Javascript avancé

Node.js

Javascript avancé

Node.js

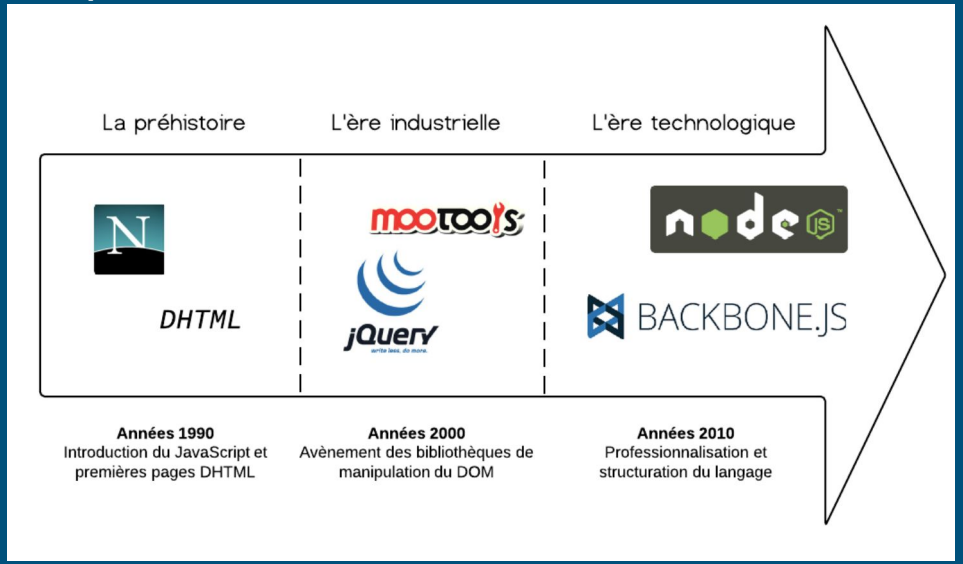
- Introduction
- Installation
- Les bases
- API REST
- Persistance de données
- Authentification
- WebSocket
- Tests avec Mocha

ENVOOO

Introduction

Le **Javascript** est un langage créé en **1995**

On peut considérer qu'il a eu 3 vies



Introduction

Node.js est un projet multi-plateforme open-source

- Permet d'exécuter du javascript côté serveur
- Comme le langage Javascript, Node.js est basé sur les évènements

Introduction

Node.js est rapide pour deux raisons

- son moteur d’exécution V8 (le même que Chrome)
- son modèle non bloquant

Introduction

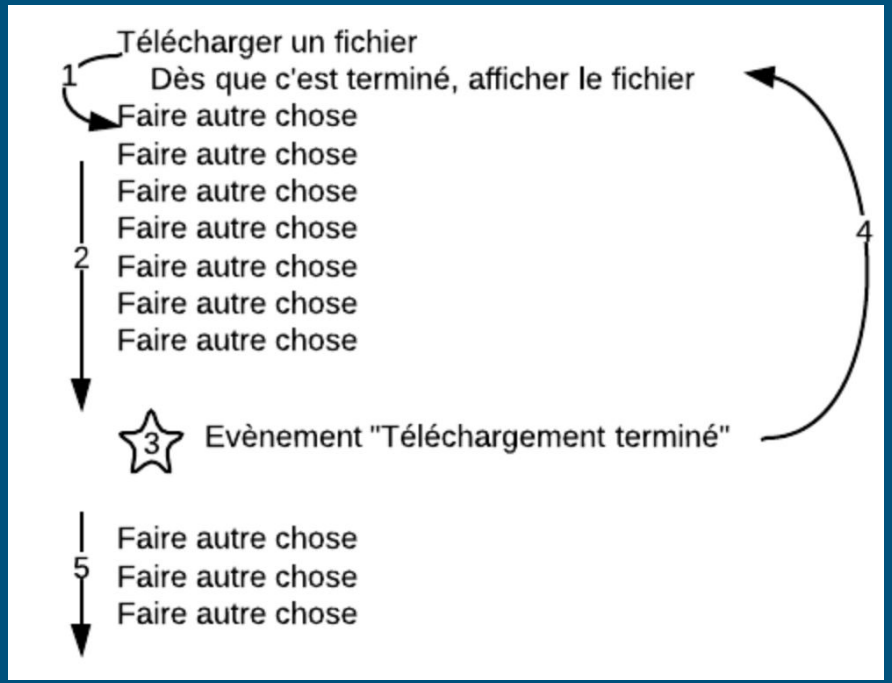
Modèle bloquant

```
1 Télécharger un fichier
2 Afficher le fichier
3 Faire autre chose
```

Modèle non bloquant

```
1 Télécharger un fichier
2     Dès que c'est terminé, afficher le fichier
3 Faire autre chose
```

