Pratique NodeJS

Sommaire Pratique NodeJS :

- Installation de NodeJS
- Choix de la version
- Ajouts des modules natifs
- Vérification de la version
- REPL NodeJS
- Création d'un premier script
- Ajouts de packages
- Critères justifiant l'utilisation de packages
- NPM
- Initialisation d'un Projet
- Configuration du projet
- Gestion des dépendances
- Environnement de travail
- Création d'une première application
- Installation de Postman, tests du serveur et alternatives.
- Installation et test de nodemon
- Lecture de fichiers
- Synchrone / Asynchrone

Installation NodeJS :

Sur MacOS :

- En utilisant Bash sur Mac
- En utilisant Homebrew sur Mac
- Installation brew :

```
/bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install.sh)"
```

puis : brew install node

Sur Windows :

Téléchargez le programme d'installation Windows directement depuis le site Web nodejs.org.

Sur Linux :

Sur Ubuntu 18.04+, vous pouvez installer Node en utilisant les commandes suivantes :

```
sudo apt update
sudo apt install nodejs
```

Choix de la version :

- LTS est l'acronyme de Long-Term Support (Support à long terme). Notez que LTS est décrit comme « Recommandé pour la plupart des utilisateurs ».

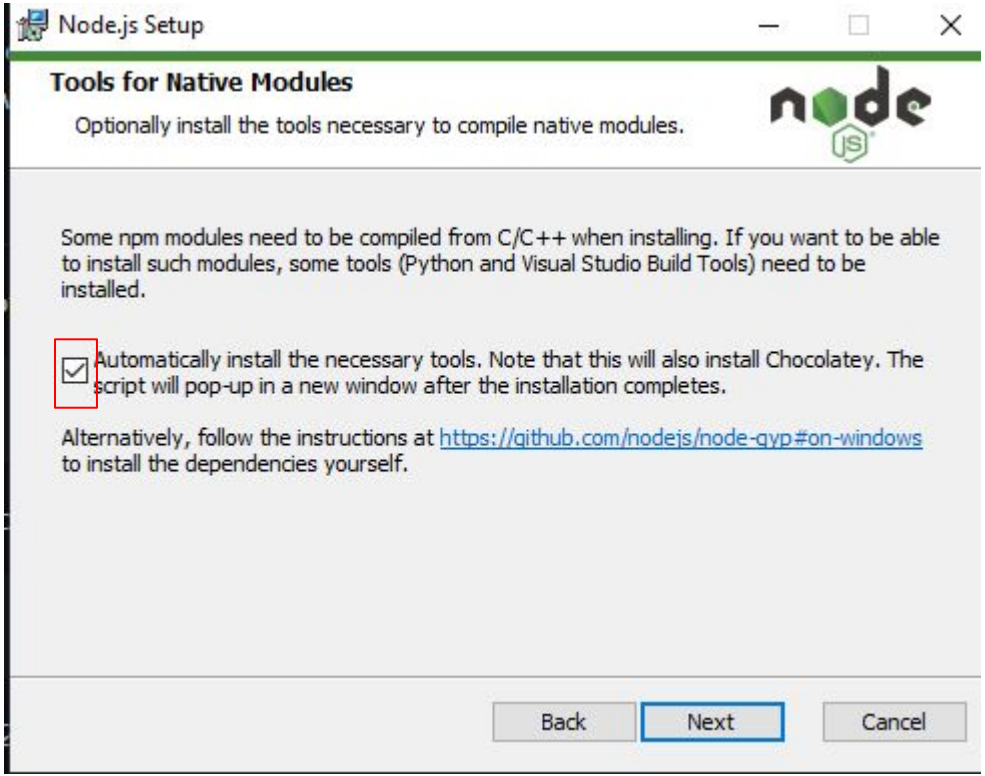
  LTS est conçu pour une utilisation d'entreprise où des mises à jour fréquentes peuvent ne pas être possibles ou ne pas être souhaitées.

- "Actuel" correspond au code source qui est en cours de développement.

  Des ajouts de fonctionnalités et des modifications avec rupture peuvent se produire. Le code doit adhérer à la gestion de versions sémantique.

IMMERSALIS CONSULTING

Ajouts des modules natifs :



IMMERSALIS CONSULTING

Vérification de la version :

Dans un terminal :

node -v

ou

node --version

```
PS C:\Users\Proprietaire> node -v
v16.14.2
```

REPL NodeJS :

Node.js dispose d'un mode de boucle de lecture/évaluation (**REPL**) intégré qui est utile pour l'évaluation et l'expérimentation rapides du code.

