Exercices API RESTful, avec node.js et Express.js

Paul Schuhmacher

Mars 2023

Module: API

Exercice 1: Contraintes REST, fondamentaux

Manipuler le cache HTTP et lire les *code status*

1. **Effectuer** une requête pour demander la ressource pointée par `/`. Faire la même chose en demandant la ressource sur l'URL `/foo`. **Noter** les *code status*. Requêter à nouveau `/`. **Observer** le *code status*. Que remarquez-vous ?

2. **Servir** un fichier statique (`index.html`) sur l'URL `/`, situé à la racine du site web dans un dossier `root`. Pour cela utiliser les fonctions middleware d'Express.

3. **Parcourir** la page HTTP caching de la MDN qui synthétise la RFC9111 - HTTP Caching (2022) du protocole HTTP/1.1. **Formuler** une synthèse (sous la forme qui vous convient) des différents types de cache du protocole HTTP. **Étudier** les concepts de cache *fresh* (frais) et *stale* (vicié) et de *validation*.

4. **Requêter** à nouveau l'URL et observer le contenu de la requête HTTP, notamment l'en-tête `Cache-Control`. Que constatez-vous ?

1. **Modifier** le fichier `index.html` (par exemple le titre), et effectuer à nouveau une requête HTTP. Que se passe-t-il ?

6. **Forcer** le rechargement de la page (souvent `Ctr+Maj+r`) et **observer** la valeur de l'en-tête `Cache-Control` dans la requête.

Exercice 2 - Design d'une API RESTful

On désire mettre en ligne un service de réservation de billets de concert. Le service ne gère pas de base de données des utilisateurs : un·e utilisateur·ice est simplement identifié·e par un pseudo au moment de la réservation.

Les cas d'utilisation définis sont :

1. L’utilisateur·ice consulte la liste des concerts disponibles
2. L’utilisateur·ice consulte les informations d’un concert
3. L’utilisateur·ice réserve une place pour un concert avec un pseudo
4. L’utilisateur·ice annule sa réservation
5. L’utilisateur·ice confirme sa réservation
6. Le gestionnaire du site consulte la liste des réservations confirmées pour un concert.

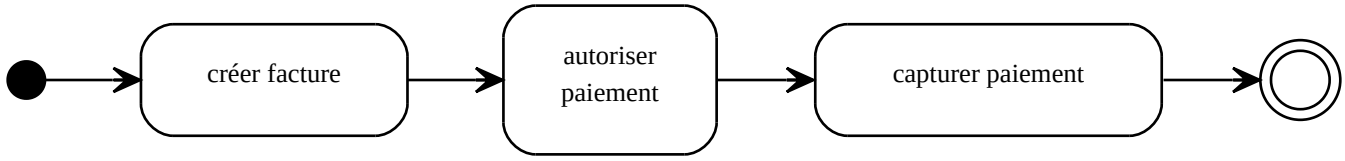
Attention, **un utilisateur qui a confirmé sa réservation ne peut plus l’annuler !**

Décrire une API Web RESTful **par des exemples de requêtes/réponses HTTP** permettant de réaliser les cas d’utilisation ci-dessus.

> *Sans utiliser Chat GPT…*

Exercice 3 - Design d’une API RESTful

Étant donné le workflow suivant décrivant le paiement d’une facture :



Les activités sont :

- *1. Créer facture* Création d’un document identifiant les coordonnées bancaires du créditeur et le montant à payer.
- *2. Autoriser paiement* Le débiteur indique ses informations de paiement (numéro de carte de paiement, date limite de validité de la carte et cryptogramme de sécurité). Le client effectue une demande auprès de l’organisme bancaire pour savoir si la transaction est autorisée. L’organisme bancaire fournit un numéro d’autorisation permettant d’identifier la transaction.
- *3. Capturer paiement* Si l’autorisation s’est bien passée, le client réalise une capture, c’est-à-dire qu’il demande à l’organisme bancaire d’enregistrer le paiement et d’effectuer le transfert du compte du débiteur vers le compte du créditeur.
- *Décrire* également des scénarios alternatifs :
    - L’autorisation échoue car les informations bancaires fournies sont incorrectes
    - L’autorisation échoue car le client ne dispose pas du crédit suffisant sur son compte.

Décrire la réalisation d’une API Web RESTful **par des exemples de requêtes/réponses HTTP**.

Exercice 4 - Implémentation d'une API RESTful

1. **Implémenter** l'API de l'exercice 2 avec node.js et Express.js. **Concevoir** et utiliser le schéma de base de données relationnelles.
2. **Développer** un ensemble de *ressources* pour qu'un agent *humain* puisse réaliser les cas d'utilisation exposés par l'API (via des pages web)

   Pour réaliser cet exercice, utiliser le starter-pack mis à votre disposition à cette adresse.

