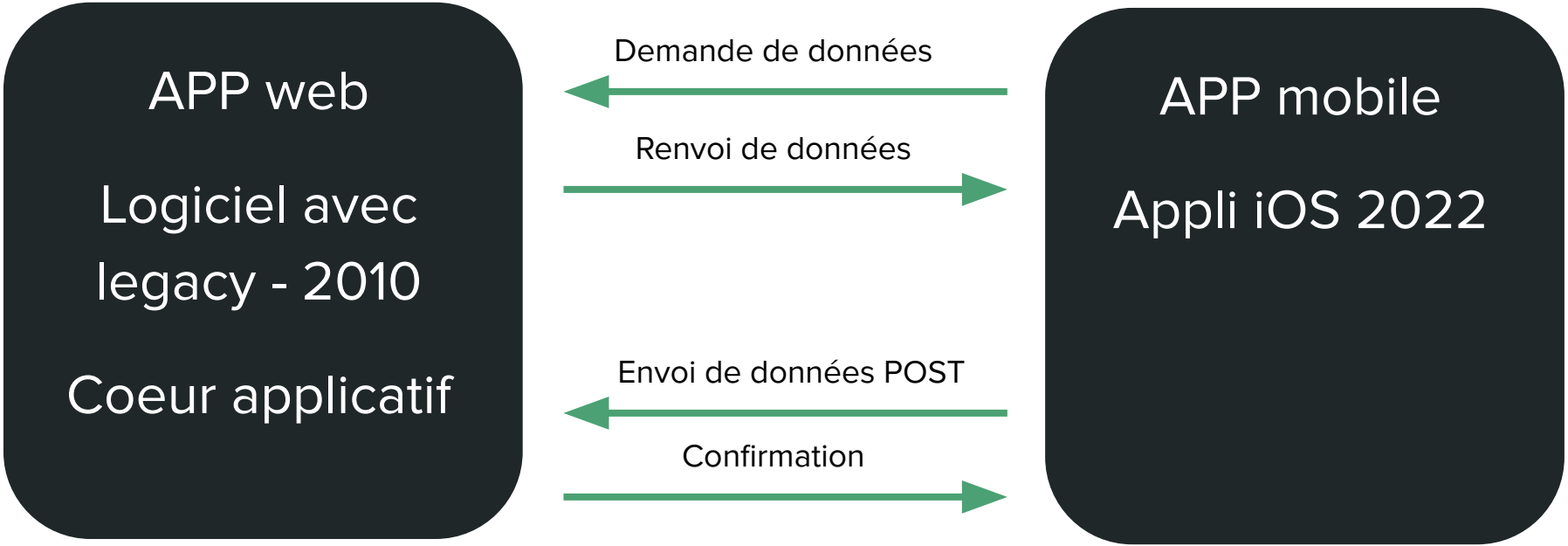
Un endpoint est le point d’entrée d’une fonctionnalité de votre API.

- Point d’entrée dans l’application destinée à l’API
- Exécute une fonction applicative (récupération de données, formatage, insertion, suppression…)
- Renvoie une donnée

Cas pratique 1 - Le CRM



Cas pratique 2 - Appli web et mobile



Une bonne API, c’est quoi ?

Documentation technique

Claire, précise et exhaustive

Doit contenir :

- Prérequis (domaine, authentification)
- Totalité des endpoints, composés de :
  - Présentation (utilité, fonction)
  - Données à envoyer (**précision**, **exhaustivité**)
  - Différentes réponses possibles (**précision**, **exhaustivité**)

Gestion des erreurs

Il est important d’accepter le fait qu’il y aura des erreurs dans votre application. Nous ne pouvons pas renvoyer du 200 à chaque appel.

- **Accepter** le principe qu’il y aura des erreurs (latence, DNS…)
- Les identifier et les anticiper
- Trier par criticité
- **Design for failure** : penser l’échec d’une fonctionnalité

Sécurité

Lors de la création d’un API, la sécurité fournie par le navigateur n’existe que peu ou plus. Il faut donc redoubler de vigilance.

- SSL **obligatoire** : CORE, Mixed-content
- Authentification et identification (Oauth, Hawk, Custom…)
- Essayer de limiter les rebonds entre serveurs
- Ne **jamais** faire confiance à l’utilisateur

Monitoring et reporting

Pour pouvoir suivre la performance et l’efficacité d’une application, il faut pouvoir en suivre le comportement.