Le mode **REPL** est un environnement de console interactif dans lequel vous pouvez entrer du code JavaScript. Node.js interprète ce code, l'exécute et affiche la sortie.

Le mode REPL Node.js fonctionne comme suit :

- **Read**: lit et analyse le code JavaScript d'entrée de l'utilisateur (ou affiche une erreur si le code n'est pas valide).
- **Eval**: évalue le code JavaScript entré.
- **Print**: affiche les résultats calculés.
- **Loop**: Effectue une boucle et attend que l'utilisateur entre une nouvelle commande (ou s'arrête si l'utilisateur entre Ctrl-c deux fois).

REPL NodeJS :

Pour démarrer le mode REPL, exécutez le programme `node` dans Azure Cloud Shell :

Bash — Copier

```
node
```

L'environnement REPL s'ouvre. La sortie suivante s'affiche :

text — Copier

```
>
```

Essayez de taper le `console.log('Hello World, from the REPL.')` de code à l'intérieur de la console REPL et affichez la sortie. Ce code imprime un « Hello World à partir du REPL ». message dans la sortie REPL :

text — Copier

```
> console.log('Hello World, from the REPL.')
Hello World, from the REPL.
```

Pour quitter l'environnement REPL, entrez **Ctrl-c** deux fois :

text — Copier

```
>
(To exit, press ^C again or type .exit)
>
```

IMMERSALIS CONSULTING
IMMERSIVE TECHNOLOGIES FOR EVERYONE

Création d’un premier script :

Créer un script : **index.js** contenant : **console.log(“premier script”);**

Exécuter le script en tapant depuis le terminal (à la racine du dossier contenant le script) : **node index.js**

Ajouts de packages :

Node.js est fourni avec de nombreuses bibliothèques principales qui s'occupent de tout :

- gestion de fichiers
- protocole HTTP
- compression des fichiers
- …

Il existe cependant un très large écosystème de bibliothèques tierces.

Avec npm (Node Package Manager), vous pouvez installer ces bibliothèques et les utiliser dans votre application.

Critères justifiant l'utilisation de packages :

- Obtenir du meilleur code (ex : lib de sécurité, compliqué à développer)

- Gain de temps (ex : bibliothèques d'utilitaires ou de composants de l'interface utilisateur, inutile de refaire du code déjà fait)

- Maintenance (devoir débugger ou mettre à jour la lib)

Critères d'évaluation d'un package :

- Taille/performances : nombre de dépendances / limitations matériels / débit.

- Licences : Les licences peuvent être un facteur à considérer si vous produisez des logiciels que vous avez l'intention de vendre. Si vous disposez d'une licence sur une bibliothèque tierce et que le créateur de la bibliothèque n'autorise pas l'inclusion de celle-ci dans un logiciel commercialisé, il peut vous placer dans une situation juridiquement problématique.

- Maintenance active : Si votre package s'appuie sur une dépendance qui est dépréciée ou qui n'a pas été mise à jour depuis longtemps, ce peut être un problème.

NPM :

- NPM est l'écosystème de paquets de Node.js.

- C'est le plus grand écosystème de toutes les bibliothèques open source au monde, avec plus d'un million de paquets.

- NPM est livré avec un utilitaire de ligne de commande.

Vous pouvez chercher des packages à installer en ligne de commande sur : https://www.npmjs.com/

Vous pouvez également gérer les versions de votre paquet, examiner les dépendances et même configurer des scripts personnalisés dans vos projets grâce à cet utilitaire de ligne de commande.

Installation de paquets via NPM :

Lorsque vous installez Node.js, NPM est automatiquement installé avec lui.

Commande pour installer un paquet avec NPM :

npm install <package-name>

Vous pouvez aussi installer plusieurs paquets à la fois :

npm install <pkg-1> <pkg-2> <pkg-3>

Vous pouvez également spécifier le flag **-g** (global) si vous souhaitez installer un paquet dans le contexte global. Cela vous permet d'utiliser le paquet n'importe où sur votre machine.

Obtenir des informations sur un package :

```
https://www.npmjs.com/package/<package name>
```

ou

```
npm view <package name>
```

Rechercher des packages :

Registres : Il peut s'agir par exemple d'un registre global comme le registre npm. Vous pouvez héberger vos propres registres, qui peuvent être privés ou publics.