Introduction



Installation

- Directement sur le site de node.js:

https://nodejs.org/

- Ou nvm pour gérer différentes versions de node

https://github.com/creationix/nvm

Les bases

L’objet global met à disposition un certain nombre de méthodes et de constantes

Comme son nom l’indique, cet objet est accessible partout

On peut y ajouter du contenu

Exemples:

__dirname

process.argv

process.exit()

Les bases

TP

Pour notre premier TP on souhaite afficher les éléments passés en argument lors du lancement de notre programme.

```
node server.js --name toto --message ‘Mon message!!!’
```

Les bases

```
process.argv.forEach((val, index) => {
    console.log(`${index}: ${val}`);
});
```

Les bases

De base chaque fichier est scopé (notion d'encapsulation)

L'objet module va nous permettre de créer un module javascript afin d'exposer tout ou partie du code de notre fichier

On va alors pouvoir utiliser notre module dans d'autres modules (fichier)

```
module.exports.log = myLogFunction;
```

Les bases

TP

On souhaite créer un module logger, afin de l’utiliser dans notre module principal

Les bases

Pour utiliser notre module externe, il suffit de déclarer son utilisation grâce à l'instruction **require**

```
const myLogger = require('./logger');
```

Les bases

Node fournit un certain nombre de modules tel que

- path
- os
- fs
- events
- http
- ...

Les bases

TP

Utiliser un setInterval et le module ‘os’ pour afficher la mémoire disponible sur votre machine avec un refresh toute les secondes

Les bases

Node met à disposition des méthodes synchrones et asynchrones

Il est préférable d'utiliser les méthodes asynchrones afin de respecter le modèle non bloquant préconisé par Node

Les méthodes asynchrones utilisent une méthode de callback qui sera appelée automatiquement à la fin du traitement

Les bases

TP

Utiliser le module ‘fs’ pour afficher le contenu du dossier courant (readdir), faîtes un appel synchrone puis un appel asynchrone avec un callback

Les bases

Node est basé sur le concept d'événement

Un événement signale que quelque chose est arrivé

On utilise la classe EventEmitter pour envoyer / recevoir un event

emit pour envoyer un event

on + callback pour écouter un type d’event

Les bases

```
const EventEmitter = require('events');
class Logger extends EventEmitter {
    log(message) {
        console.log(message);
        this.emit('messageLogged');
    }
module.exports = Logger;
```

Les bases

```
const Logger = require('./logger');
const logger = new Logger();

logger.on("messageLogged", (args) => {
   console.log("caught...")
});
logger.log("mon message");
```

Les bases

Node permet la création d'un serveur web en quelques lignes de code

Peut donc être utilisé pour réaliser un serveur web ou une API REST

Néanmoins le code peut vite devenir énorme

On utilise alors express (basé sur le module http) qui permet de simplifier la mise en place des routes

Les bases

```
const http = require('http');
const server = http.createServer((req, res) => {
        if (req.url === '/') {
              res.write('HELLO WORLD');
              res.end();
          }
});
server.listen(3000);
```

Les bases

Avant d'aller plus loin, nous allons créer notre premier projet en utilisant les commandes ci-dessous:

On va alors obtenir un dossier contenant un fichier package.json

```
mkdir express-demo
cd express-demo
npm init
```

API REST

Pour nous simplifier la vie nous allons utiliser le framework Express pour créer notre API

Il suffit de l'installer avec la ligne ci-dessous

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

Les bases

```
const express = require('express');
const app = express();


app.get('/', (req, res) => {
    res.send("HELLO WORLD");
});
const port = process.env.PORT || 3000;
app.listen(port, () => console.log(`Listening on port ${port}...`));
```

API REST

Pour éviter de faire un arrêt/relance de notre programme à chaque modification nous allons utiliser Node Monitor

```
npm install -g nodemon
```

```
nodemon app.js
```

API REST

Pour récupérer les paramètres de route et les paramètres de requête on utilise:

```
app.get('/api/posts/:year/:month', (req, res) => {
    res.send(`${req.params.year} ${req.query.sortBy}`);
});
```

API REST

Pour récupérer les paramètres d'une requête POST on utilise:

```
app.post('/api/posts', (req, res) => {
    res.send(`${req.body.name}`);
});
```

API REST

Pour valider les paramètres en entrées d'une requête, on va utiliser le package **joi**

```
npm install --save joi
```

API REST

Exemple de validation basique:

```
const Joi = require('joi');
app.use(express.json());
app.post('/api/posts', (req, res) => {
    const schema = {
        name: Joi.string().min(3).required()
    };
    const result = Joi.validate(req.body, schema);
    if (result.error) return res.status(400).send(`${result.error.details[0].message}`);
    res.send(`${req.body.name}`);
});
```

