

Concevoir une API

Table des matières

I. Découverte de Node.js	3
II. Exercice : Appliquer la notion	8
III. Concevoir une API REST avec Express.js	9
IV. Exercice : « La gestion des utilisateurs »	14
V. Mise en ligne de son API REST	14
VI. Exercice : Appliquer la notion	21
VII. Essentiel	21
VIII. Auto-évaluation	21
A. Exercice	21
B. Test	22
Solutions des exercices	23

I. Découverte de Node.js

Durée : 1 h

Environnement de travail : Windows / (Mac OSX)

Pré-requis :

- Bases en REST
- Bases en Git
- Bases en JavaScript (idéalement bases de Node.js)

Contexte

Les API REST ont de très nombreuses applications de nos jours, au point que cette compétence est devenue un pré-requis pour toute personne souhaitant faire carrière dans le développement web.

Tous les langages informatiques ont les fonctionnalités nécessaires pour la réalisation d'API REST.

Nous le verrons, concevoir une API REST n'est pas bien différent de développer une application serveur !

Une fois la requête HTTP reçue par notre application, il faudra l'interpréter et retourner la réponse adaptée.

Dans ce cours, nous allons devoir installer Node.js. C'est ce qui nous permettra d'utiliser le langage JavaScript en dehors du navigateur web, et de l'héberger sur un serveur !

Ensuite, nous allons développer une API REST complète : définition des ressources, des fonctionnalités, mais également la définition du format des données retournées.

Enfin, dans la dernière partie de ce cours nous mettrons en ligne notre application sur Heroku.

Objectifs

- Installation de Node.js
- Premiers scripts
- Installer des dépendances

Contexte

Impossible de faire une carrière dans le web sans connaître un peu de JavaScript, n'est-ce pas ?

Mais le vrai problème de JavaScript, c'est que c'est un langage de programmation « *front* », qui ne peut pas être utilisé en dehors d'un navigateur web. Il y a quelques années, les développeurs et développeuses web devaient alors apprendre un langage serveur comme PHP, Java ou Python pour développer leurs applications côté serveur.

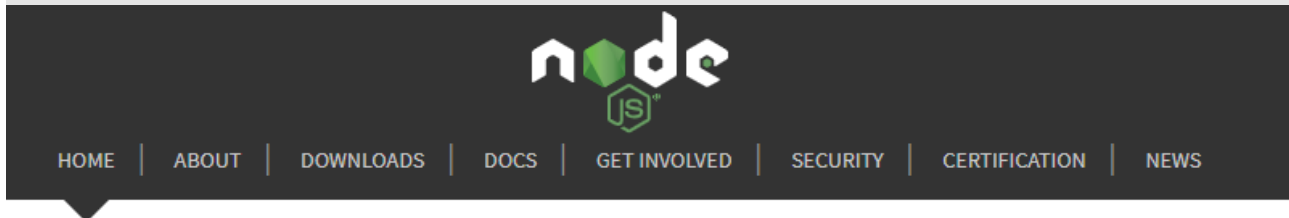
Mais en 2009, lors d'une conférence technique un développeur présente une plateforme capable d'exécuter JavaScript en dehors d'un navigateur web. Il fait alors la démonstration devant une salle ébahie de la première API REST développée à l'aide de JavaScript, c'est-à-dire capable de répondre à la requête d'un utilisateur en passant par la couche HTTP.

Aujourd'hui, Node.js est utilisé par des millions de développeurs et des dizaines de milliers d'entreprises dans le monde. Dans cette partie, nous allons installer Node.js et apprendre à installer des bibliothèques fournies par la communauté.

Méthode

Installation de Node.js et npm

Pour installer Node.js, il faut commencer par aller sur leur site nodejs.org¹ et télécharger une version en fonction de votre système d'exploitation (Windows, Mac OSX ou même GNU/Linux) :



Node.js® is a JavaScript runtime built on [Chrome's V8 JavaScript engine](#).

#BlackLivesMatter

New security releases now available for 15.x, 14.x and 12.x release lines

Download for Windows (x64)

14.15.1 LTS

Recommended For Most Users

15.3.0 Current

Latest Features

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

[Other Downloads](#) | [Changelog](#) | [API Docs](#)

Or have a look at the [Long Term Support \(LTS\) schedule](#).

Comme vous pouvez le voir, l'éditeur de Node.js vous laisse le choix entre 2 versions :

- Une version « *LTS* »
- Une version « *Current* »

La version « *LTS* » pour « *Long Time Survey* » est une version de production qui est recommandée pour la plupart des utilisateurs et qui est maintenue un peu plus longtemps.

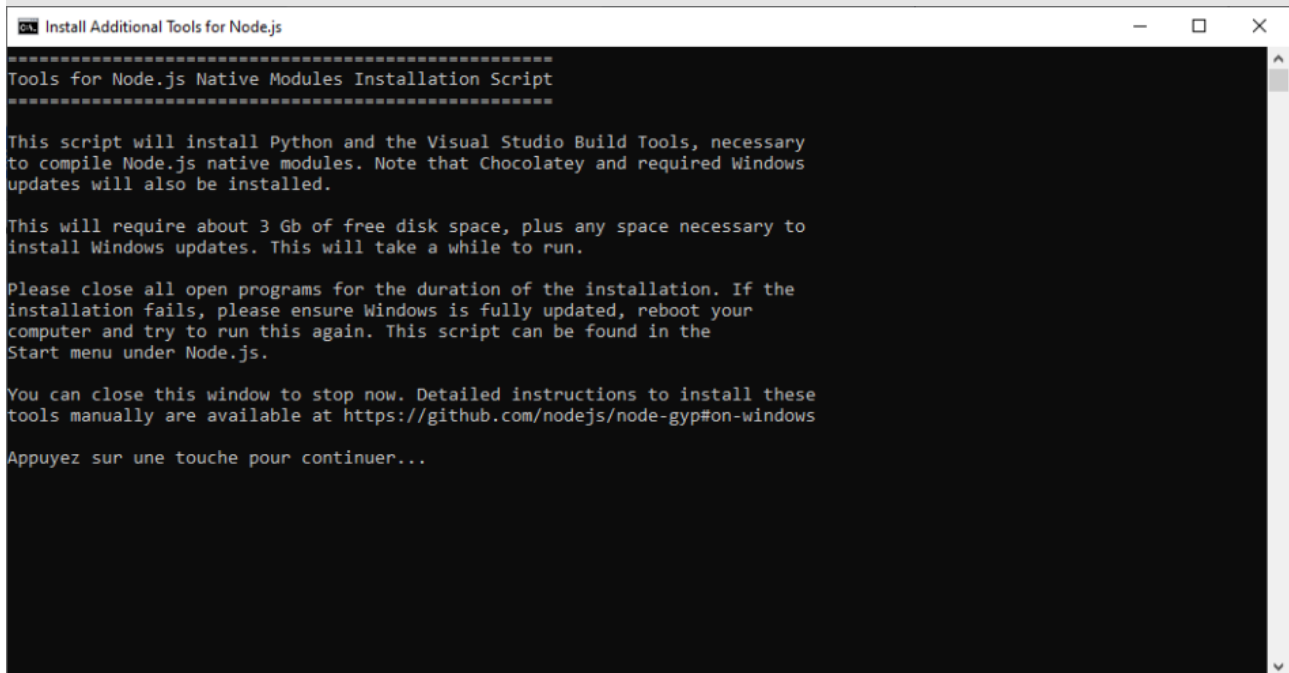
A contrario, la version « *Current* » contient toutes les dernières nouveautés, mais n'est pas maintenue très longtemps.