Dépôts : Au lieu de pointer vers un registre, il est possible de donner une URL GitHub et d'installer le package à partir de là.

Fichiers : Vous pouvez installer un package depuis un dossier local ou un fichier zippé.

Répertoires : Vous pouvez aussi l'installer directement depuis un répertoire.

Possibilités des commandes NPM :

Gérer les dépendances : Un certain nombre de commandes couvrent l'installation, la suppression et le nettoyage après l'installation des packages. D'autres commandes permettent aussi de mettre à jour les packages.

Exécuter des scripts : L'outil npm peut vous aider à gérer les flux dans le développement de votre application. Exécuter des tests, générer du code ou exécuter des outils d'amélioration de la qualité pour effectuer le linting de votre code sont des exemples de flux d'application.

Configurer l'environnement : Vous pouvez configurer beaucoup de choses, comme l'emplacement des installations et depuis où les packages doivent être installés. Il est courant d'avoir une configuration qui vous permet d'installer des packages depuis le registre npm global et aussi depuis un registre spécifique à l'entreprise.

Créer et publier des packages : Il existe plusieurs commandes qui peuvent vous aider à effectuer des tâches comme créer un package compressé et envoyer (push) un package vers un registre.

La liste détaillée des commandes est sur : `npm --help` dans le terminal.

Mettre à jour un package en utilisant npm :

```
npm update <name of package>@<optional argument with version number>
```

- Version majeure. J'accepte d'effectuer une mise à jour vers la dernière version majeure dès qu'elle est publiée. J'accepte le fait que je devrai peut-être modifier le code de mon côté.
- Version mineure. J'accepte l'ajout d'une nouvelle fonctionnalité. Je n'accepte pas que mon code ne puisse plus fonctionner correctement.
- Version corrective. Les seules mises à jour que j'accepte sont des correctifs de bogues.

Type	**Changement**
Version majeure	1.0.0 passe à 2.0.0
Version mineure	1.1.1 passe à 1.2.0
Version corrective	1.0.1 passe à 1.0.2

Initialisation et Configuration d’un projet :

Le fichier package.json est le fichier manifeste de votre projet Node.js.

- Il contient des informations de métadonnées sur votre projet.

- Il détermine également des aspects comme la façon dont vos dépendances sont gérées, quels fichiers sont placés dans un package destiné à npm, etc.

Un fichier package.json ne se crée pas manuellement.

Il est le résultat de l'exécution de la commande `init`.

Il existe deux moyens principaux d'exécuter cette commande :

- `npm init`.

Cette commande démarre un assistant qui vous invite à entrer des informations sur le projet :

Nom, Version, Description, Point d'entrée, Commande de test, Dépôt Git, Mots clés, Créateur et Licence.

```
package name: (node)
version: (1.0.0)
description:
entry point: (index.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to C:\Users\Proprietaire\Node\package.json:

{
  "name": "node",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}


Is this OK? (yes)
```

- `npm init -y`.

Cette commande, avec l'indicateur `-y`, est une version plus simple de `npm init`.

Elle affecte des valeurs par défaut pour toutes les valeurs que vous devez fournir si vous exécutez `npm init`.

```
PS C:\Users\Proprietaire\Node2> npm init -y
Wrote to C:\Users\Proprietaire\Node2\package.json:

{
  "name": "node2",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Configuration du projet :

Contenu du fichier package.json

Tous les champs possibles du fichier package.json peuvent être considérés comme appartenant aux groupes suivants :

- **Méta-informations** : Ce groupe contient des méta-informations sur le projet, comme son nom, sa description, son créateur et ses mots clés.
- **Dépendances** : Il existe deux propriétés, `dependencies` et `devDependencies`, qui sont utilisées pour décrire les bibliothèques utilisées.
- **Scripts** : Vous pouvez placer dans cette section des scripts qui effectuent des opérations comme le démarrage, la génération, le test et le linting d'un projet.

Configuration du projet :

Scripts pour la gestion du projet

Vous devez configurer différents scripts et les nommer d'une certaine façon, conformément aux attentes de la communauté des développeurs et de différents outils :

- **Start** : Une commande start appelle `node` avec le fichier d'entrée comme argument. Elle peut se présenter comme suit : `node ./src/index.js`. Elle indique d'appeler `node` et d'utiliser le fichier d'entrée `index.js`.
- **Build** : Cette commande décrit comment générer le projet. Le processus de génération doit produire quelque chose que vous pouvez livrer.
- **Test** : Cette commande doit exécuter les tests pour votre projet. Si vous utilisez une bibliothèque de tests tierce, la commande doit appeler le fichier exécutable de la bibliothèque.
- **Lint** : Cette commande doit appeler un programme linter comme ESLint. Le *linting* recherche les incohérences dans le code. En général, un linter offre également la possibilité de les corriger (meilleures cohérence du code = meilleure productivité).

Configuration du projet :

- l'action `start` qui lance l'application.
- l'action `test` exécute les tests en utilisant le framework de test `jest`
- l'action `build` utilise le compilateur TypeScript `tsc` pour compiler le code TypeScript en un langage compréhensible par le navigateur
- l'outil de linting `eslint` recherche les incohérences et éventuellement les erreurs dans le code.

JSON