- Enregistrement des requêtes (logs, base de données)
- Système d’alerte en cas d’échec de requête
- Possibilité de rejouer celles qui ont échoué

Scalabilité

La scalabilité concerne la capacité d'un système à s'adapter et faire face à différents volume de charge. Dans le cas d'une api, cela implique :

- Etre capable de gérer un important volume de données
- Etre capable de faire face à la multiplicité des requêtes
- **Importance du choix de la technologie** - ex : PHP peu adapté à la parallélisation, au contraire de NodeJs

Performance

Comme pour le web, une API doit viser la performance.
L'utilisateur ne doit **jamais** se retrouver forcé à recharger la page ou renvoyer une action.

- Limitation des données (récupérées et envoyées) au minimum
- Rendre la main le plus vite possible au client
- Éviter les rebonds entre services à cause de latences serveur et http
- Possibilité de faire du traitement asynchrone : retour avant le traitement
- Utilisation intelligente du cache

Maintenance

Comme tout système, votre API sera amenée à évoluer.
Plusieurs pièges sont à éviter :

- **Bannir** au maximum les changements non rétrocompatibles. L'utilisateur ne doit pas être dépendant des évolutions de notre API.
- Documentation **à jour et en ligne** (Read the docs, Wordpress, Bit.ai)
- Dans le cas d'une API intégrée à une appli, attention aux couplages forts et aux effets de bord

Deux approches différentes

REST

Representational State Transfer

Ensemble de principes architecturaux

Flexible et facile à utiliser

Basé sur HTTP

Retour le plus souvent en JSON

SOAP

Simple Object Access Protocol

Protocole officiel W3C

Protocole => plus lourd

Pas de protocole imposé (HTTP, SMTP...)

Retour en XML

Les webservices

APIs REST

Bonnes pratiques d’architecture et de design d’APIs ReSTful

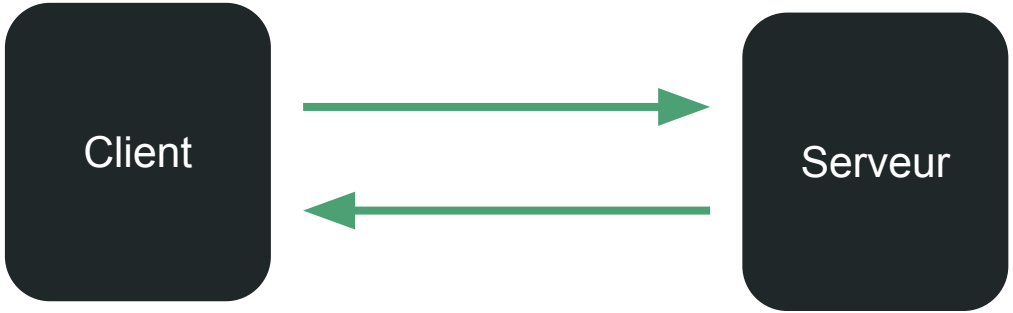
Les 6 contraintes de REST

1. Client - serveur
2. Stateless
3. Mise en cache
4. En couches
5. Interface uniforme
6. Code à la demande

1 - Client Serveur

Responsabilités client / serveur séparées : le client n’est en aucun cas responsable des données. Le serveur ne gère pas l’affichage.

Les composants doivent pouvoir évoluer indépendamment.



HTTP

2 - Stateless

Pas de sessions. Toutes les requêtes (même envoyées par un même client) sont traitées indépendamment.

La requête du client doit donc contenir tous les paramètres nécessaires pour que le serveur puisse répondre.

3 - Mise en cache

Mise en cache possible par les clients et serveurs

Le fonctionnement du cache, même absent, doit être défini pour éviter à l'utilisateur de récupérer des données obsolètes.

Améliore la performance pour les deux parties.

4 - Les couches

Architecture en couche. Chaque couche possède une fonctionnalité et responsabilité définies.

Le client ne doit rien savoir de la façon dont a été obtenue sa réponse. Des serveurs intermédiaires peuvent être utilisés, le client ne doit rien en aucun cas le savoir.

5 - Interface uniforme

Cette interface uniforme doit standardiser les échanges. Elle est orientée ressources. Chaque ressource doit :

- **posséder** un identifiant unique (exemple api.com/*users* ou api.com/*projects*)
- **disposer** d'une représentation suffisante pour identification / modification / suppression (un *uuid* ou *slug* par exemple)
- chaque ressource doit être **auto-décrite** (contenir assez d'infos pour se suffire à elle-même). Le header Content-type doit être ajouté à la réponse HTTP par exemple.

6 - Code à la demande (facultative)

Un client peut demander à obtenir du serveur un code à exécuter (JS, HTML, applet JAVA). Le client sera alors chargé de l’exécuter (PS : attention à la sécurité).

