Node.js

Développer des applications en Javascript

Prérequis

- *Node.js*
- *Git*
- *IDE : Visual Studio Code, Eclipse, IntelliJ, etc…*

Qu’est-ce que Node.js

Présentation et historique

La plateforme Node.js

« Node.js est une plateforme logicielle libre et événementielle en JavaScript orientée vers les applications réseau qui doivent pouvoir monter en charge. »

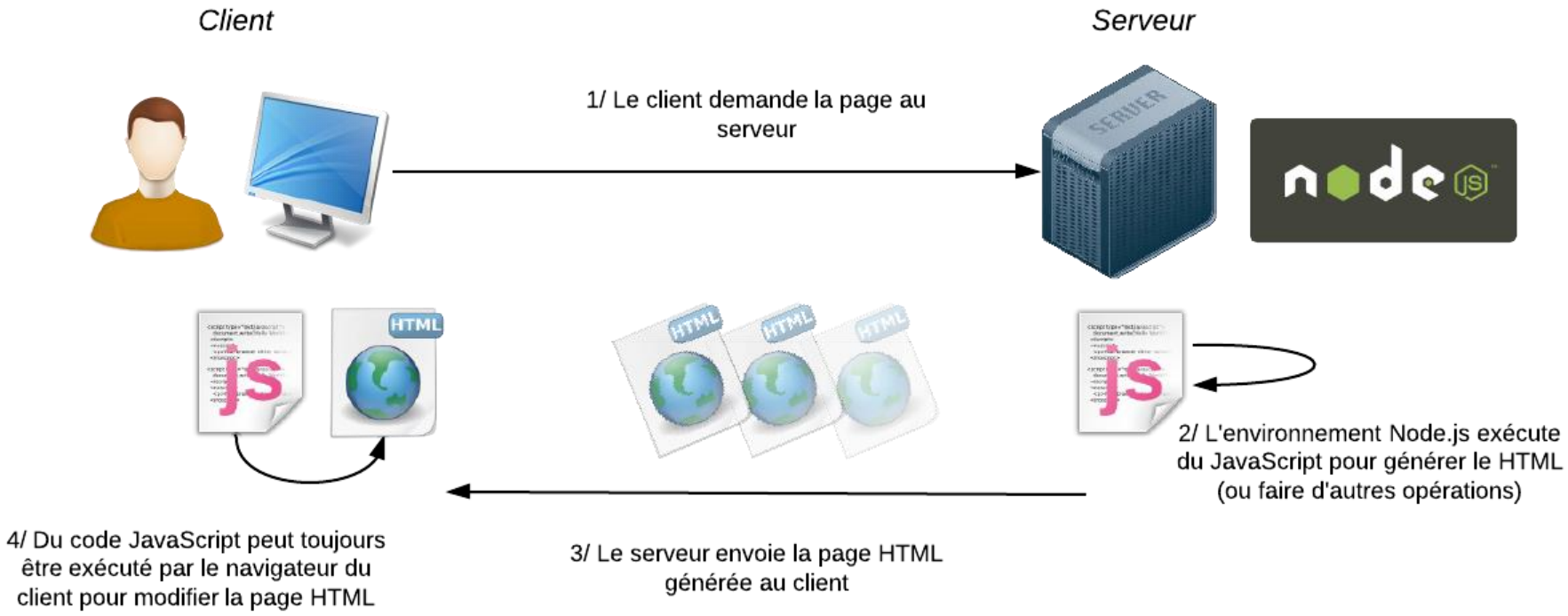
-- Wikipedia

- Programmer en javascript en dehors du navigateur web
- Multi-plateforme
- Exécutable, frameworks, gestionnaire de paquets, etc…

Le logiciel Node.js

- Se base sur le moteur javascript de Google V8 (libre et opensource)
- Ecrit en C++
- Multi-plateforme
- Créé par Ryan Dahl en 2009
- Applications en ligne de commande
- Applications serveurs