```
"scripts" : {
  "start" : "node ./dist/index.js",
  "test": "jest",
  "build": "tsc",
  "lint": "eslint"
}
```

Vous appelez des actions en entrant `npm run <action>`.
Il existe cependant des actions *spéciales* : `start` et `test`.
Elles sont spéciales, car vous pouvez omettre `run`.
Vous pouvez donc entrer `npm start` au lieu de `npm run start`, ce qui réduit légèrement le nombre de caractères.

Gestion des majs de dépendances du projet :

Les dépendances appartiennent à une des deux catégories suivantes :

- **Dépendances de production** : Les dépendances de production sont des dépendances dont vous avez besoin pour exécuter une application en production. Il s'agit par exemple d'un framework web avec lequel vous pouvez créer une application web.

- **Dépendances de développement** : Les dépendances de développement sont des dépendances dont vous avez besoin seulement pendant le développement votre application. Quand vous avez terminé le développement, vous n'en avez plus besoin. Il peut s'agir par exemple de bibliothèques de test, d'outils de linting ou d'outils de création de bundle.

Cette séparation n'est pas seulement conceptuelle. L'outil npm écrit dans un fichier manifeste en y ajoutant des entrées quand vous téléchargez des dépendances.

L'outil vous permet de différencier les deux types de dépendances en ajoutant un indicateur à la commande d'installation. L'indicateur place le nom de la dépendance et sa version dans une section appelée `dependencies` ou `devDependencies`.

Nettoyer les dépendances :

- **Désinstaller** : Pour désinstaller un package, exécutez la commande :

  `npm uninstall <name of dependency>`.

  Cette commande va supprimer le package du fichier manifeste et du répertoire node_modules.

- **Nettoyer** : Vous pouvez aussi exécuter la commande : `npm prune`.

  L'exécution de cette commande entraîne la suppression de toutes les dépendances dans le dossier **node_modules** qui ne sont pas listées comme dépendances dans le fichier manifeste.
  Cette commande peut être un bon choix pour supprimer plusieurs dépendances sans devoir exécuter la commande `uninstall` pour chaque dépendance.

  Pour cela, supprimez les entrées de la section `dependencies` ou `devDependencies`, puis exécutez la commande `npm prune`.

Environnement de travail :

- Des consoles pour chaque serveur ouvert
- Un IDE pour l'édition du code
- Un navigateur pour tester le front et le fonctionnement global en temps qu'utilisateur
- Un logiciel pour tester les requêtes HTTP

Première application :

Création d'un serveur avec le code minimum nécessaire dans index.js :

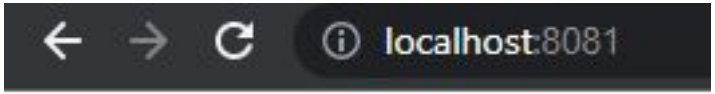
```
const http = require("http");

const server = http.createServer(function(req, res) {
  // A l'arrivée d'une requête
  console.log("Serveur running at http://127.0.0.1:8081/")
  res.write("Hello World!");
  res.end();
})

// Lancer le serveur (en écoute au port 8081)
  server.listen(8081)
```

Première application :

- Lancement du serveur : **node [nom du fichier]**

```
PS C:\Users\Proprietaire\Node> node .\index.js
Server running at http://127.0.0.1:8081/
```

- Consulter la réponse du serveur :



Hello World

Chargement des modules :

👉 **Core modules**