Exemple : une api peut, sur demande, renvoyer du HTML + CSS pour que le client n’ait qu’à “afficher” le contenu obtenu

REST - L’importance du HTTP

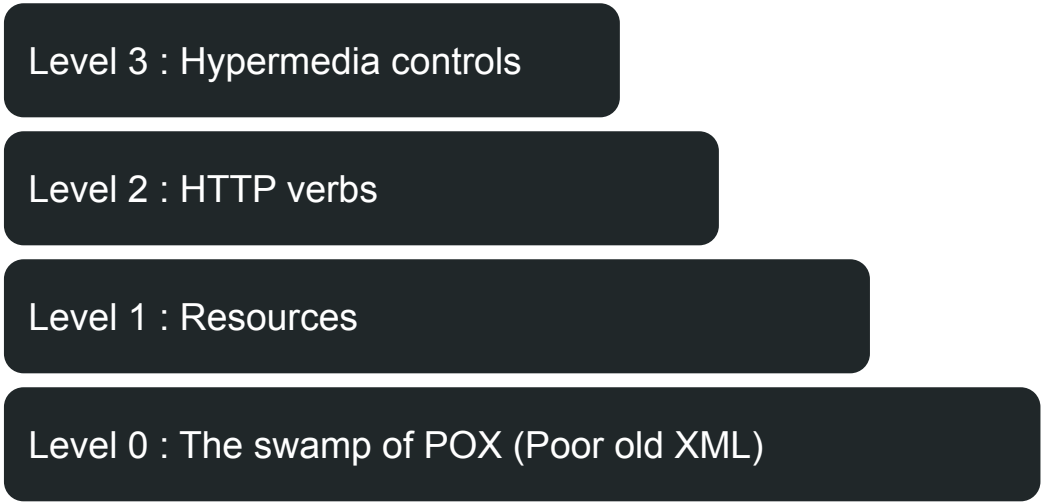
- Une URI de base (exemple https://api.com/users)
- **Méthode HTTP** : absolu des méthodes HTTP (GET, POST, PUT, DELETE, PATCH...) => *GET /user != POST /users*
- **Réponse HTTP** : version du protocole, code status, Content-type

```
HTTP/1.1 200 OK
Date:Tue, 31 Jan 2017 13:18:38 GMT
Content-Type: application/json

{
    "current status" : "Everything is ok!"
}
```

Le modèle de maturité de Richardson

Un moyen d’évaluer son API selon 4 niveaux de 0 à 3



https://martinfowler.com/articles/richardsonMaturityModel.html

Le modèle de maturité de Richardson

Level 0 : The swamp of POX (Poor old XML)

Une API de niveau 0 n’est pas une API REST (peut-être du SOAP).

Par exemple :

- Un seul point d’entrée par exemple /api
- Une seule méthode HTTP (GET le plus souvent)

Le modèle de maturité de Richardson

Level 1 : Resources

Séparation des ressources : les URI doivent correspondre à la ressource demandée. Par exemple :

- **/users** pour une liste de users
- **/users/create** pour créer un utilisateur
- **/users/14** pour un seul utilisateur
- **/users/14/update** pour mettre à jour un utilisateur

Le modèle de maturité de Richardson

Level 2 : HTTP verbs

Utilisation des méthodes HTTP et des retours HTTP en fonction de l'action demandée. Penser pour cela au CRUD (Create Read Update Delete).

Create : POST /users

Read : GET /articles

Update : PUT /users/14

Delete : DELETE /users/14

Le modèle de maturité de Richardson

Level 3 : Hypermedia controls

Intégration des liens hypermedia permettant de “se rapprocher” d’une page web classique. Ex pour la route **/users** :

```
{
    "uid" : 123,
    "firstName" : "Jean",
    "lastName" : "Martin",
    "links" : {
        "public" : "https://monsite.com/user/123",
        "avatar" : "https://monsite.com/img/12fgh52s.jpg"
    }
},
{
    "uid" : 228,
    "firstName" : "Lucie",
    "lastName" : "Dupuis",
    "links" : {
        "public" : "https://monsite.com/user/228",
        "avatar" : "https://monsite.com/img/sd412fhf.jpg"
    }
}
```

Evaluation - Projet technique - MAJ

Webservice REST - Le CRM

Ajout d’une propriété photos pour les clients.

Un client peut avoir une ou plusieurs photos. Ces photos seront CRUD* par des administrateurs.

* CRUD = Create Read Update Delete

Evaluation - Projet technique - MAJ

Webservice REST - Le CRM

Envoi d’un email à l’administrateur au moment de son inscription.

L’email doit contenir un lien vers la doc de l’api pour lui permettre de l’utiliser.

La réponse de l’API ne doit pas être dépendante de l’envoi du mail.