Node.js

m2iformation.fr

Sommaire

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Introduction à Node.js

Qu'est-ce que Node.js ?

- Node.js est un environnement d'exécution JavaScript multiplate-forme **côté serveur**
- Node.js est construit sur le moteur JavaScript **V8** de Google Chrome
- Il est principalement utilisé pour **créer des serveurs web** - mais il n'est pas limité à cela
- Il a été créé en **2009** par **Ryan Dahl**

Fonctionnement

- Une application Node.js s'exécute dans **un seul processus**, sans créer un nouveau thread pour chaque requête
- Node.js fournit un ensemble de primitives **d'E/S asynchrones** dans sa bibliothèque standard qui empêchent le code JavaScript de se bloquer
- Cela permet à Node.js de gérer des **milliers de connexions simultanées** avec un seul serveur sans avoir à gérer la concurrence des threads

Différence navigateur / Node.js

Node.js	Navigateur
Environnement d'exécution JavaScript basé sur le moteur V8 de Chrome	Environnement d'exécution JavaScript intégré au navigateur web
Permet de créer des applications web côté serveur	Permet de créer des applications web côté client
Utilise le module npm pour gérer les dépendances	Utilise le module CDN ou les balises script pour charger les bibliothèques
Supporte les fonctionnalités ES6+ sans transpilation	Nécessite souvent une transpilation pour assurer la compatibilité entre les navigateurs
Peut accéder au système de fichiers et aux ressources du serveur	Ne peut pas accéder au système de fichiers et aux ressources du serveur pour des raisons de sécurité

V8 Engine

- Le moteur V8 est le nom du moteur JavaScript qui équipe Google Chrome
- Il indépendant du navigateur dans lequel il est hébergé
- Node.js utilise le moteur V8 pour analyser et exécuter le code JavaScript sur le serveur
- Le moteur V8 est écrit en C++
- Le moteur V8 compile le code JavaScript en interne avec une compilation juste à temps (JIT) pour optimiser les performances d'exécution

Installation

- Node.js peut s'installer de 2 façons
  1. Via un gestionnaire de package
  2. Via un exécutable

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

npm

Présentation

- est le gestionnaire de paquets standard pour Node.js
- En septembre 2022, plus de 2,1 millions de paquets ont été répertoriés dans le registre npm
- Il existe des alternatives à comme `yarn` et `pnpm`

Initialiser un projet Node.js

- `npm init` (`--yes`) permet d'initialiser un nouveau projet npm
- Cette commande créé un fichier **package.json** avec toutes les informations du projet :
  - nom
  - version
  - description
  - auteur
  - ...

package.json

- npm gère les téléchargements des **dépendances** d'un projet à l'aide d'un fichier nommé `package.json` à la racine du répertoire
- un fichier **package.json** peut être décrit comme un **manifeste** d'un projet qui inclut les paquets dont il dépend:
  - des informations sur le contrôle de source
  - métadonnées spécifiques (nom, description, auteur)

package.lock

- Le `package-lock.json` est un fichier contenant une représentation à un instant *t* de l'arbre de dépendances d'un projet JavaScript
- L'intérêt du `package-lock` est le suivant: avoir une représentation déterministe du dossier **node_modules** sans avoir à commiter le node_modules

Installation de packages

- Installer tous les packages du projet : `npm install`
- Installer un package : `npm install <package-name>`
  - `-O, --save-optional` : installation en production dans package.json
  - `-D, --save-dev` : installation dans package.json en dev
  - `-O, --save-optional` : installation dans package.json optionnel
  - `--global, -g` : installation globale

Mise à jour des packages

- Mettre à jour les paquets :
  - `npm update` : mise à jour des packages de production
  - `npm update <package-name>` : mise à jour d'un package spécifique
  - `npm update --dev` : mise à jour des packages de dev
  - `npm update -g` : mise à jour des packages globaux

Lister les packages

- `npm list` : lister les dépendances du projet
- `npm list -g --depth 0` lister toutes les dépendances installées globalement
- `npm view` : lister les versions les plus récentes des dépendances du projet
- `npm outdated` : lister les dépendances périmées