Le logiciel Node.js



Source : https://openclassrooms.com/courses/des-applications-ultra-rapides-avec-node-js/node-js-mais-a-quoi-ca-sert

Installation

Installer et configurer le logiciel

Installation

- Node.js (« node ») est un outil en ligne de commande. Il ne dispose pas d'interface graphique
- Multi-plateforme :
  - Linux
  - Windows
  - Mac OS
  - Docker
  - …
- ATTENTION : différentes versions et types de versions : stable, LTS, etc …

Installer sous Windows

- Télécharger le binaire https://nodejs.org/en/download/
- Installer avec les options par défaut
- Vérifier l'installation : node -v

Exemple : « Hello World »

- Créer un fichier vide nommé hello.js
- Insérer le texte `console.log('Hello ' + (process.argv[2] || ''));`
- Ouvrir une fenêtre console
- Exécuter le script : `node hello.js Dominik`

- La variable ‘process’ est fournie par Node.js et contient des informations sur le process actuel et son environnement, notamment les arguments transmis via la ligne de commande

Exercice 1

- Ecrire une application node qui retourne le nombre total de caractères des arguments passés via la ligne de commande

- Exemple : ***node app.js hello bonjour hola*** => affiche : **16**

- Astuce 1 : process.argv => tableau permettant de récupérer les arguments passés via la ligne de commande
- Astuce 2 : se servir des méthodes map et reduce pour calculer la somme
  - https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/map
  - https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Array/reduce

Exercice 2

- Ecrire une application node qui additionne les nombres passés comme arguments

- Exemple : ***node app.js 45 35 12*** => affiche : **92**

- Astuce 1 : process.argv => tableau permettant de récupérer les arguments passés via la ligne de commande
- Astuce 2 : se servir de parseInt pour transformer les arguments en nombre entier
- Astuce 3 : se servir de reduce pour calculer la somme

NPM

Utiliser des packages et modules via le package manager

NPM

- Node Package Manager
- Gestion des packages node
- Plateforme publique : https://www.npmjs.com/
- Outil ligne de commande
- Installé par défaut avec Node.js
- Une liste de packages intéressants : https://node.cool/

Qu’est-ce qu’un package ?

- Un ensemble de ressources (fichiers javascripts et autres (css, json, …)
- Destiné à être réutilisé (= librairie)
- Gestion des dépendances : un package peut « dépendre » d’un autre et utiliser son contenu
- En général : des packages open-source et libre

Comment utiliser npm ?

- Installation/utilisation locale : npm install nom-du-package
- Installation/utilisation globale : npm install --global nom-du-package
- Ajouter une dépendance : npm install --save nom-du-package
- Aide et liste de toutes les commandes : npm -l

npm install

- Télécharger un paquet : npm install nom-du-paquet
- Installer tous les paquet demandés dans package.json : npm install
- « Installation » local dans le dossier de travail dans un dossier « node_modules »
- Si le paquet demandé dépend d'autres paquets, alors ces paquets sont également télécharger automatiquement
- L'option '--save' permet d'enregistrer le nom du paquet comme dépendance dans le fichier package.json du dossier courant (pas besoin de spécifier cette option dans des versions récentes de npm)
- L'option '--global' (ou -g) permet d'installer un paquet, pour l'utiliser comme un outil de ligne de commande par la suite

Autres commandes npm

- Init : Permet d'initialiser un fichier package.json dans le dossier courant
- start, stop, test : commandes configurable dans package.json
- login, config : utile pour publier soi-même des packages sur npmjs.com

Exemple : Utiliser un package avec npm

- npm init
- npm install --save load-json-file
- Créer un fichier `app.js` avec le contenu suivant :

```
const loadJsonFile = require('load-json-file');
loadJsonFile('foo.json').then(json => {
  console.log(json);
  console.log('foo is ' + json.foo);
});
```