Vous allez donc sélectionner la version LTS, quel que soit son numéro de version.

Le processus d'installation ne contient aucune difficulté particulière : il faut accepter (cocher) tout ce qui est proposé.

¹ <https://nodejs.org/en/>

Sur Windows, il est possible qu'à la fin de l'installation l'invite de commandes s'ouvre :



```
Install Additional Tools for Node.js

Tools for Node.js Native Modules Installation Script

This script will install Python and the Visual Studio Build Tools, necessary
to compile Node.js native modules. Note that Chocolatey and required Windows
updates will also be installed.

This will require about 3 Gb of free disk space, plus any space necessary to
install Windows updates. This will take a while to run.

Please close all open programs for the duration of the installation. If the
installation fails, please ensure Windows is fully updated, reboot your
computer and try to run this again. This script can be found in the
Start menu under Node.js.

You can close this window to stop now. Detailed instructions to install these
tools manually are available at https://github.com/nodejs/node-gyp#on-windows

Appuyez sur une touche pour continuer...
```

C'est parfaitement normal. Appuyez sur la touche « Entrée » et laissez le processus d'installation se terminer.

Cette étape va installer des outils et logiciels nécessaires à l'exécution de node.js et npm sur Windows.

Une fois l'installation terminée, ouvrez l'invite de commandes (votre terminal) et tapez l'instruction suivante :
node.

Puis le code JavaScript suivant :

```
console.log("JavaScript dans l'invite de commandes !");
```

Et appuyez sur la touche « Entrée » : félicitations, vous avez exécuté du code JavaScript en dehors de votre navigateur !

Maintenant, à l'aide de votre éditeur de code favori, copiez le contenu de cette instruction dans un fichier appelé `index.js`. Puis dans votre invite de commandes, tapez l'instruction suivante :

```
node index.js
```

Ci-dessous, le résultat obtenu à l'aide de mon éditeur de code favori (Visual Studio Code) :

The screenshot shows the Visual Studio Code interface. The editor window displays a file named `index.js` with the following code:

```
1 console.log('Hello world');
```

Below the editor is a terminal window with the following output:

```
micka@DESKTOP-D0QDBAR MINGW64 ~/Projects/conception-apis
$ node index.js
Hello world

micka@DESKTOP-D0QDBAR MINGW64 ~/Projects/conception-apis
$
```

Vous l'aurez compris, Node.js est capable d'exécuter des fichiers JavaScript.

Si l'ambition de ce cours n'est pas de vous apprendre Node.js, nous allons devoir aller un tout petit peu plus loin et découvrir NPM (Node.js Packages Manager), l'installateur de dépendances de Node.js.

Installation de dépendances avec NPM

L'installation et la gestion des dépendances sont des fonctionnalités importantes pour la maintenance des applications. Que vous soyez actuellement dans un parcours plutôt PHP, Python, Java ou JavaScript toutes ces plateformes disposent d'un gestionnaire de dépendances.

Composer¹ pour PHP, pip² pour Python, Maven³ pour Java, etc. Le gestionnaire de dépendances de JavaScript le plus populaire, c'est npm⁴ !

1 <https://getcomposer.org/>

2 <https://pypi.org/project/pip/>

3 <https://maven.apache.org/>

4 <https://www.npmjs.com/>

Comment installer une dépendance ?

Avant cela, nous allons devoir créer un fichier de configuration qui va conserver une trace des dépendances que nous allons utiliser.

Tapez l'instruction suivante **dans le dossier qui contient le fichier JavaScript sur lequel nous travaillons** : `npm init -y`.

Cette instruction exécute un logiciel qui va générer le fichier de configuration `package.json` à côté de votre fichier `index.js`.

Si vous souhaitez en savoir plus sur ce fichier de configuration et cette commande, consultez la documentation de `npm`¹.

S'il y a une librairie JavaScript que vous souhaitez installer dans votre projet, il y a de fortes chances qu'elle soit disponible sur `npm`. Pour le vérifier, vous pouvez aller sur le site de `npm`² et taper le nom de la librairie dans le moteur de recherche. Voici le résultat pour la librairie de gestion de dates « `moment.js` »³ :

The screenshot shows the npm package page for 'moment'. At the top, it displays 'moment' with version '2.29.1', 'Public' status, and 'Published 2 months ago'. Below this are links for 'Readme', 'Explore' (with a 'BETA' tag), '0 Dependencies', '47892 Dependents', and '71 Versions'. The main section features the 'Moment.js' logo and a series of status badges: 'npm v2.29.1', 'downloads 69M/month', 'license MIT', 'build passing', 'coverage 89%', 'license scan passing', and 'server stability 96%'. A description states: 'A JavaScript date library for parsing, validating, manipulating, and formatting dates.' Below this is the 'Project Status' section, which notes that Moment.js is a legacy project in maintenance mode. On the right side, there is an 'Install' section with a code block showing '> npm i moment'. Below that is a 'Weekly Downloads' chart showing 15,532,610 downloads. A table provides further details: Version 2.29.1, License MIT, Unpacked Size 4.21 MB, Total Files 533, Issues 67, and Pull Requests 25.

Pour installer cette dépendance, utilisez l'instruction suivante **dans le dossier qui contient le fichier JavaScript sur lequel nous travaillons** :

```
npm i moment
```

`npm` va installer la librairie `moment` et télécharger les fichiers dans un dossier appelé `node_modules` : c'est ce dossier qui contiendra toutes les dépendances de votre projet Node.js.

Comment utiliser une dépendance que nous venons d'installer ?

C'est tout l'intérêt de Node.js ! Il nous suffira de faire appel à la fonction `require()` qui va importer la dépendance à l'aide de son nom.

Dans le fichier `index.js`, écrivez le code JavaScript suivant :

```
1 var moment = require('moment');  
2 console.log(moment().format('D/M/y H:m'));
```

Puis exécutez à nouveau ce script « `node` » dans l'invite de commandes :

```
node index.js
```

1 <https://docs.npmjs.com/creating-a-package-json-file>

2 <https://www.npmjs.com/>

3 <https://www.npmjs.com/package/moment>

```
micka@DESKTOP-D0QDBAR MINGW64 ~/Projects/conception-apis
$ node index.js
26/11/2020 7:52
```

Super ! Comme vous l'espériez, vous avez bien retrouvé un objet JavaScript « *Moment* » qui a « *formaté* » la date du jour.

Si vous ne connaissez pas cette librairie¹, elle est très utilisée pour la gestion d'applications destinées à l'international (gestion des dates et fuseaux horaires notamment).

Exemple

```
1 // Dans le terminal
2 > npm init
3 > npm i moment
4
5 // Dans un fichier "index.js"
6 var moment = require('moment');
7 console.log(moment().format('D/M/y H:m'));
8 // Dans le terminal
9 > node index.js
10 26/11/2020 7:52
```

Remarque Syntaxe « à retenir »

Node.js est une plateforme logicielle qui permet d'exécuter du code JavaScript en dehors du navigateur web. Il est installé avec **npm** qui permet d'installer des dépendances.