Suppression et vulnérabilités

- `npm rm <package>` : supprime un package
- `npm audit` Scanne et liste les vulnérabilités des packages
- `npm audit fix` Fixe les vulnérabilités des packages

Exécuter des commandes

- Le fichier **package.json** prend en charge un format permettant de spécifier des tâches en ligne de commande : `npm run <task-name>`
- Les commandes se situent dans l'objet **scripts**

```
{
  "scripts": {
    "watch": "webpack --watch --progress --colors --config webpack.conf.js",
    "dev": "webpack --progress --colors --config webpack.conf.js",
    "prod": "NODE_ENV=production webpack -p --config webpack.conf.js"
  }
}
```

npx

- `npx` est un outil qui permet d'exécuter des paquets npm sans les installer globalement
- Il permet d'utiliser des versions spécifiques ou des tags d'un paquet sans modifier le fichier **package.json**
- Il évite de polluer l'espace global avec des paquets inutilisés ou obsolètes

```
npx create-react-app myreactapp
```

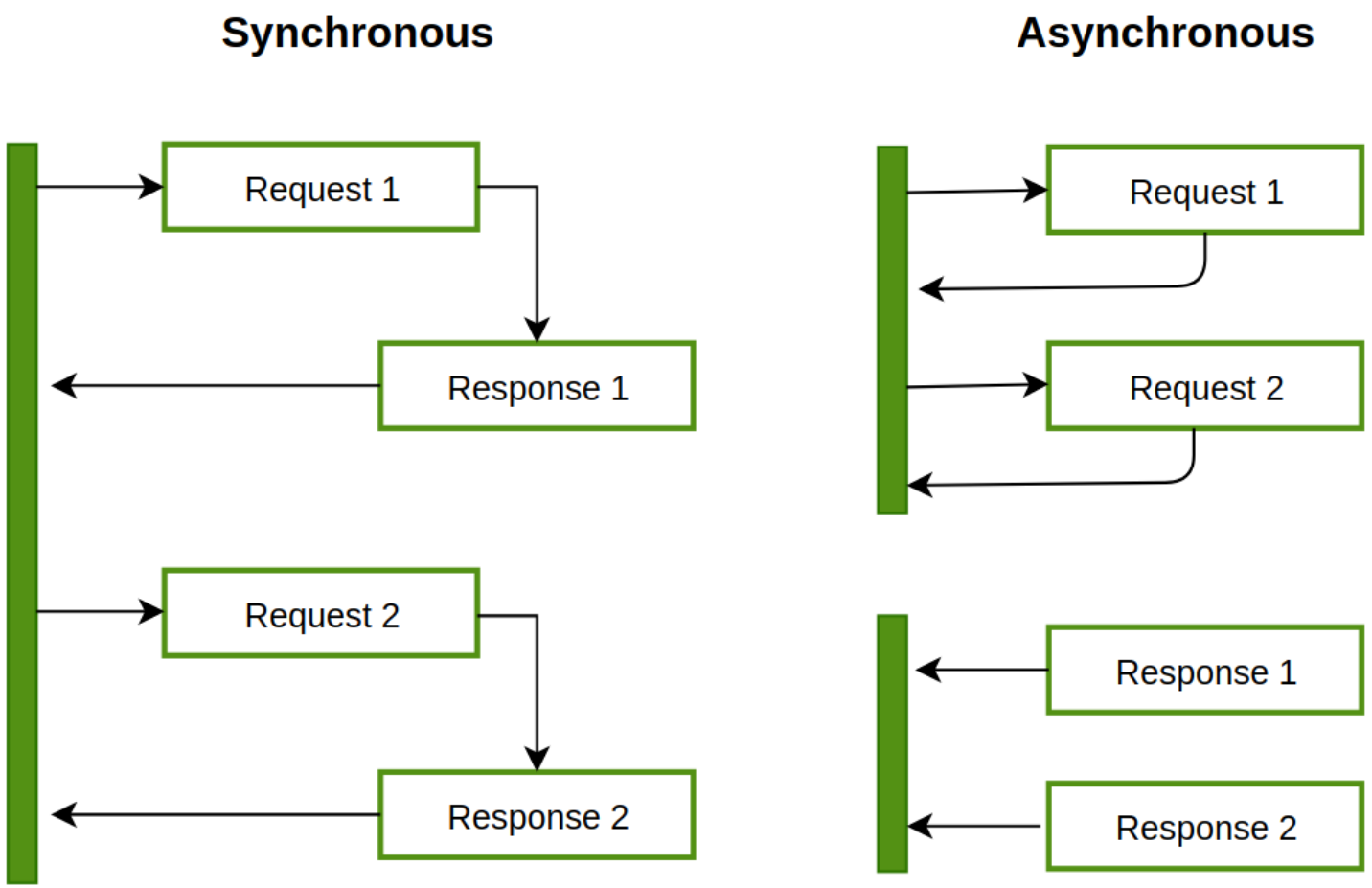
UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Asynchrone