Exercice 5 - Implémentation d'une API RESTful avec authentification et autorisation par JWT

1. **Implémenter** l'API de l'exercice 3 avec node.js et Express.js. **Concevoir** et utiliser le schéma de base de données relationnelles.
2. **Développer** un ensemble de *ressources* pour qu'un agent *humain* puisse réaliser les cas d'utilisation exposés par l'API (via des pages web)
3. **Mettre en place un système d'authentification et d'autorisation** par **JSON Web Token**. On créera le compte utilisateur à la main *directement en base* via une requête SQL (inutile d'exposer une ressource */sign-up* sur l'API pour le faire)

   Pour réaliser cet exercice, utiliser le starter-pack mis à votre disposition à cette adresse.

Exercice 6 - API RESTful de carnet d'adresses

Cet exercice est tiré de l'ouvrage Bien architecturer une application REST (voir ressoures)

Notre application de carnet d'adresses va proposer un service que des clientsREST pourront utiliser. Par client REST, nous entendons un programme, écrit dans un langage quelconque, qui interrogera des URL via le protocole HTTP pour accéder aux données du carnet d'adresses, dans un format à définir (HTML, XML, etc.).

Du côté des fonctionnalités, au strict minimum, il faudrait pouvoir **lire** une carte de visite qui y sera stockée, et bien évidemment en **ajouter** ou **compléter** les renseignements saisis. Il serait bon également de pouvoir **regrouper** les fiches par groupes (professionnel, amis, vie courante), y faire des **recherches**, **exporter** ou **importer** des fiches dans d'autres formats, etc.

1. **Identifier** les ressources.
2. **Proposer** une URL qui identifie la ressource *le carnet d'adresses d'Olivier*
3. **Représenter** le *ressource state* sous forme d'un graphe (l'ensemble des ressources disponibles via leurs URL) (cf cours)
4. **Écrire les pseudo-requêtes/réponses HTTP (avec les code status)** pour

1. Acceder à une carte du carnet, par exemple celle de Alexandre Dumas
2. Acceder a un groupe de fiches, par exemple le groupe *amis*
3. Créer une nouvelle carte dans le carnet
4. Modifier une fiche
5. Supprimer un groupe
6. Supprimer un groupe qui n'existe pas
7. Créer une carte avec une représentation incompréhensible
8. Erreur du serveur lors de la demande de la carte d'Alexandre Dumas (par exemple limite de place sur l'espace disque)

5. **Implémenter** l'API avec node.js et express.js.

Annexe: node.js et express.js, Getting started

```
npm init
npm install express --save
```

Créer le code pour démarrer un serveur

```
// index.js
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
    res.send('Hello World!')
})

app.listen(port, () => {
      console.log(`Demo REST, servie à l'url: http://localhost:${port}`)
})
```

Lancer le serveur

```
node index.js
```

Ressources

Node, Express et libs

- Source des exercices
- nodejs
- Express, framework web minimal pour les applications node.js
- Express, *Hello world*
- Générateur d'application Express
- Routage Express
- En-tête HTTP `Cache-Control`
- Cache headers in Express js app, un bon article qui explique la gestion du cache dans des applications express

- Express, static files
- Express, static files,demo
- Express: meilleures pratiques en production : performances et fiabilité
- mysql.js: Escaping query values

Protocole HTTP

- HTTP Code status 304
- HTTP Caching, une synthèse sur l'implémentation du cache du protocole HTTP. Attention tous les navigateurs n'implémentent pas le standard au même point.
- Un tutoriel de la mise en cache, un très bon tutoriel en français sur la mise en cache du protocole HTTP

Design API

- REST APIs must be hypertext-driven, billet de blog de Roy T. Fiedling très intéressant sur le fait qu'une API RESTful doit être orientée *hypertexte* (ou de manière générale par les *hypermédia*). Concepts fondamentaux à suivre.
- JSON Hypertext Application Language draft-kelly-json-hal-08, HAL representation pour les modèles de données. Une proposition de standard
- API RESTful, spécification des schémas de données *HAL*, les différents types d'hypermédia définis pour le protocole HTTP et pour construire des API plus robustes. Le livre de l'auteur Building Hypermedia APIs with HTML5 and Node, Amundsen, a l'air très intéressant
- API RESTful, spécification des schémas de données *JSON-LD 1.1, A JSON-based Serialization for Linked Data*, une autre spécification des données renvoyées par une API, soutenue et recommandée par le W3C
- Schema.org, *Schema.org is a collaborative, community activity with a mission to create, maintain, and promote schemas for* structured data* on the Internet*. Propose une liste de schémas à suivre pour différents modèles de données
- Microformats wiki, un wiki qui décrit des specifications de structure de données interopérables
- Hydra, Hydra is an effort to simplify the development of interoperable, hypermedia-driven Web APIs. The two fundamental building blocks of Hydra are JSON-LD and the Hydra Core Vocabulary.
- Zalando RESTful API and Event Guidelines
- Bien architecturer une application REST, par Olivier Gutknecht, avec la contribution de Jean Zundel, Eyrolles, 2009

JWT

- JSON Web Token (JWT), la rfc du standard
- RFC 9068: JWT Profile for OAuth 2.0 Access Tokens
- Introduction to JSON Web Tokens, une introduction aux JWT
- Décoder le JWT, une application web pour décoder le contenu d'un JWT

Conception base de données relationnelles

- Le Dictionnaire de Données, établir un dictionnaire de données est une étape fondamentale de tout travail de conception d’une base de données