Mais pour cela, vous avez dû créer un fichier `package.json` à l'aide de l'instruction `npm init`.

Une fois ce fichier créé, on peut aller sur le site de npm² et rechercher les librairies dont vous pourriez avoir besoin.

Une fois la librairie trouvée, on l'installe à partir de son nom avec la commande `npm i` suivi du nom de la librairie : cette instruction se retrouve sur le site de npm et vous pouvez la recopier.

Complément

Node.js³

II. Exercice : Appliquer la notion

La popularité du langage JavaScript et de Node.js font qu'il existe énormément de librairies open source et gratuites pour vous aider dans vos développements.

¹ <https://momentjs.com/>

² <https://www.npmjs.com/>

³ <https://nodejs.org/en/>

Question

[solution n°1 p.25]

Vous devez réaliser un script qui prend un fichier Markdown¹ et le convertir en HTML.

Indice :

Il y a probablement une librairie JavaScript qui est capable de faire cela : pourquoi ne pas l'installer et l'utiliser ? Pour vérifier votre application, utilisez le contenu d'exemple suivant :

```
# Marked in Node.js\n\nRendered by **marked**.
```

III. Concevoir une API REST avec Express.js

Objectifs

- Définir les ressources de l'API
- Définir le format de l'API
- Définir les fonctionnalités de l'API (CRUD)

Contexte

Il existe de très nombreuses librairies dans l'écosystème Node.js pour développer très rapidement des applications REST. Mais quand on demande à des développeurs et développeuses la première qu'ils ont en tête, ils répondent spontanément Express.js !

La réalisation d'une API REST ne vous demandera que quelques lignes de code, ce qui vous laissera plus de temps pour bien réfléchir aux **ressources**, au **format** et aux **fonctionnalités** proposés par celle-ci.

Au travers de la réalisation d'une API de gestion de livres en librairie, nous allons aborder ces concepts.

Ressources

Quelles données doit gérer votre API REST ? S'agit-il d'une gestion de produits, d'utilisateurs ?

Comment sont structurées ces données **entre elles** ?

Ces questions vous permettront de définir quelles ressources doivent être disponibles (on dit « *exposées* ») par l'API REST.

Pour la gestion d'une librairie, nous avons au minimum besoin de gérer **des livres** :

- Titre,
- Auteur,
- Identifiant ISBN,
- Disponibilité,
- Date de retour si emprunté.

Et **des utilisateurs** :

- Nom,
- Prénom,
- Email,
- Livres empruntés.

Format

La différence entre une bonne et une mauvaise API tient dans *la communication* entre ceux qui la développent et ceux qui l'utilisent (ou vice-versa !).

¹ <https://fr.wikipedia.org/wiki/Markdown>

Une API doit être facile à manipuler. Une bonne expérience pour se rendre compte de la qualité d'une API REST est de développer l'application JavaScript cliente de cette API, c'est-à-dire qui l'utilise pour accéder aux ressources et aux fonctionnalités.

Enfin, il faudra aussi faire le choix du type de format à retourner entre JSON, HTML et XML selon le besoin de vos utilisateurs.

Dans la pratique, le JSON est utilisé par la très grande majorité des API REST.

Quelques bonnes pratiques :

- Chaque ressource a son **propre URI** et une structure **spécifique**.
- Les listes de ressources contiennent **toutes les propriétés indispensables pour les besoins les plus demandés** (nécessitent parfois d'améliorer l'API après l'avoir mise en ligne).
- Si une ressource est liée à une autre, il est plus intéressant, dans un premier temps, d'ajouter cette autre ressource (même si ça peut poser des problèmes de performance en cas de gros trafic).

Passons à la pratique, en revenant sur notre application de gestion de livres.

Si nous devons retourner un livre, un bon format de retour serait celui-ci :

```
1 {
2   "titre": "Nom du livre",
3   "auteur": "Nom de l'auteur",
4   "isbn": "Une clé unique",
5   "disponibilité": true/false,
6   "date_retour": null ou une date
7 }
```

En admettant que nous n'allons pas gérer la disponibilité d'un bouquin en considérant une liste entière, le retour de l'API pour la liste des livres de la librairie aurait plutôt ce format-là :

```
1 [
2   {
3     "titre": "Livre 1",
4     "auteur": "Jonathan",
5     "isbn": "12345679",
6     "disponibilite": true,
7     "date_retour": null
8   },
9   {
10    "titre": "Livre 2",
11    "auteur": "Elise",
12    "isbn": "98765428",
13    "disponibilite": false,
14    "date_retour": "01/01/2028"
15  },
16  {
17    "titre": "Livre 3",
18    "auteur": "Zineb",
19    "isbn": "87357493",
20    "disponibilite": false,
21    "date_retour": null
22  }
23 ]
```

De cette façon, les personnes qui récupèrent ce contenu peuvent le manipuler facilement avec le langage JavaScript.

Fonctionnalités

Que fournit votre API ? Est-ce qu'il s'agit seulement de récupérer et pouvoir modifier des informations en base de données (ce que l'on appelle parfois un « *CRUD* »), ou a-t-elle des fonctionnalités particulières ?

Pour l'application d'exemple sur laquelle nous sommes en train de travailler, voici des fonctionnalités que nous pourrions considérer :

- Lister un livre, des livres,
- Lister un utilisateur, des utilisateurs,
- Créer un livre,
- Supprimer un livre,
- Changer la disponibilité d'un livre et sa date de retour,
- Créer un utilisateur,
- Modifier un utilisateur,
- Supprimer un utilisateur,
- Etc.

Dans cette partie, nous allons voir comment développer certaines de ces fonctionnalités à l'aide d'Express.js.

Méthode

Express.js¹ est un Framework (un ensemble de bibliothèques et modules) adapté à la réalisation d'applications REST.

Il est composé d'un **serveur**, d'un routeur² et d'un ensemble d'autres modules que nous n'aborderons pas dans ce cours.

Le module serveur va permettre à votre code JavaScript d'être accessible au travers d'un navigateur web, mais cette fois côté « serveur » justement, quand le « routeur » va vous permettre de définir les points d'entrée de votre API.

La première chose à faire est d'installer express.js !

Pour cela, utilisez l'instruction suivante **dans le dossier qui contient le fichier JavaScript sur lequel nous travaillons** :

```
npm i express
```

Et changez le contenu du fichier `index.js` par le contenu suivant :

```
1 const express = require('express')
2 const app = express()
3 const port = 3000
4
5 app.get('/', (req, res) => {
6   res.send('Hello World!')
7 })
8
9 app.listen(port, () => {
10   console.log(`Serveur [OK]`)
11 })
```

Ensuite, exécutez le script (pour rappel : `node index.js` dans le terminal) et accédez cette fois-ci à votre navigateur sur l'URL suivante : `http://localhost:3000`³

Le message « *Hello World !* » s'affiche dans votre navigateur !