API REST

Sur le même principe que les méthodes **get**/**post**, express propose les méthodes **put** et **delete**

```
app.put('/api/posts', (req, res) => {
});
```

```
app.delete('/api/posts', (req, res) => {
});
```

API REST

Gestion des erreurs

```
app.use((req, res, next) => {
    const error = new Error('Not found...');
    error.status = 404;
    next(error);
});

app.use((err, req, res, next) => {
    res.status(err.status || 500);
    res.json({error: {message: err.message}});
});
```

API REST

TP

Créer la base d’une API permettant de prendre rendez-vous chez un médecin

Persistance de données

Pour simplifier l'accès à la base de données, nous allons utiliser Mongoose

```
npm install --save mongoose
```

Persistance de données

Pour simplifier l'accès à la base de données, nous allons utiliser Mongoose

```
const mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/test');

const catSchema = new mongoose.Schema({
name: String
});
const Cat = mongoose.model('Cat', catSchema);
app.post('/api/cats', (req, res) => {
      const kitty = new Cat({ name: 'Zildjian' });
      kitty.save().then(() => console.log('meow'));
      res.status(201).send();
});
```

Persistance de données

On peut facilement récupérer les données

```
app.get('/api/cats', (req, res) => {
  Cat.find((err, kittens) => {
    if (err) return console.error(err);
    res.status(200).send(kittens);
  });
});
```

API REST

TP

Poursuivre l’API en modélisant les données nécessaires

Authentification

Afin de pouvoir identifier chaque utilisateur, nous allons utiliser une authentification par jeton (JWT)

```
npm install --save jsonwebtoken
```

Authentification

Création d'un token

```
const jwt = require('jsonwebtoken');

const jwtKey = process.env.JWT_KEY || 'secret';
const token = jwt.sign({email: user.email, id: user.id}, jwtKey,
{expiresIn:'1h'});
```

Authentification

Récupération du token via un middleware

```
const jwt = require('jsonwebtoken');
module.exports = (req, res, next) => {
   try {
       const jwtKey = process.env.JWT_KEY || 'secret';
       const token = req.headers.authorization.split(' ')[1];
       const decoded = jwt.verify(token, jwtKey, null);
       req.userData = decoded;
       next();
   } catch (error) {
       res.status(401).json({message: 'Unauthorized'});
   }
};
```

Authentification

Utilisation du middleware pour protéger une route

```
const checkAuth = require('./check-auth');
app.get('/', checkAuth, (req, res) => {
...
});
```

WebSocket

Les WebSockets permettent de créer des interfaces qui peuvent être mises à jour en temps réel (chat, système de commandes…)

Nous allons utiliser la librairie socket.io

```
npm install --save socket.io
```

WebSocket

TP

Ajouter la possibilité de remonter en temps réel le planning des médecins

Test avec Mocha

Mocha est un framework de test

```
npm install -D mocha
```

```
module.exports = (req, res, next) => {
    req.requestedTime = new Date.now();
    next();
};
```

Test avec Mocha

Il suffit alors de créer un dossier test, contenant un fichier request-time.spec.js

Mocha utilise la variable globals

On crée alors une Suite qui va contenir nos tests à jouer

```
describe('requestTime middleware', () => {
    //Tests go here
});
```

Test avec Mocha

Mocha ne fournit pas de mécanisme d'assertion

On utilise le module **assert** de node

```
const assert = require('assert');
const requestTime = require('../lib/request-time');
describe('requestTime middleware', () => {
it('should add a requestTime property to the req parameter', () => {
            const req = {};
            requestTime(req, null, ()=>{});
            assert.ok(req.requestTime > 0);
      });
});
```

Test avec Mocha

```
$ mocha
  requestTime middleware
    ✓ should add a timestamp `requestTime` prop to `req`


  1 passing (5ms)
```

Test avec Mocha

Pour faciliter nos tests d’intégration sur un serveur basé sur Express, nous allons utiliser le package supertest

```
npm install -D supertest
```

Test avec Mocha

Code à tester

```
const requestTime = require('../lib/request-time');

app.use(requestTime);

app.get('/unix-timestamp', (req, res) => {
        res.json({timestamp: Math.floor(req.requestTime / 1000)});
});
```

Test avec Mocha

Test d'intégration

```
const assert = require('assert');
const app = require('../app');
const request = require('supertest');
describe('GET /unix-timestamp', () => {
        it('should respond with JSON object containing timestamp', (done) => {
            return request(app).get('/unix-timestamp').expect(200).then(
                    res => {
                        assert.ok(res.body.timestamp < 1e10);
                    });
        });
});
```

Test avec Mocha

```
$ node_modules/.bin/mocha

  GET /unix-timestamp
    ✓ should respond with JSON object containing timestamp

  requestTime middleware
    ✓ should add a timestamp `requestTime` prop to `req`


  2 passing (32ms)
```