Définition

- Les **ordinateurs sont asynchrones** par conception
- Asynchrone signifie que des choses peuvent **se produire indépendamment du flux principal** du programme
- **JavaScript est synchrone** par défaut et ne comporte qu'un seul fil d'exécution (un thread)
- Cela signifie que le code **ne peut pas créer de nouveaux threads** et s'exécuter en parallèle

Asynchrone vs Synchrone



Asynchrone Node.js

- Le navigateur offre un moyen d'exécuter du code asynchrone en fournissant un ensemble d'API
- Toutes les méthodes d'E/S de la bibliothèque standard Node.js proposent des versions asynchrones, qui ne sont pas bloquantes, et acceptent des fonctions de rappel (**callback**)
- Certaines méthodes ont également des équivalents bloquants, dont les noms se terminent par **Sync**

Callback

- Un callback est une **fonction** simple qui est **transmise comme valeur à une autre fonction** et qui ne sera exécutée que lorsque l'événement se produira
- JavaScript a des **fonctions de première classe**, qui peuvent être assignées à des variables et transmises à d'autres fonctions (appelées **fonctions d'ordre supérieur**)

```
const fs = require("fs");
fs.readFile("/file.md", (err, data) => {
  if (err) throw err;
});
```

Gestion des erreurs

- Node.js adopte d'initialiser l'objet d'erreur en première position dans les callbacks, c'est ce qu'on appelle **error-first callback**

```
const fs = require('fs');

fs.readFile('/file.json', (err, data) => {
  if (err) {
    // handle error
    console.log(err);
    return;
  }
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Modules

Modules de base de Node.js

Module	Description
http	http comprend des classes, des méthodes et des événements permettant de créer un serveur http Node.js.
url	url comprend des méthodes pour la résolution et l'analyse d'URL
path	path module comprend des méthodes pour traiter les chemins de fichiers
fs	fs module inclut des classes, des méthodes et des événements pour travailler avec les entrées/sorties de fichiers
util	Le module util comprend des fonctions utilitaires utiles aux programmeurs
os	fournit des informations et des fonctionnalités liées au système d'exploitation sur lequel s'exécute le programme nodejs

Importer des modules

- Pour utiliser les modules de Node.js ou les modules npm il faut les importer à l'aide de `require()`
- La fonction `require()` renverra un objet, une fonction, une propriété ou tout autre type JavaScript, en fonction de ce que le module spécifié renvoie

```
const circle = require("./circle.js");
console.log(`The area of a circle of radius 4 is ${circle.area(4)}`);
```

require()

- Les modules peuvent être importés à partir de `node_modules`
- Les chemins relatifs sont résolus indépendamment du système d'exploitation quand le chemin commence par `/`, `./`, ou `../`

```
// Importation d'un module local depuis le répertoire courant
const myLocalModule = require("./path/myLocalModule.js");