```
require('http');
```

👉 **Developer modules**

```
require('./lib/controller');
```

👉 **3rd-party modules (from NPM)**

```
require('express');
```

Notion d'événement pour Node :

IMMERSALIS CONSULTING

Utilisation de Postman :

POSTMAN

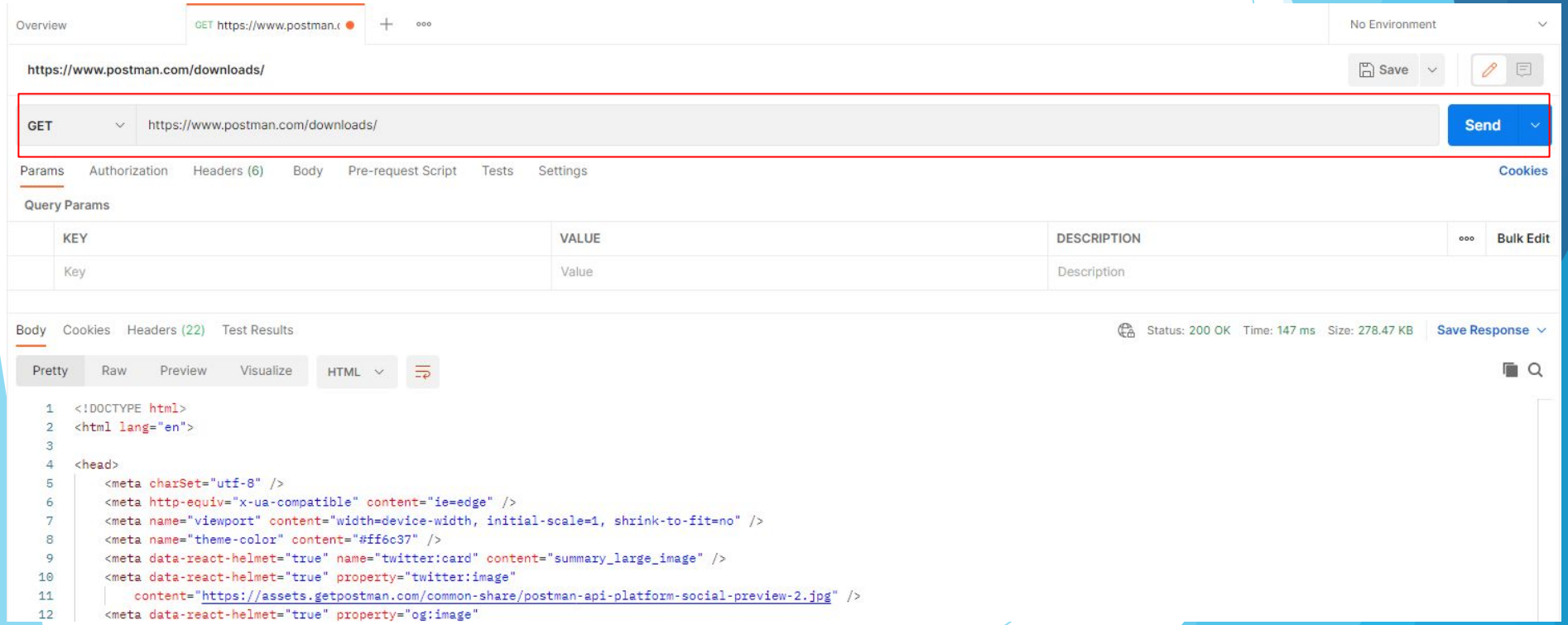
Postman est une application permettant de tester des API.

Postman regroupe chaque test d'API dans une collection, permettant de mutualiser leurs URLs et authentifications.

- Des variables pouvant changer selon l'environnement sélectionné.
- Une gestion de versions des tests et environnements.
- Des tests de performance.
- Importation et exportation en JSON.
- Exportation des tests pour qu'ils soient exécutés depuis différents clients HTTP .
- Authentification par JSON Web Token.
- API REST
- Une console de débogage qui garde en mémoire les requêtes et réponses des précédents appels lancés.
- Des scripts pouvant automatiser les tests en récupérant leurs résultats dans des variables.

Installation et Test Postman :

https://www.postman.com/downloads/



Alternatives à postman :

HTTPie : https://httpie.io/download (client http en ligne de commande)

Insomania : https://insomnia.rest/ (équivalent postman)

Installation de Nodemon :

Nodemon est aide au développement node.js en redémarrant automatiquement l'application node lorsque des modifications de fichiers dans le répertoire sont détectées.

Commande d'installation :