¹ <https://expressjs.com/fr/>

² <https://expressjs.com/fr/starter/basic-routing.html>

³ <http://localhost:3000/>

Quelques explications pour bien comprendre ce code d'exemple :

- Express a donc une fonction `get()` qui prend en paramètre une URL et permet de retourner une réponse HTTP à l'aide de la fonction `send()` de la variable `res`,
- Les variables `res` et `req` sont respectivement la requête et la réponse HTTP,
- La fonction `listen()` d'Express est ce qui rend notre application disponible dans le navigateur !

Le Framework Express a une fonction pour chaque méthode HTTP¹ :

- `get()` pour écouter les requêtes HTTP GET
- `post()` pour écouter les requêtes HTTP POST
- `put()` pour écouter les requêtes HTTP PUT
- `delete()` pour écouter les requêtes HTTP DELETE

Enfin, pour retourner une réponse au format HTTP, nous pouvons utiliser la méthode `json()` de l'objet `res`.

Après consultation de la documentation d'Express.js², pour la gestion des livres nous parvenons au code complet et fonctionnel ci-dessous :

Exemple

```
1 const express = require('express')
2 const app = express()
3 const port = 3000
4 app.use(express.json())
5
6 // Une liste d'exemple de livres, qui pourrait être récupérée en base de données
7 const books = [
8   {
9     "titre": "Livre 1",
10    "auteur": "Jonathan",
11    "isbn": "12345679",
12    "disponibilite": true,
13    "date_retour": null
14  },
15  {
16    "titre": "Livre 2",
17    "auteur": "Elise",
18    "isbn": "98765428",
19    "disponibilite": false,
20    "date_retour": "01/01/2028"
21  },
22  {
23    "titre": "Livre 3",
24    "auteur": "Zineb",
25    "isbn": "87357493",
26    "disponibilite": false,
27    "date_retour": null
28  },
29 ]
30
31 app.get('/books/', (req, res) => {
32   res.json(books)
33 })
```

¹ <https://expressjs.com/fr/guide/routing.html>

² <https://expressjs.com/fr/>

```
34
35 app.get('/books/:isbn', (req, res) => {
36   const isbn = req.params.isbn
37   const book = books.find(book => book.isbn === isbn)
38
39   res.json(book)
40 })
41
42 app.post('/books', (req, res) => {
43   books.push(req.body)
44   res.status(200).json(books)
45 })
46
47 app.put('/books/:isbn', (req, res) => {
48   const isbn = req.params.isbn
49   let book = books.find(book => book.isbn === isbn)
50
51   // Modification du livre
52   book.titre = req.body.titre
53   book.auteur = req.body.auteur
54   book.disponibilite = req.body.disponibilite
55   book.date_retour = req.body.date_retour
56
57   res.status(200).json(book)
58 })
59
60
61 app.delete('/books/:isbn', (req, res) => {
62   const isbn = req.params.isbn
63   const book = books.find(book => book.isbn === isbn)
64   books.splice(books.indexOf(book), 1)
65
66   res.json(books)
67 })
68
69 app.listen(port, () => {
70   console.log('Serveur démarré')
71 })
```

Attention

À chaque modification du fichier `index.js`, il faudra **arrêter** le serveur (en exécutant Ctrl + C ou Cmd + C pour Mac OS X dans le terminal) et relancer le serveur à l'aide de la commande `node index.js`. Si vous envisagez de travailler régulièrement avec Node.js, vous pouvez installer une librairie appelée `nodemon`¹.

Remarque **Syntaxe « à retenir »**

Pour concevoir une application REST, il faut définir les ressources, les fonctionnalités et sous quel(s) format(s) les rendre disponibles à nos utilisateurs et utilisatrices.

Une fois ceci fait, des librairies spécialisées comme Express.js² permettent de développer **très rapidement** une API REST complète. De plus, utiliser un Framework populaire permet l'accès à de nombreux tutoriels (écrits, projets, vidéos) pour monter en compétences et trouver de l'aide.

¹ <https://www.npmjs.com/package/nodemon>

² <https://expressjs.com/fr/>

Si nous ne devons retenir que 2 choses sur Express.js, c'est qu'il fournit **un serveur** capable de rendre disponible notre application JavaScript en passant par une URL (et un port HTTP) et un **routeur** en charge d'**associer nos fonctions JavaScript à des méthodes HTTP**, comme recommandé dans l'architecture REST.

Si ce cours n'a pas pour objectif d'acquérir la maîtrise complète du Framework Express.js, il vous aura permis en quelques minutes d'obtenir une API REST complète.

Consultez la vidéo ci-dessous pour apprendre comment tester votre API et vérifier qu'elle fonctionne !

Complément

- Modèle de Richardson^(EN)¹
- Nodemon² (ne plus avoir besoin d'arrêter le serveur Node.js)

IV. Exercice : « La gestion des utilisateurs »

Question

[solution n°2 p.25]

En adaptant le code fourni précédemment, fournissez le script capable de gérer une liste d'utilisateurs.

Il va falloir créer une liste « *virtuelle d'utilisateurs* », puis assigner une ou plutôt des URLs aux actions suivantes :

- Voir un utilisateur (en fonction de son email)
- Voir la liste de tous les utilisateurs
- Créer un utilisateur
- Modifier un utilisateur existant
- Supprimer un utilisateur existant

Pour les livres actuellement chez un utilisateur, stockez cela sous forme de liste d'ISBN de sorte à pouvoir identifier le livre au besoin.

Indice :

Basez-vous sur l'exemple présenté dans le cours pour les livres et consultez cet article³ en Français si vous êtes en difficulté.

V. Mise en ligne de son API REST

Objectifs

- Découverte et installation d'Heroku Toolbet
- Mettre en ligne son API
- Vérifier son travail avec Postman

1 <https://martinfowler.com/articles/richardsonMaturityModel.html>

2 <https://www.npmjs.com/package/nodemon>

3 <https://practicalprogramming.fr/node-js-api/>

Contexte

La mise en ligne d'une application est une étape très importante et une compétence précieuse qu'il s'agit d'acquérir au plus tôt dans votre formation.

Il existe de nombreuses possibilités pour mettre en ligne une application « *serveur* ». Vous pourriez commander un espace mutualisé ou dédié chez un grand hébergeur comme OVH, acheter un Raspberry Pi et le rendre disponible sur le réseau en utilisant votre box internet.

Il existe aussi des plateformes Cloud qui fonctionnent par abonnement annuel et dont la spécialité est de faciliter au maximum le déploiement de vos applications.