// Importation d'un module à partir de node_modules ou Node.js
const crypto = require("crypto");
```

Exporter des modules

- `module.exports` est un objet spécial qui est inclus par défaut dans chaque fichier JavaScript de l'application Node.js
- `module` est une variable qui représente le module actuel, et `exports` est un objet qui sera exposé en tant que module

```
const { PI } = Math;

exports.area = (r) => PI * r ** 2;

exports.circumference = (r) => 2 * PI * r;
```

CommonJS ou ESM

Node.js prend entièrement en charge les modules ECMAScript (`import` , `export`) et assure l'interopérabilité entre eux et son format de module d'origine, CommonJS (`require()` , `module.exports`)
Pour utiliser la syntaxe de ES6, il faut soit:

- Ajouter l'extension `.mjs` à un fichier js
- Ajouter la clé `"type": "module"` dans le fichier package.json

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Manipulation de fichiers

Informations sur les fichiers

- Chaque fichier est accompagné d'un ensemble de détails que nous pouvons inspecter à l'aide de Node.js
- En particulier, en utilisant la méthode `stat()` fournie par le module fs

```
const fs = require("fs");

fs.stat("/Users/joe/test.txt", (err, stats) => {
  // Accès aux informations du fichier
});
```

Méthodes de l'objet stats

- `stats.isFile()` et `stats.isDirectory()` permettent de savoir si l'élément est un fichier ou un dossier
- `isSymbolicLink()` pour savoir si le fichier est un lien symbolique
- `stats.size` la taille du fichier en octet

```
stats.isFile(); // true
stats.isDirectory(); // false
stats.isSymbolicLink(); // false
stats.size; // 1024000 //= 1MB
```

Gestion des répertoires

Le module `path` permet d'avoir des informations d'un chemin d'accès à l'aide de ces méthodes :

- `dirname` : permet d'obtenir le dossier parent d'un fichier
- `basename` : permet d'obtenir la partie du nom du fichier
- `extname` : permet d'obtenir l'extension du fichier

```
const path = require("path");
const notes = "/users/joe/notes.txt";

path.dirname(notes); // /users/joe
path.basename(notes); // notes.txt
path.extname(notes); // .txt
```

Manipulation de chemins

- `path.join()` permet de joindre deux ou plusieurs parties d'un chemin

```
path.join("/", "users", name, "notes.txt");
```

- `path.resolve()` permet de calculer le chemin absolu d'un fichier à partir d'un chemin relatif

```
path.resolve("joe.txt"); // '/Users/joe/joe.txt' depuis le répertoire courant
```

Lire un fichiers

- `fs.readFile()` permet de lire un fichier en lui passant le chemin du fichier, l'encodage et une fonction de rappel qui sera appelée avec les données du fichier (et l'erreur) :

```
const fs = require("fs");

fs.readFile("/Users/joe/test.txt", "utf8", (err, data) => {
  if (err) {
    console.error(err);
    return;
  }
  console.log(data);
});
```

Lire un fichier avec les promesses

```
const fs = require("fs/promises");

async function doSomething(file) {
  try {
    const data = await fs.readFile(file, { encoding: "utf8" });
    console.log(data);
  } catch (err) {
    console.log(err);
  }
}
doSomething();
```

Information concernant la lecture

- Les trois programmes `fs.readFile()`, `fs.readFileSync()` et `fsPromises.readFile()` lisent **le contenu intégral du fichier en mémoire** avant de renvoyer les données
- Cela signifie que les gros fichiers auront un **impact majeur sur la consommation de mémoire** et la vitesse d'exécution du programme
- Dans ce cas, il est préférable de lire le contenu du fichier à l'aide de flux (stream)

Paramètre de lecture/écriture de fichiers

Indicateur	Description	Le fichier est créé s'il n'existe pas
r+	Ce drapeau ouvre le fichier pour la lecture et l'écriture.	❌
w+	Cet indicateur ouvre le fichier pour la lecture et l'écriture et positionne le flux au début du fichier	✅
a	Cet indicateur ouvre le fichier en écriture et positionne également le flux à la fin du fichier	✅
a+	Cet indicateur ouvre le fichier en lecture et en écriture et positionne également le flux à la fin du fichier	✅

Ecrire dans un fichier

La manière la plus simple d'écrire dans des fichiers en Node.js est d'utiliser l'API `fs.writeFile()`