- Créer un fichier `foo.json` avec le contenu `{"foo": true}`
- `node app.js`

Modules

Modules

- Un module = un fichier javascript
- Chaque module expose un ou plusieurs composants via la variable module.exports (fournie par Node.js)
- Un module peut utiliser un autre module via la méthode require(…) fournie par Node.js
- Node contient beaucoup de modules par défaut : https://nodejs.org/api/
- Un package npm contient un ou plusieurs modules
- Pour aller (beaucoup) plus loin :

https://medium.freecodecamp.org/requiring-modules-in-node-js-everything-you-need-to-know-e7fbd119be8

Modules : exemple simple

- Créer un fichier monModule.js avec le contenu suivant :

```
function hello(name) {
   return 'Hello ' + name;
}
function bonjour(name) {
   return ‘Bonjour ' + name;
}
module.exports.hello = hello;
module.exports.bonjour = bonjour;
```

- Créer un fichier app.js avec le contenu suivant :

```
let monModule = require(‘./monModule’);
let h = monModule.hello;
let name = process.argv[2];
let message = h(name);
console.log(message);

console.log(monModule.bonjour(name));
```

- Lancer : node app.js

Modules : exemple simple

- Créer un fichier hello.js avec le contenu suivant :

```
function hello(name) {
   return 'Hello ' + name;
}
module.exports = hello;
```

Créer un fichier app.js avec le contenu suivant :

```
let hello = require('./hello’);
let message = hello(name);
console.log(message);

console.log(helloModule.bonjour(name);
```

- Lancer : node app.js

Exercice : Modules

- Créer un programme qui permet d'exécuter les 2 opérations des exercices prédédents : compter les lettres et faire la somme. Utiliser des modules pour bien séparer le code des 2 opérations.

node app.js count hello bonjour => résultat : 12

node app.js sum 12 4 7 => résultat : 23

Modules intégrés

Node contient quelques modules par défaut, que l'on peut utiliser sans avoir installé de package supplémentaire.

Par exemple

- Filesystem 'fs'
- Path 'path'

Exercice : Modules node

Ecrire une application Node.js qui concatène 2 fichiers donnés en arguments.

node app.js monFichier1.txt MonFichier2.txt

=> résultat attendu : un fichier ‘concat.txt’ qui contient la concaténation des 2 fichiers.

Modules d'un package

Un package contient 1 ou plusieurs modules.

let monModule = require('nom-du-package')

⇒Fait référence au fichier index.js dans le package

Let monModule = require('nom-du-package/nomModule')

=> Fait référence au fichier nomModule dans le package

Exercice : Modules d'un package

Ecrire une application qui permet de convertir du markdown en html

Exemple :

node app.js monFichierMarkdown.md monFichierHtml.html

=> Convertir le fichier « monFichierMarkdown.md » en html et écrit le résultat dans « monFichierHtml.html »

Astuces :

- Chercher un package existant (librairie) qui permet de convertir du markdown en html (rappel : https://www.npmjs.com/)
- Créer un fichier app.js
- Utiliser la librairie identifiée sur npm pour la conversion (require … etc) en se basant sur l'aide sur la page web du module

Express

Créer des applications web avec Node.js et Express

Express

- Un framework web pour node.js
- Un package npm pour créer des applications de type serveur web
- http://expressjs.com/

Exemple : Hello Web

- Installer la dépendance express npm install --save express
- Utiliser le module et créer une application
- Ajouter un handler pour renvoyer une réponse
- Ecouter sur un port

```
const express = require('express');
const app = express();
app.get('/', function(request, response) {
  response.send('Hello World');
});
app.listen(80);
```

Programmation asynchrone

- Approche traditionnelle node : callbacks

```
function action(param, callback) {
  var result = doSomething(param);
  callback(result);
}
```

- Autre approche : promise

```
var promise = action(param);
promise.then(successHandler, errorHandler);
```

- Les 2 approches peuvent être utilisé dans node, beaucoup d'APIs utilisent des callbacks, il peut être plus simple et lisible d'utiliser des promises.
- Syntaxe async / await : utilisable avec des promises

Routing

- Le routage de base se fait selon le pattern suivant :

app.METHOD(PATH, HANDLER)