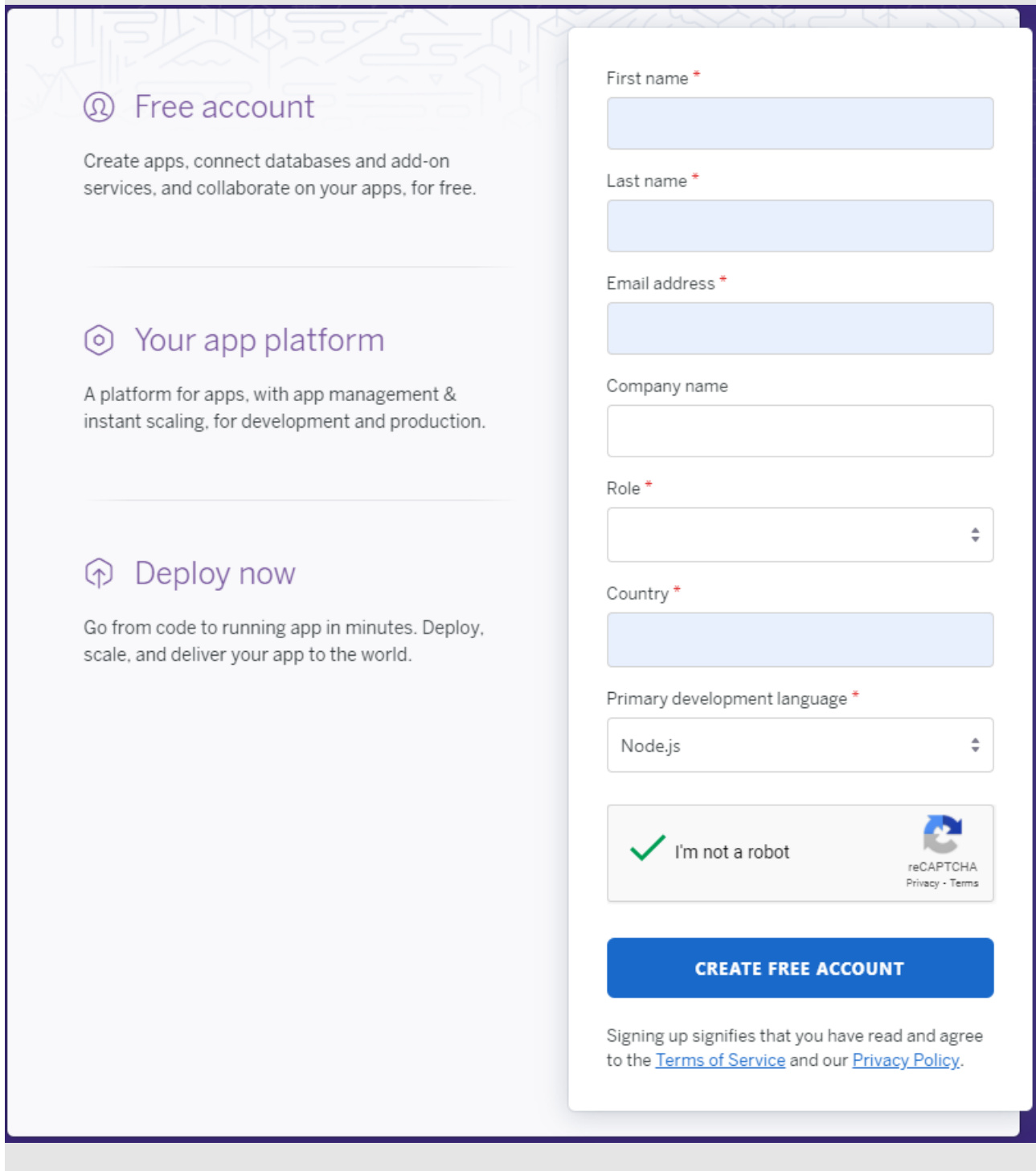
Et parmi ces plateformes, une est particulièrement appréciée par la communauté Node.js : il s'agit de la plateforme Heroku.

En configurant votre application et après avoir créé un compte sur cette plateforme, vous allez installer Heroku Toolbelt et mettre en ligne votre application !

Méthode

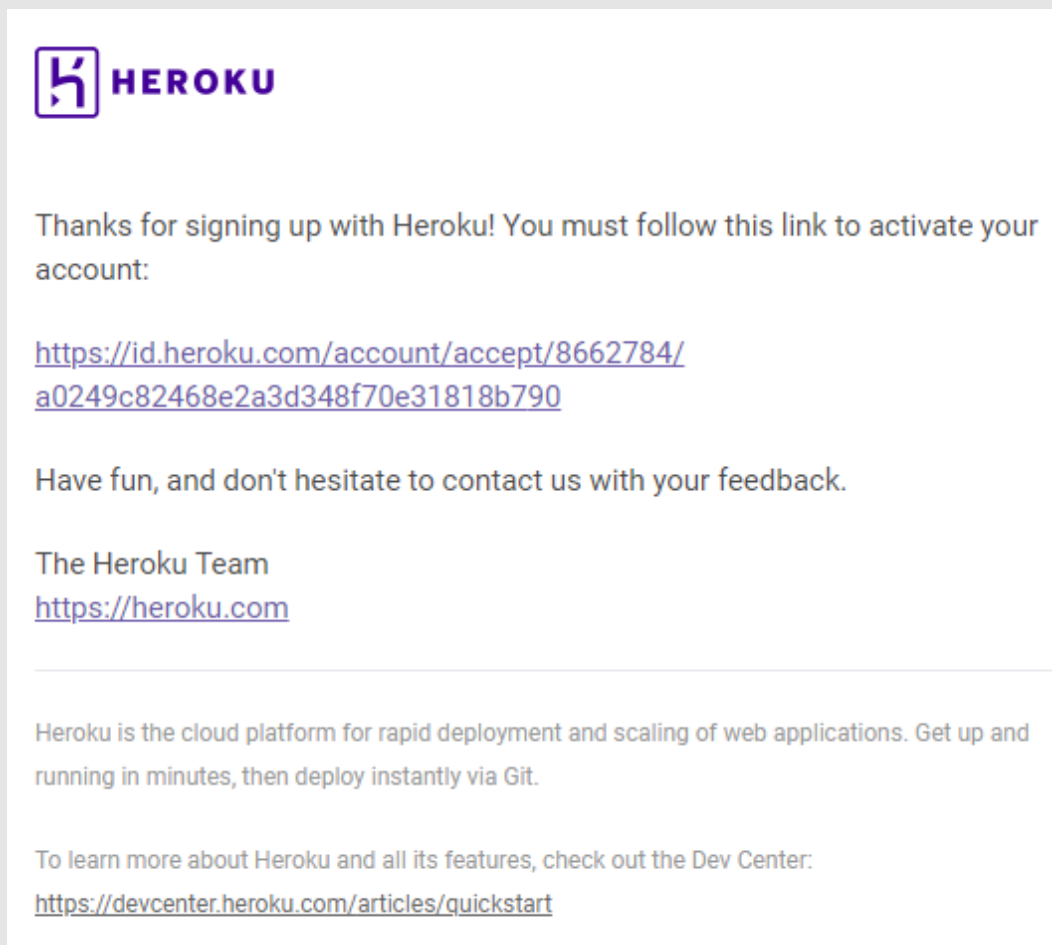
Création de comptes et installation d'Heroku CLI

Tout d'abord, créez un compte sur la plateforme sur Heroku en accédant à leur site : <https://www.heroku.com/>. Il faudra ensuite cliquer sur « *Sign Up* » et remplir le formulaire avec vos informations, comme présenté dans l'exemple ci-dessous :