```
npm install -g nodemon
```

Commande de lancement :

```
nodemon [votre application]
```

Si problème de droit d'exécution du script :

```
Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
```

IMMERSALIS CONSULTING
IMMERSIVE TECHNOLOGIES FOR EVERYONE

Test de Nodemon :

- Lancement :

```
PS C:\Users\Proprietaire\Node> nodemon .\index.js
[nodemon] 2.0.15
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node .\index.js`
Server running at http://127.0.0.1:8081/
```

- Après modification du code :

```
[nodemon] restarting due to changes...
[nodemon] starting `node .\index.js`
Server running at http://127.0.0.1:8081/
```

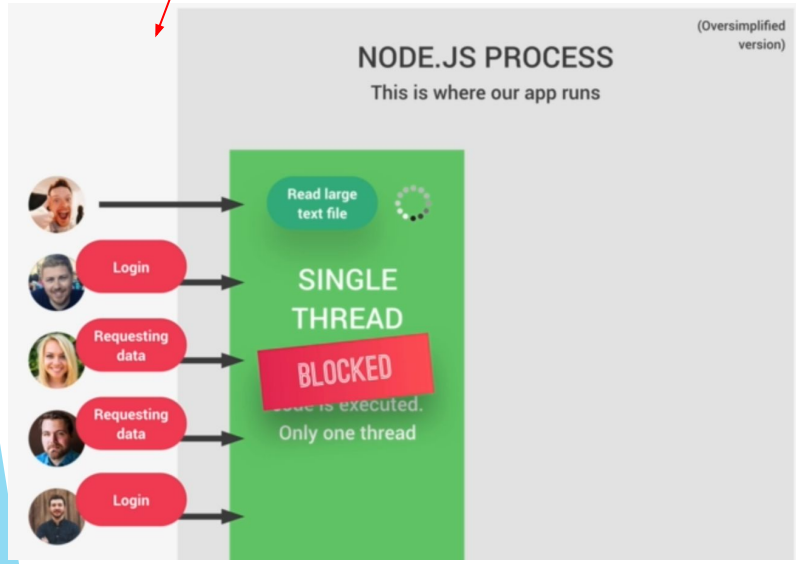
IMMERSALIS CONSULTING
IMMERSIVE TECHNOLOGIES FOR EVERYONE

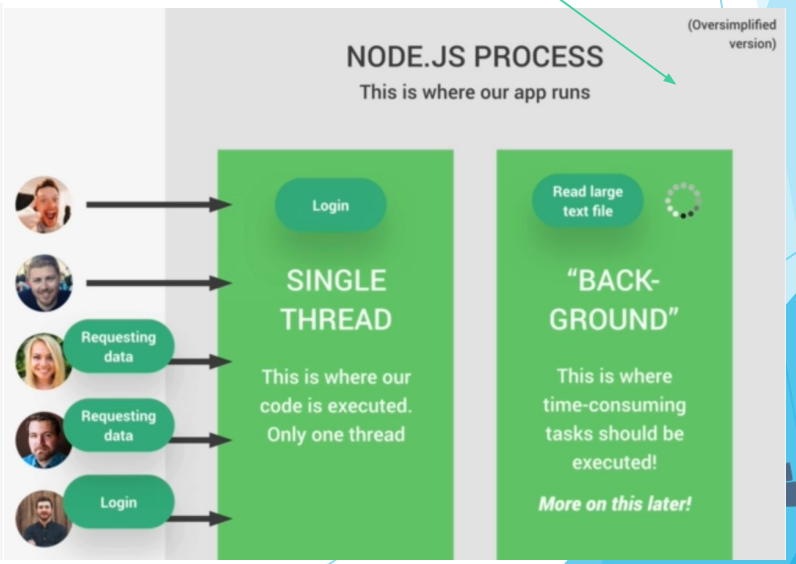
Lecture et écriture de fichier :

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

const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
console.log(textIn);

const textOut = 'Ecriture de fichier';
fs.writeFileSync('./txt/output.txt', textOut);
```

Synchrone / Asynchrone :

```
const textIn = fs.readFileSync('./txt/input.txt', 'utf-8');
console.log(textIn);


fs.readFile('input.txt', 'utf-8', (err, data) => {
    console.log(data);
})
console.log('reading file...');
```





IMMERSALIS CONSULTING
IMMERSIVE TECHNOLOGIES FOR EVERYONE