- METHOD possibles : GET, POST, PUT, DELETE etc (verbes http)
- PATH : le chemin d'URL (en chaîne de caractère)
- HANDLER : une fonction javascript, qui prend les paramètres request et response:

```
function (request, response) {

});
```

Request et Response

- L'objet request contient toute les données de la requête : entêtes (headers), corps (body), paramètres, url, query etc.
- L'objet response contient tout ce qui va être transmis au client : headers, body …
- La méthode response.send(…) ne peut être appeler qu'une seule fois.
- La méthode response.json(…) permet de renvoyer du contenu json au client

Fichiers statiques

- Express met à disposition un moyen simple pour servir des fichiers statiques

```
app.use(express.static('public'))
```

- Cet exemple sert tous les fichiers contenus dans le sous-dossier ‘public’ du repertoire courant

```
app.use('/static', express.static(path.join(__dirname, 'public')))
```

- Cet exemple sert tous les fichiers contenus dans le sous-dossier ‘public’ du répertoire dont lequel se trouve le script exécute par node

Middle-wares

- Dans express on appelle middle-ware toutes les fonctions qui traitent les requêtes entrantes.
- Il existent beaucoup de packages/modules de type middle-ware sur npm
- Les middle-wares peuvent :
  - Exécuter tout type de code.
  - Apporter des modifications aux objets de requête et de réponse.
  - Terminer le cycle de requête-réponse.
  - Appeler la fonction middleware suivant dans la pile. (next)

Middle-wares

- Exemple 1 :

```
var logRequests = function(req, res) {
  console.log(req.url);
}
app.use(logRequests);
```

- Exemple 2 :

```
app.use(express.static('nomDuDossier'));
```

- Pour aller (beaucoup) plus loin :

http://expressjs.com/fr/guide/using-middleware.html

Middle-wares : packages utiles

- body-parser : par défaut, le body d'une requête n'est pas chargé dans express, la propriété req.body est vide. Il existe un middle-ware pour cela :

```
app.use(require('body-parser'));
```

- CORS : permet d'authoriser des requêtes cross-origines

```
app.use(require('cors')());
```

TP : Créer un backend REST simple

- Créer un backend REST pour lister, ajouter et supprimer des superhéros, format d'échange JSON
- Exemple « hero » : {id: 12345, name: 'Superman'}
- Requêtes :
  - GET /api/heroes => lister les héros
  - GET /api/heroes/:id => récupérer un héro via son id
  - POST /api/heroes => ajouter un héro
  - DELETE /api/heroes/:id => supprimer un héro de la liste
  - Bonus : GET /api/heroes/?name=Toto => liste les héros dont le nom contient Toto

TP : Créer un backend REST simple (suite)

- Astuce : exemple d'un objet hero : {id: 43, name: "Spiderman"}
- Astuce : utiliser Postman https://www.getpostman.com/ pour tester les requêtes (ou https://insomnia.rest)
- Astuce : stocker le tableau en mémoire (on n'utilise pas de base de données)

Express generator

Générer une application express

Express generator

- Permet de générer un projet express de base
- Rajoute d'autres dépendances
  - Body-parser
  - Cookie-parser
  - Jade / pug (moteur de templating html)
- Initialise package.json
- Crée une arborescence de dossier
  - public : fichiers statiques tel que css, js client, images
  - routes : les définitions de routes (code serveur)
  - views : les vues (par défaut fichiers Jade / pug)
  - app.js : l'application Node

Générer une application

- npm : npm install --global express-generator
- Pour créer un nouveau projet :
  express
- Installer les dépendances :
  npm install
- Pour démarrer le serveur :
  npm start
- Ouvrir le navigateur : localhost:3000

Les sous-routes

- Possibilité de créer un ensemble de routes dans un fichier et de l'importer dans un autre => permet de ne pas avoir toutes les routes dans un fichier.
- Fichier route : ex : userRoutes.js

```
var express = require('express');
var router = express.Router();
router.get('/subroute', function(req, res, next) {
  res.send('respond with a resource');
});
module.exports = router;
```