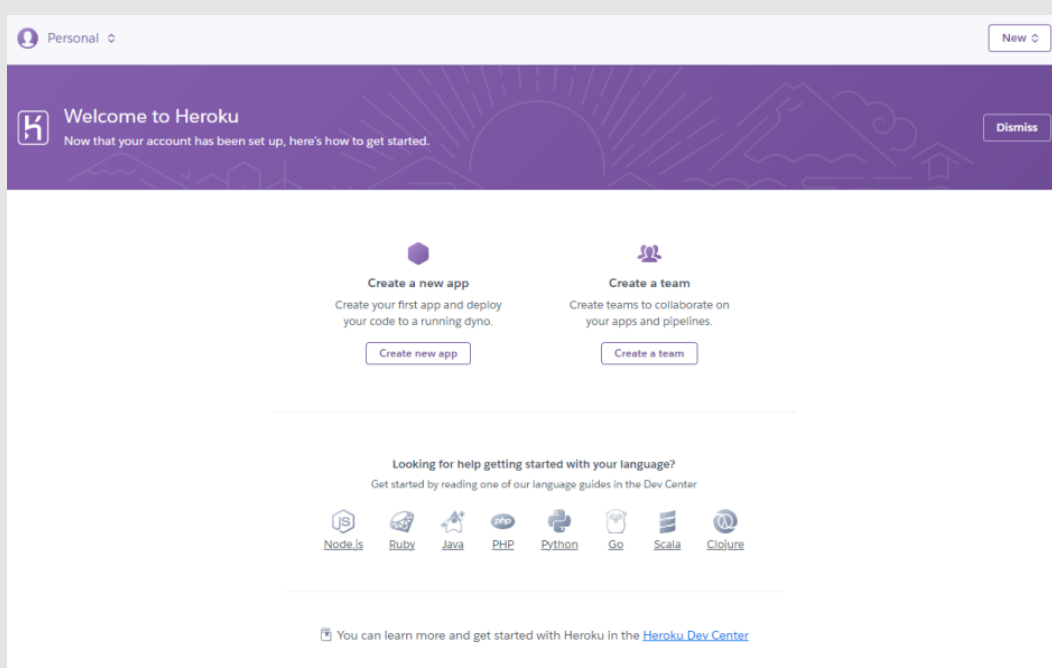


The screenshot shows the Heroku sign-up page. On the left, there are three sections: 'Free account' (with a person icon), 'Your app platform' (with a gear icon), and 'Deploy now' (with an upward arrow icon). Each section has a brief description. On the right, there is a registration form with the following fields: 'First name' (required), 'Last name' (required), 'Email address' (required), 'Company name', 'Role' (dropdown), 'Country' (required), and 'Primary development language' (dropdown, currently set to 'Node.js'). Below these fields is a reCAPTCHA 'I'm not a robot' checkbox. At the bottom of the form is a blue button labeled 'CREATE FREE ACCOUNT'. Below the button, there is a line of text stating: 'Signing up signifies that you have read and agree to the [Terms of Service](#) and our [Privacy Policy](#).'

Ensuite, vous allez recevoir un email de confirmation qu'il faudra valider en cliquant sur le lien comme dans l'exemple ci-dessous :



Une fois que vous avez cliqué sur ce lien, on vous demande de choisir un mot de passe sécurisé et une fois ceci fait, vous serez connecté à Heroku avec un plan « *Personal* » gratuit !



Configuration de notre application

Avant de pouvoir déployer l'application, nous allons devoir adapter notre code pour ajouter deux informations nécessaires au déploiement d'applications Node.js dans Heroku.¹

Tout d'abord, nous allons devoir configurer un « *script npm*² ». En effet, au déploiement de l'application Heroku va exécuter la commande `npm start` qui aura pour rôle d'appeler la commande que nous exécutons dans le terminal, pour rappel :

```
node index.js
```

Pour cela, nous allons éditer le fichier `package.json` et ajouter une ligne dans la section `"scripts"`. Le fichier final doit ressembler à l'exemple ci-dessous :

```
1 {
2   "name": "conception-apis",
3   "version": "1.0.0",
4   "description": "",
5   "main": "index.js",
6   "scripts": {
7     "start": "node index.js"
8   },
9   "author": "",
10  "license": "ISC",
11  "dependencies": {
12    "express": "^4.17.1"
13  }
14 }
```

Ensuite, vous avez choisi le port 3000 pour accéder à notre application dans le navigateur.

Pour rappel, c'est la ligne suivante de votre application qui configure ce comportement :

```
const port = 3000;
```

Seulement voilà, dans Heroku cette variable doit être égale à `process.env.PORT`.

Nous allons donc modifier le code de notre projet :

```
const port = process.env.PORT || 3000;
```

Si la variable `process.env.PORT` existe (comme dans Heroku), elle sera utilisée et sinon c'est la valeur 3000 qui le sera.

De cette façon, votre application continuera de fonctionner sur votre poste de travail !

Initialisation du dépôt git et de l'application Heroku

Heroku ne fonctionne qu'avec des applications versionnées avec le logiciel git. Il faut donc initialiser le projet à l'aide de la commande suivante **dans le dossier qui contient le fichier JavaScript sur lequel nous travaillons** :

```
git init
git add .
git commit -m "Projet versionné avec git"
```

Maintenant, nous pouvons enfin utiliser Heroku Toolbelt qui est une application utilisable dans l'invite de commandes.

À nouveau, dans le dossier qui contient notre projet, exécutez les instructions suivantes :

```
heroku login
heroku create
```

¹ <https://devcenter.heroku.com/articles/getting-started-with-nodejs#deploy-the-app>

² <https://docs.npmjs.com/cli/v6/using-npm/scripts>

La première commande va ouvrir un navigateur et vous demander de vous connecter à Heroku, tandis que la seconde va créer une application Heroku.

Une application Heroku, c'est un espace de stockage accessible via une URL spécifique.

Par exemple, voici ce que l'on peut obtenir :

```
C:\Users\micka\Projects\conception-apis>heroku create
Creating app... done, ⬢ stormy-escarpment-67384
https://stormy-escarpment-67384.herokuapp.com/ | https://git.heroku.com/stormy-escarpment-67384.git
```

La création de l'application Heroku crée une nouvelle destination de déploiement git, qui est automatiquement ajoutée au dépôt que nous avons créé.

Déploiement

Pour terminer, il faudra publier votre code sur la plateforme à l'aide de l'instruction suivante :

```
git push heroku master
```

Cette instruction git aura pour effet de publier le code de votre projet sur le serveur qu'Heroku a mis à votre disposition.

Vous devriez avoir dans l'invite de commande, un message de succès de déploiement et l'URL d'accès au serveur :

```
remote: ----> Launching...
remote:      Released v6
remote:      https://stormy-escarpment-67384.herokuapp.com/ deployed to Heroku
remote:
```

Félicitations, vous avez déployé avec succès votre première API REST dans le cloud !