```
const fs = require("fs");

const content = "Vive les chips";

fs.writeFile("/Users/toto/test.txt", content, (err) => {
  if (err) {
    console.error(err);
  }
  // Traitement réussi
});
```

Ecrire dans un fichier avec une promesse

```
const fs = require("fs/promises");

async function example() {
  try {
    const content = "Vive les frites";
    await fs.writeFile("/Users/toto/test.txt", content);
  } catch (err) {
    console.log(err);
  }
}
example();
```

Informations complémentaires

Par défaut, cette API remplacera le contenu du fichier s'il existe déjà.

- Pour ajouter du contenu à un fichier avec un flag :

```
fs.writeFile("/Users/joe/test.txt", content, { flag: "a+" }, (err) => {});
```

- Pour ajouter du contenu en fin de fichier on peut utiliser:

```
fs.appendFile("file.log", content, (err) => {
  if (err) {
    console.error(err);
  }
  // done!
});
```

Gestion de dossiers

- Vérifier l'existence d'un dossier: `fs.access()`, `fsPromises.access()`
- Créer un dossier: `fs.mkdir()`, `fsPromises.mkdir()`
- Lister les éléments d'un dossier: `fs.readDir()` `fsPromises.readDir()`
- Renommer un dossier: `fs.rename()`, `fsPromises.rename()`
- Supprimer un dossier: `fs.rmdir()`, `fsPromises.rmdir()`
- Supprimer de manière récursive:

```
fs.rm(dir, { recursive: true, force: true }, (err) => {});
```

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Utilisation du module HTTP

Créer un serveur

La création d'un serveur se fait avec la fonction `createServer` :

```
const http = require("http");

http
  .createServer((request, response) => {
    // la magie se créée ici !
  })
  .listen(8080);
```

La fonction passée à createServer est **appelée une fois pour chaque requête** HTTP effectuée sur ce serveur (un gestionnaire de requêtes)

La méthode `listen()` permet d'écouter le serveur sur un port

Méthode, URL et header

La méthode et l'url sont des propriétés de la requête :

```
const { method, url } = request;
```

- **l'url** contient l'url complet sans le serveur, le protocole et le port
- **la méthode** contient un des verbes HTTP sous forme de chaîne
- **Le header** est un objet de la requête :

```
const { headers } = request;
const userAgent = headers["user-agent"];
```

Body Request

Lors d'une requête HTTP en **PUT** ou en **POST**, il est possible de récupérer des éléments du corps de la requête. Le flux présent dans le corps de la requête peut être lu en écoutant les événements `data` et `end` du flux.

```
let body = [];
request
  .on("data", (chunk) => {
    body.push(chunk);
  })
  .on("end", () => {
    body = Buffer.concat(body).toString();
    // Body contient la totalité du corps de la requête sous forme de chaîne
  });
```

Entête de la réponse HTTP

Par défaut, la réponse HTTP renvoie le code 200. Pour définir un code différent il suffit de modifier la propriété.

```
response.statusCode = 404;
```

Il est également possible d'éditer l'entête HTTP avec le mutateur `setHeader()`.

```
response.setHeader("Content-Type", "application/json");
```

On peut également utiliser le raccourci `writeHead()`.