- Fichier principal : ex : users.js

```
const users = require('./userRoutes');
app.use('/users', users);
```

- La route '/users/subroute' est accessible, le handler est défini dans le fichier users.js

TP : express-generator

- Initialiser une nouvelle application avec le generateur
- Ajouter une page ‘about’
- Ajouter un menu qui s’affiche sur toutes les pages avec des liens vers ‘home’ et ‘about’
- Astuce : pour en savoir plus sur ‘jade’ / ‘pug’ : https://pugjs.org
  - https://pugjs.org/language/tags.html
  - https://pugjs.org/language/attributes.html

Persister

Persister des données avec MongoDB et Mongoose

MongoDB

- https://www.mongodb.com/fr
- Une base NoSQL
- Enregistre des « documents » dans des « collections » (plutôt que des « lignes » dans des « tables »
- Travaille avec un format semblable au json en interne
- Chaque « document » a un « id », qui est souvent de type « ObjectId » (~ une chaine de caractère assez longue).
- Il existe un client MongoDB pour node : package : ‘mongodb’

MongoDB

- Installer le package mongodb : `npm install --save mongodb`
- Se connecter à la base de données :

```
var mongoClient = require('mongodb').MongoClient;

mongoClient.connect('mongodb://localhost:27017', function(error, db) {
  if (error) {
    console.log(error);
  } else {
    var dbAnimals = db.db('animals');
}});
```

MongoDB

- Une fois connecté on peut utiliser l'objet db :

```
dbAnimals.collection('animals').find().toArray(function(error, results) {
  console.log(results);
});
```

- Pour insérer un document :

```
dbAnimals.collection('animals').save({name: 'Nemo', type: 'fish'}, null, function
(error, results) {
    if (error) throw error;
    console.log(results.ops[0]);
});
```

- Pour trouver un document via son id :

```
const ObjectID = require('mongodb').ObjectID;
dbAnimals.collection('animals').findOne({ _id: new ObjectID(req.params.id) },
function(error, result) {console.log( result);  });
```

Mongoose JS

- Simplifier le travail avec MongoDB via Object Modeling en utilisant le package npm **mongoose**
- Code pour la connexion

```
const mongoose = require('mongoose’);
mongoose.connect('mongodb://localhost/test’);

mongoose.connection.once(‘open’, () => {
        // mettre ici le code à exécuter après connexion à la base de donnée.
});
```

- Modélisation des données : on définit d’abord les champs dont on a besoin sur les données avant de requêter ou insérer les données

```
const Chat = mongoose.model('Chat', { name: String });
```

Nom de la collection en base

Schéma de la collection (pas besoin de spécifier la propriété id)

Mongoose JS

- Exemple de code pour l'enregistrement : on crée un nouvel objet à partir du modèle.

```
const Chat = mongoose.model('Chat', { name: String });
const chat = new Chat({ name: 'Zildjian' });
chat.save().then((chat) => console.log(chat));
```

- Exemple de code pour la recherche d'une liste ou d'une seul entrée : on utilise les méthodes 'find' du modèle

```
Chat.find({name: 'Zildjian'}).then((chats) => console.log(chats), (error) =>
console.log(error));

Chat.findById(id).then((chat) => console.log(chat), (error) => console.log(error));
```

TP : Backend REST MongoDB / Mongoose

- Installer MongoDB : https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/
- Copier le TP Backend REST dans un nouveau répertoire
- Remplacer l'utilisation du tableau par une base MongoDB
- Utiliser Mongoose pour enregistrer et rechercher les heros dans la base MongoDB.

Authentification

Gérer l’authentification via JWT

Authentification

- Plusieurs méthodes possibles pour sécuriser des applications web
- Authentification http simple (Basic Auth, « .htaccess »)
- Authentification via token statique
- JWT (token dynamique)
- OAuth ou OAuth2

Authentification via un simple token

- Le but : interdire l'accès si le client n'envoie pas un token / « mot de passe »
- Côté client : on transmet à chaque requête un header « Authorization » contenant un token (~ mot de passe) prédéfini.
- Côté serveur : on vérifie à chaque requête que le header est bien présent, et qu'il contient le token attendu.