Exemple

```
1 # index.js
2 const express = require('express')
3 const app = express()
4 const port = process.env.PORT || 3000;
5 app.use(express.json())
6
7 const books = [
8   {
9     "titre": "Livre 1",
10    "auteur": "Jonathan",
11    "isbn": "12345679",
12    "disponibilite": true,
13    "date_retour": null
14  },
15  {
16    "titre": "Livre 2",
17    "auteur": "Elise",
18    "isbn": "98765428",
19    "disponibilite": false,
20    "date_retour": "01/01/2028"
21  },
22  {
23    "titre": "Livre 3",
24    "auteur": "Zineb",
25    "isbn": "87357493",
26    "disponibilite": false,
27    "date_retour": null
28  }
29 ]
```

```

28   },
29 ]
30
31 app.get('/', (req, res) => {
32   res.json({ 'message': "Accueil de l'API" })
33 })
34
35 app.get('/books/', (req, res) => {
36   res.json(books)
37 })
38
39 app.get('/books/:isbn', (req, res) => {
40   const isbn = req.params.isbn
41   const book = books.find(book => book.isbn === isbn)
42
43   res.json(book)
44 })
45
46 app.post('/books', (req, res) => {
47   books.push(req.body)
48   res.status(200).json(books)
49 })
50
51 app.put('/books/:isbn', (req, res) => {
52   const isbn = req.params.isbn
53   let book = books.find(book => book.isbn === isbn)
54
55   // Modification du livre
56   book.titre = req.body.titre
57   book.auteur = req.body.auteur
58   book.disponibilite = req.body.disponibilite
59   book.date_retour = req.body.date_retour
60
61   res.status(200).json(book)
62 })
63
64 app.delete('/books/:isbn', (req, res) => {
65   const isbn = req.params.isbn
66   const book = books.find(book => book.isbn === isbn)
67   books.splice(books.indexOf(book), 1)
68
69   res.json(books)
70 })
71
72 app.listen(port, () => {
73   console.log('Serveur démarré')
74 })

```

```

1 # package.json
2 {
3   "name": "conception-apis",
4   "version": "1.0.0",
5   "description": "",
6   "main": "index.js",
7   "scripts": {
8     "start": "node index.js"
9   },
10  "author": "",

```

```

11  "license": "ISC",
12  "dependencies": {
13    "express": "^4.17.1"
14  }
15 }

```

Remarque Syntaxe « à retenir »

Pour pouvoir déployer son API REST sur le service Heroku, il va falloir adapter votre application.

Tout d'abord, nous devons rendre le port de l'application configurable avec la variable reconnue par Heroku `process.env.PORT`. Ensuite, déclarer un script dans le fichier `package.json` pour qu'Heroku sache quelle commande exécuter pour démarrer Node.js.

Enfin, Heroku - comme de nombreuses autres plateformes de déploiement - n'est compatible qu'avec le logiciel git. Il faudra donc versionner votre projet, ce qui est une bonne pratique de développement.

Complément

Déployer une application Node.js sur Heroku^(EN)¹

VI. Exercice : Appliquer la notion

Question

[solution n°3 p.26]

À votre tour d'effectuer le déploiement de votre API REST sur Heroku !

Indice :

Reprenez étape par étape les différentes instructions de cette partie du cours et mettez-les en application.

Une fois l'application déployée, connectez-vous sur votre compte Heroku et accédez aux détails de l'application que vous venez de déployer : combien d'addons ont été activés ?

VII. Essentiel

VIII. Auto-évaluation

A. Exercice

Dans le projet d'API que nous avons présenté, nous n'avons pas géré une fonctionnalité super importante : le retrait et le retour des livres par les utilisateurs.

Question

[solution n°4 p.26]

Dans cet exercice final, à vous d'implémenter ces deux fonctionnalités et de mettre en ligne cette application comme un(e) vrai(e) professionnel(le) du web !

Objectif :

Ajout de 2 fonctionnalités à l'API actuelle (qui comprend la gestion des utilisateurs vu précédemment) :

Sur l'URI `/emprunt/<email-utilisateur>/<isbn-livre>` (ex. : `/emprunt/john.doe@studi.fr/123456789`)

- Passer la disponibilité du livre concerné à `false` et ajouter une date de retour de 14 jours par rapport à la date de l'emprunt,
- Ajouter dans la propriété « *livres* » de l'utilisateur/trice l'ISBN du livre.

¹ <https://devcenter.heroku.com/articles/getting-started-with-nodejs#deploy-the-app>

Sur l'URI /retour/<email-utilisateur>/<isbn-livre> (ex. : /retour/john.doe@studi.fr/123456789)

- Passer la disponibilité du livre concerné à `true` et supprimer la date de retour,
- Supprimer de la propriété « *livres* » de l'utilisateur/trice l'ISBN du livre.

Bonus : gérer le cas où le livre que l'on souhaite emprunter n'est pas disponible en retournant un message d'erreur.

B. Test

Exercice 1 : Quiz

[solution n°5 p.27]

Question 1

Qu'est-ce que Node.js ?

- ☐ Un site web
- ☐ Une plateforme logicielle
- ☐ Un langage de programmation

Question 2

Qu'est-ce que npm ?

- ☐ Un gestionnaire de dépendances
- ☐ Un langage de programmation
- ☐ Une librairie qu'on installe dans notre projet node.js
- ☐ Un logiciel en ligne de commandes

Question 3

Pourquoi faut-il généralement installer la version LTS de Node.js ?

- ☐ Il est plutôt recommandé d'installer la version la plus récente possible
- ☐ Elle est plus stable
- ☐ Elle est maintenue plus longtemps
- ☐ Elle contient les dernières fonctionnalités

Question 4

Quel est le format de retour d'une API REST recommandé par ce cours ?

- ☐ XML
- ☐ JSON
- ☐ REST
- ☐ SOAP

Question 5

Quelle méthode d'Express.js doit-on utiliser quand l'objectif est de récupérer de l'information ?

- ☐ `delete()`
- ☐ `post()`
- ☐ `put()`
- ☐ `get()`

Question 6

Dans la conception d'une API REST, à quoi correspondent les ressources ?

- ☐ Les appels API (ex. : /livres ou /utilisateurs/<email>)
- ☐ Les objets métiers de l'application (ex. : des livres ou des utilisateurs)
- ☐ Les deux !

Question 7

(Rappels HTTP) : quelle méthode HTTP est recommandée pour la suppression d'une ressource ?

- ☐ GET
- ☐ POST
- ☐ PUT
- ☐ DELETE

Question 8

Quel outil est recommandé dans ce cours pour tester son API REST ?

- ☐ npm
- ☐ Postman
- ☐ Heroku

Question 9

Dans Heroku CLI (ou Toolbelt), quelle instruction permet de s'identifier ?

- ☐ `heroku connect`
- ☐ `heroku login`
- ☐ `heroku signin`

Question 10

Dans Heroku CLI (ou Toolbelt), quelle instruction permet de créer une nouvelle application ?

- ☐ `heroku new:app`
- ☐ `heroku init -y`
- ☐ `heroku create`

Solutions des exercices

p. 9 Solution n°1

Comme le laissait deviner l'indice et même si ce n'était pas obligatoire de le faire avec cette librairie, vous pouvez installer et utiliser marked¹.