```
response.writeHead(200, { "Content-Type": "application/json" });
```

Envoyer une réponse HTTP

L'écriture d'une de la réponse se fait avec la méthode `write()`

```
response.write("<h1>Hello, World!</h1>");
response.end();
```

La méthode `end()` peut également prendre des données facultatives en paramètres

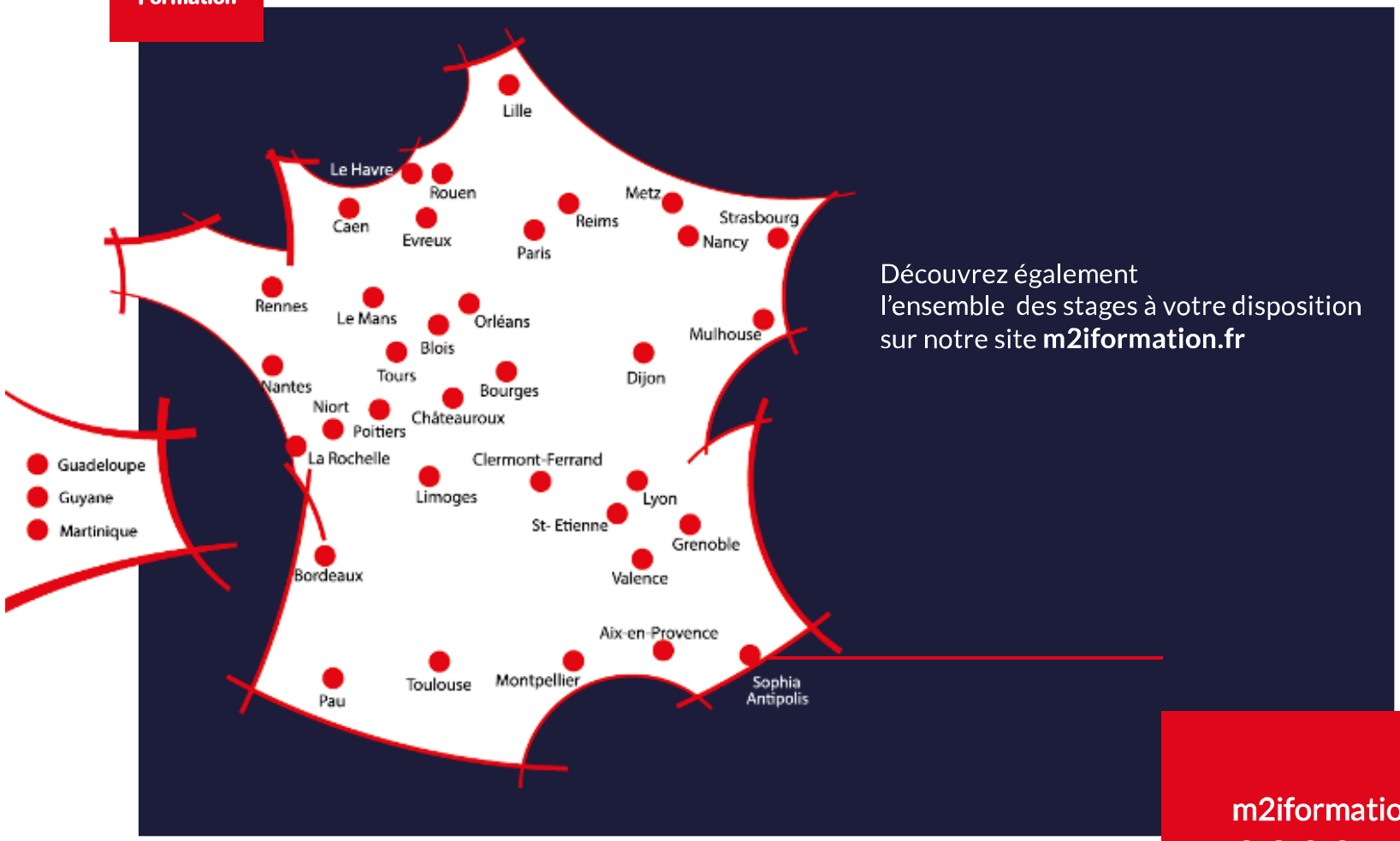
```
response.end("<h1>Hello, World!</h1>");
```

⚠️ Il est important **de définir le statut et les en-têtes** avant de commencer à écrire dans le corps de la réponse

UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

Merci pour votre attention

Des questions ?



Découvrez également
l'ensemble des stages à votre disposition
sur notre site **m2iformation.fr**

m2iformation.fr