TP : protéger un backend express via token

- Reprendre le TP Backend REST
- Ajouter un « middle-ware » qui test la présence du token attendu (une valeur fixe)
- Si le token est présent on laisse passer la requête, s'il n'est pas présent, on renvoie un status 401 et un message d'erreur.
- Astuce : se servir de Insomnia pour tester les requêtes (ajouter le header « Authorization »)

Json Web Token (JWT)

« JSON Web Token (JWT) est un standard ouvert (RFC 7519) pour échanger de l'information de manière sécurisée via un jeton signé. »

-- Wikipedia

- Token d'authentification transmis dans l'entête de la requête HTTP
- Le token jwt contient des données chiffrées (à l'aide d'une clé connu uniquement côté serveur) tel que : l'id de l'utilisateurs, la durée de validité du token et d'autres données éventuels.
- Package npm pour la gestion jwt avec express : jsonwebtoken

TP : protéger un backend express avec jwt

- Utiliser le package ‘jsonwebtoken’
- Ajouter les routes suivantes :
  - POST /auth/register : créer un utilisateur (en postant un objet suivant :

  {"nom": "…", "email": "...", password: ""}
  - POST /auth/login : récupérer un jeton JWT (token) en postant un objet suivant : {"email": "...", password: ""}
  - GET /api/test : une route de test, interdire l’accès si pas de token valide transmis
  - Si utilisateur non autoriser, retourner un code erreur 401
- Astuce : le tutoriel suivant peut servir de base :

https://www.codementor.io/olatundegaruba/5-steps-to-authenticating-node-js-with-jwt-7ahb5dmyr

- Astuce : se servir de Postman pour tester les requêtes

Socket.IO

Créer une application avec des websockets

Socket.IO

- Messages entre serveur et client
- Se base sur la technologie websocket : https://developer.mozilla.org/fr/docs/WebSockets
- Communication bi-directionnelle
- Paquet npm socket.io
- https://socket.io/

TP Simple Socket.IO

- Utiliser le package socket.io
- Au chargement de la page demander la saisie du nom via prompt(…)
- Notifier les utilisateurs déjà connectés via alert(…) d'une connexion d'un nouveau socket
- Notifier les utilisateurs déjà connectés via alert(…) de la déconnexion d'un socket connecté
- Ajouter un bouton permettant de broadcaster un message via alert(…)

Training

Exercices divers

TP Blog API

- Créer une API REST pour un gestionnaire de blog en utilisant Node, Express, Mongoose et MongoDB
- Créer des ressources CRUD (create, read, update, delete) pour les collections suivantes :
  - Article (propriétés : titre, contenu, dateCreation, auteur (id et nom), tags)
  - Auteur (propriétés : nom, prénom, pseudo, adresse)
  - Utilisateur (propriétés : nom, prénom, login, email, password)
- Créer une ressource /api/auteurs/:id/article qui renvoie les articles d'un auteur
- Protéger l'API par simple token (version simple)
- Bonus : protéger l'API par JWT (ajout des ressources de /login et /register)

- (voir https://codebrains.io/build-simple-blog-api-express-es6-mongodb/)

TP JWT avec passport

- Protéger un backend avec passport et jwt (passport est une librairie d'authentification/autorisation)
- Suivre le tutoriel suivant : https://codebrains.io/add-jwt-authentication-to-an-express-api-with-passport-and-es6/

TP Chat Angular

- Créer un backend node et utiliser le package socket.io
- Voir tutoriel :

http://www.syntaxsuccess.com/viewarticle/socket.io-with-rxjs-in-angular-2.0

TP API Blog avec Loopback

- Loopback est un framework Node.js créé par IBM, permet de créer des APIs très facilement
- Créer l'API Blog avec Loopback

https://loopback.io/doc/en/lb3/Create-a-simple-API.html