Dans l'invite de commandes :

```
npm i marked
```

Dans le fichier index.js :

```
1 const {marked} = require("marked");
2 const html = marked("# Marked in Node.js\n\nRendered by **marked**.");
3 console.log(html)
```

Voici le résultat attendu dans votre invite de commandes :

```
micka@DESKTOP-D0QDBAR MINGW64 ~/Projects/conception-apis
$ node index.js
<h1 id="marked-in-nodejs">Marked in Node.js</h1>
<p>Rendered by <strong>marked</strong>.</p>
```

p. 14 Solution n°2

Comme l'indice l'indiquait, il suffisait d'adapter un peu le code d'exemple précédent en prenant en compte la structure de l'utilisateur décrite dans le cours :

```
1 const express = require('express')
2 const app = express()
3 const port = 3000
4 app.use(express.json())
5
6 // Une liste d'exemple de livres, qui pourrait être récupérée en base de données
7 const utilisateurs = [
8   {
9     "nom": "Doe",
10    "prenom": "John",
11    "email": "john.doe@studi.fr",
12    "livres": [ ],
13  },
14  {
15    "nom": "Doe",
16    "prenom": "Jane",
17    "email": "jane.doe@studi.fr",
18    "livres": [ ],
19  },
20  {
21    "nom": "Mickaël",
22    "prenom": "Andrieu",
23    "email": "mickael.andrieu@exemple.fr",
24    "livres": ["12345679", "87357493" ],
25  },
26 ]
```

¹ <https://marked.js.org/>

```

27
28 app.get('/users/', (req, res) => {
29   res.json(utilisateurs)
30 })
31
32 app.get('/users/:email', (req, res) => {
33   const email = req.params.email
34   const utilisateur = utilisateurs.find(utilisateur => utilisateur.email === email)
35
36   res.json(utilisateur)
37 })
38
39 app.post('/users', (req, res) => {
40   utilisateurs.push(req.body)
41   res.status(200).json(utilisateurs)
42 })
43
44 app.put('/users/:email', (req, res) => {
45   const email = req.params.email
46   const utilisateur = utilisateurs.find(utilisateur => utilisateur.email === email)
47
48   // Modification de l'utilisateur
49   utilisateur.nom = req.body.nom
50   utilisateur.prenom = req.body.prenom
51   utilisateur.email = req.body.email
52   utilisateur.livres = req.body.livres
53
54   res.status(200).json(utilisateur)
55 })
56
57 app.delete('/users/:email', (req, res) => {
58   const email = req.params.email
59   const utilisateur = utilisateurs.find(utilisateur => utilisateur.email === email)
60   utilisateurs.splice(utilisateurs.indexOf(utilisateur), 1)
61
62   res.json(utilisateurs)
63 })
64 app.listen(port, () => {
65   console.log('Serveur démarré')
66 })

```

p. 21 Solution n°3

Cette application ne nécessitait aucun add-on et aucun n'a été activé !

Mais si un jour vous souhaitez utiliser une base de données, il faudra activer l'add-on correspondant et le configurer¹.

p. 21 Solution n°4

¹ <https://elements.heroku.com/addons>

Une solution complète est disponible sur GitHub à cette adresse¹. Intéressons-nous ici aux deux actions supplémentaires qu'il fallait développer !

```
1 /**
2  * Book Store management
3  */
4
5 app.post('/borrow/:mail/:isbn', (req, res) => {
6   const mail = req.params.mail
7   const isbn = req.params.isbn
8
9   const user = users.find(user => user.mail === mail)
10  const book = books.find(book => book.isbn === isbn)
11
12  user.books.push(book.isbn)
13  book.availability = false
14  book.return_date = moment().add(14, 'days');
15
16  return res.json(book)
17 })
18
19 app.post('/bring-back/:mail/:isbn', (req, res) => {
20   const mail = req.params.mail
21   const isbn = req.params.isbn
22
23   const user = users.find(user => user.mail === mail)
24   const book = books.find(book => book.isbn === isbn)
25
26   user.books.splice(user.books.indexOf(book), 1)
27   book.availability = true
28   book.return_date = null;
29
30   return res.json(user)
31 })
```

Exercice p. 22 Solution n°5

Question 1


Qu'est-ce que Node.js ?

- ☐ Un site web
- ☒ Une plateforme logicielle
- ☐ Un langage de programmation
- ☐ Node.js est une plateforme logicielle open source permettant l'exécution de JavaScript en dehors du navigateur web.

Question 2


Qu'est-ce que npm ?

¹ <https://github.com/mickaelandrieu/bookstore-node-rest-api>

- ☒ Un gestionnaire de dépendances
 - ☐ Un langage de programmation
 - ☐ Une librairie qu'on installe dans notre projet node.js
 - ☒ Un logiciel en ligne de commandes
-  npm est le gestionnaire de dépendances le plus populaire de l'écosystème Node.js.


Question 3

Pourquoi faut-il généralement installer la version LTS de Node.js ?

- ☐ Il est plutôt recommandé d'installer la version la plus récente possible
 - ☒ Elle est plus stable
 - ☒ Elle est maintenue plus longtemps
 - ☐ Elle contient les dernières fonctionnalités
-  La version LTS (« *Long Time Survey* ») est recommandée pour la plupart des usages, car c'est une version stable et qui est maintenue plus longtemps.


Question 4

Quel est le format de retour d'une API REST recommandé par ce cours ?

- ☐ XML
 - ☒ JSON
 - ☐ REST
 - ☐ SOAP
-  L'essentiel des API REST utilise le format JSON (plutôt) que du XML, car c'est un format très facile à manipuler.


Question 5

Quelle méthode d'Express.js doit-on utiliser quand l'objectif est de récupérer de l'information ?

- ☐ `delete()`
 - ☐ `post()`
 - ☐ `put()`
 - ☒ `get()`
-  Selon l'architecture REST, la récupération de ressources se fait à l'aide de la méthode GET, il faut donc utiliser la méthode `get()` du Framework.


Question 6

Dans la conception d'une API REST, à quoi correspondent les ressources ?

- ☐ Les appels API (ex. : /livres ou /utilisateurs/<email>)
- ☒ Les objets métiers de l'application (ex. : des livres ou des utilisateurs)
- ☐ Les deux !
-  Selon l'architecture REST, les ressources sont les différents objets métiers que l'API doit manipuler.


Question 7

(Rappels HTTP) : quelle méthode HTTP est recommandée pour la suppression d'une ressource ?

- ☐ GET
- ☐ POST
- ☐ PUT
- ☒ DELETE
-  Selon l'architecture REST, l'utilisation d'une requête de type `DELETE` est recommandée pour demander au serveur la suppression d'une ressource.


Question 8

Quel outil est recommandé dans ce cours pour tester son API REST ?

- ☐ npm
- ☒ Postman
- ☐ Heroku
-  Le logiciel Postman est un client HTTP, c'est l'outil le plus adapté pour tester votre API REST.


Question 9

Dans Heroku CLI (ou Toolbelt), quelle instruction permet de s'identifier ?

- ☐ `heroku connect`
- ☒ `heroku login`
- ☐ `heroku signin`
-  L'instruction `heroku login` permet de s'authentifier en ligne de commandes et de rattacher notre invite de commandes au compte en ligne sur lequel il faut s'inscrire.

Question 10

Dans Heroku CLI (ou Toolbelt), quelle instruction permet de créer une nouvelle application ?

- ☐ `heroku new:app`
- ☐ `heroku init -y`
- ☒ `heroku create`
-  L'instruction `heroku create` va créer une nouvelle application dans Heroku, un peu comme si la plateforme vous avait donné accès à un serveur et un nom de domaine.