RESTful API Design

Brève introduction

Adrien CAUBEL

Ressource :
- https://readthedocs.org/projects/apiguide/downloads/pdf/latest/

Introduction

Pourquoi utiliser une API ?

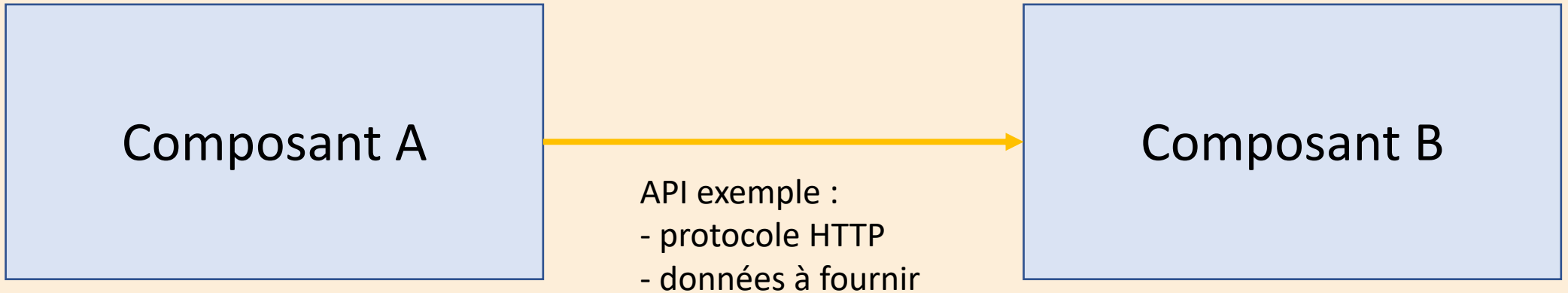
Question :

A votre avis ?

- **Interopérabilité** : Les API permettent à différents systèmes logiciels de communiquer entre eux. Elles offrent aux applications un moyen normalisé d'interagir, ce qui permet une intégration et un échange de données sans faille.
- **Modularité et abstraction** : Les API permettent aux développeurs de décomposer des systèmes complexes en éléments plus petits et plus faciles à gérer. Cette approche modulaire favorise la réutilisation et facilite la maintenance en abstrayant les fonctionnalités sous-jacentes.
- **Compatibilité multiplateforme :** Les API facilitent la création de logiciels qui fonctionnent sur différentes plateformes (web, mobile, bureau) en fournissant une interface cohérente pour la communication.
- **Monétisation et opportunités commerciales** : peut être vendu, offerte à un client. Une API prend de la valeur lorsqu'elle devient un canal offrant de nouveaux modes d'accès à la valeur déjà proposée par une entreprise.

Définition API

Une API sert d'intermédiaire pour permettre à différentes applications logicielles de communiquer entre elles. Elle définit les méthodes et les protocoles que les développeurs peuvent utiliser pour interagir avec un composant logiciel, un service ou une plateforme



Histoire

API SOAP : permettant la communication entre des systèmes distribués sur le web.
Cette époque a également vu l'introduction de XML-RPC.

API REST : s'est imposé comme un style architectural pour la conception d'applications en réseau. Les API RESTful, qui utilisent le protocole HTTP (GET, POST, PUT, DELETE), ont été largement adoptées en raison de leur simplicité et de leur évolutivité.

OpenAPI spécification (swagger) : essayer de fournir une documentation standart pour les API REST

GraphQL : une alternative aux API REST

RESTful API Design

Dans cette introduction au API Design nous n’évoquerons que les API Rest

A quoi ça ressemble ? Au tableau

Question :

« récupérer l'ensemble des étudiants de l'iut »

« récupérer l'étudiant ayant l'id 1 »

« récupérer l'ensemble des étudiants de 3ème année »

« récupérer l'ensemble des étudiants de 3ème année informatique »

« mettre à jour l'ensemble des étudiants de 3ème année informatique nommés toto avec un nouveau nom titi »

A quoi ça ressemble ? Au tableau

- GET /iut/v1/etudiants
- GET /iut/v1/etudiants/1
- GET /iut/v1/edutiants?annee=3
- GET /iut/v1/edutiants?annee=3&filiere=informatique (filtre)

- PUT /iut/v1/edutiants/ + corps de requête en json

```
{
   name: toto,
   annee: 3
   filiere: informatique,
   nouveauNom: titi
}
```

Protocole HTTP

Ce base sur le protocole réseau HTTP

Utilise les méthode CRUD du protocole HTTP

- GET retourner une ressource
- POST créer une ressource
- PUT mettre à jour une ressource
- DELETE supprimer une ressource

Protocole HTTP : retour

Ce base sur le protocole réseau HTTP

Utiliser les codes d'état HTTP appropriés pour indiquer l'état d'une demande :

- 200 OK : La demande a abouti.
- 201 Created : La ressource a été créée avec succès.
- 404 Not found : La ressource n'a pas été trouvée.
- 400 Bad Request : Format de requête non valide.
- 401 Unauthorized : Accès non autorisé.
- 500 Internal Server Error : Erreur côté serveur, etc.

- **Si quelque chose alors format JSON**

Versionnage

L’avantage est que nous pouvons travailler sur de nouvelles fonctionnalités ou améliorations d'une nouvelle version pendant que les clients utilisent toujours la version actuelle et ne sont pas affectés par des changements radicaux.

- /api/**v1**/etudiants
- /api/**v2**/etudiants

Bonnes pratiques

Bonnes pratiques

- URLs doivent inclure des noms et non des verbes
  - Api/v1/~~getEtudiant~~ --> api/v1/etudiant
  - Api/v1/updateEtudiant --> api/v1/etudiant

Question :

Hum … mais c'est pareil ? Comment faire la différence ?

- Utiliser des nom au pluriel
  - api/v1/etudiant**s**

- Utiliser les statuts de réponse HTTP
  - 200, 505, etc …

Profondeur maximale

Eviter les urls plus profonde que
`resource/identifier/resource`.

All articles in (or belonging to) this magazine:

```
GET /api/v1/magazines/1234/articles.json HTTP/1.1
Host: www.example.gov.au
Accept: application/json, text/javascript
```

Quelques exemples

List of magazines:

```
GET /api/v1/magazines.json HTTP/1.1
Host: www.example.gov.au
Accept: application/json, text/javascript
```

All articles in (or belonging to) this magazine:

```
GET /api/v1/magazines/1234/articles.json HTTP/1.1
Host: www.example.gov.au
Accept: application/json, text/javascript
```

Filtering and sorting are server-side operations on resources:

```
GET /api/v1/magazines.json?year=2011&sort=desc HTTP/1.1
Host: www.example.gov.au
Accept: application/json, text/javascript
```

Ressource/identifiant/ressource

Quelques mauvais exemple

Non-plural noun:

```
GET /magazine HTTP/1.1
Host: www.example.gov.au
Accept: application/json, text/javascript
```

```
GET /magazine/1234 HTTP/1.1
Host: www.example.gov.au
Accept: application/json, text/javascript
```

Verb in URL:

```
GET /magazine/1234/create HTTP/1.1
Host: www.example.gov.au
Accept: application/json, text/javascript
```

Filter outside of query string:

```
GET /magazines/2011/desc HTTP/1.1
Host: www.example.gov.au
Accept: application/json, text/javascript
```