

Programmation Web côté serveur

Introduction

6-sept-2021

Joseph AZAR

Votre instructeur

- Joseph AZAR
- doctorat en informatique (2017-2020) université bourgogne franche-comté
- **recherche:** Internet des objets, intelligence artificielle, traitement du signal
- **enseignement:** Développement Web et Mobile, base de données
- MCF IUT-BM 2021
- joseph.azar@univ-fcomte.fr
- bat F, etage 2, bureau 147



Niveau de français: intermédiaire+






À propos de ce cours

- développement web
- nous travaillerons principalement sur le **backend**
- nous allons utiliser NodeJS
- environ 32 heures (présentation/TD + TP)
- évaluation: projets + connaissances

Objectif du cours

- créer une application Web
- comprendre comment fonctionne exactement une application de **backend** en considérant:
 - échelle (scale)
 - quantité de données
 - nombre d'utilisateurs
 - performance
 - maintenir une application en croissance

J'ai présumé ce qui suit

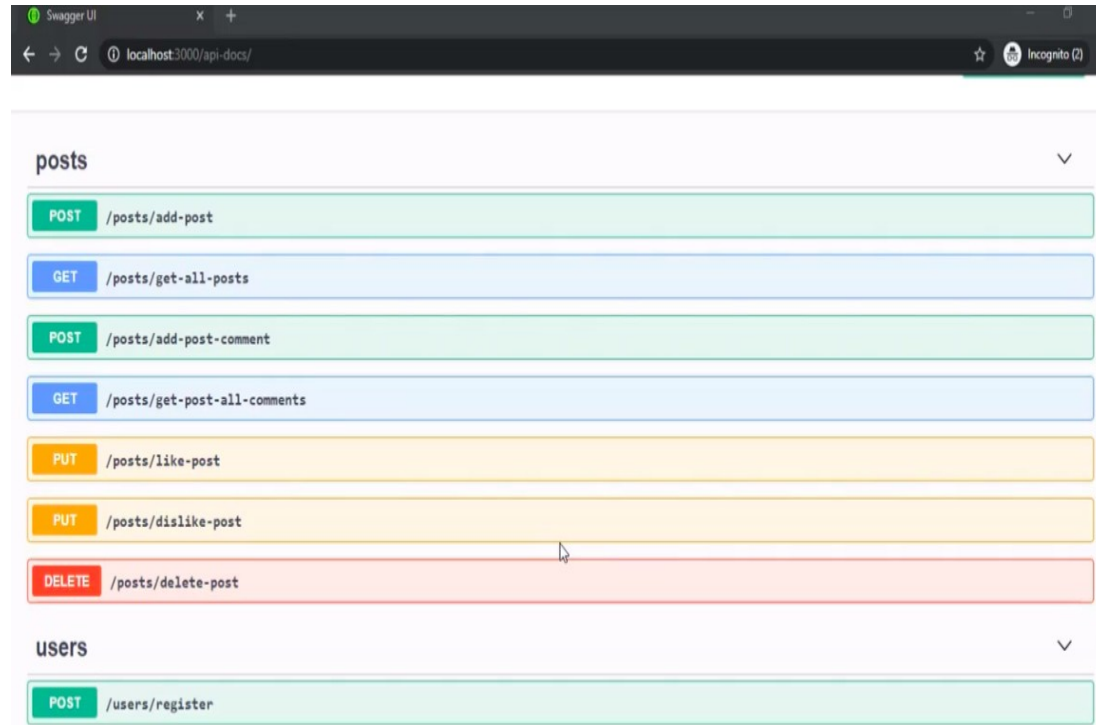
- vous savez déjà comment écrire un logiciel 
- vous connaissez la structure de la page Web et l'apparence avec HTML5 et CSS 
- vous connaissez JavaScript (connaissances de base) 
- vous connaissez l'interactivité côté client JS DOM et événements 
- vous connaissez SQL 

Sujets que nous allons aborder

- systèmes de fichiers
- le Framework Express
- REST API: besoin et style de travail
- REST API: architecture, différentes approches
- travail détaillé pour la route, le contrôleur et les services
- comprendre les méthodes: GET, POST, PUT & DELETE
- comprendre le HTTP Response
- connexion de base de données
- documentation Swagger

En codant une application backend:

- vous avez besoin d'un langage de programmation
- vous avez besoin d'une base de données pour stocker les données
- vous avez besoin d'une technologie pour afficher correctement les données à votre développeur frontend
- votre développeur frontend doit comprendre ce que vous avez écrit
- Représenter votre application de manière correcte:
 - la manière dont le backend doit être appelé
 - dans quel format les données sont acceptées,
 - dans quel format vous donnez les données,
 - vous devez expliquer cela très bien dans une documentation.



URLs et serveurs Web

```
https://server/path/file
```

Habituellement, lorsque vous tapez une URL dans votre navigateur:

- votre ordinateur recherche l'adresse IP du serveur à l'aide de DNS
- votre navigateur se connecte à cette adresse IP et demande le fichier donné
- le logiciel Web Server (E.G. Apache) prend ce fichier à partir du système de fichiers local du serveur, puis renvoie son contenu à vous

Certaines URL spécifient des programmes que le serveur Web doit exécuter, puis vous renvoyer leur sortie à la suite:

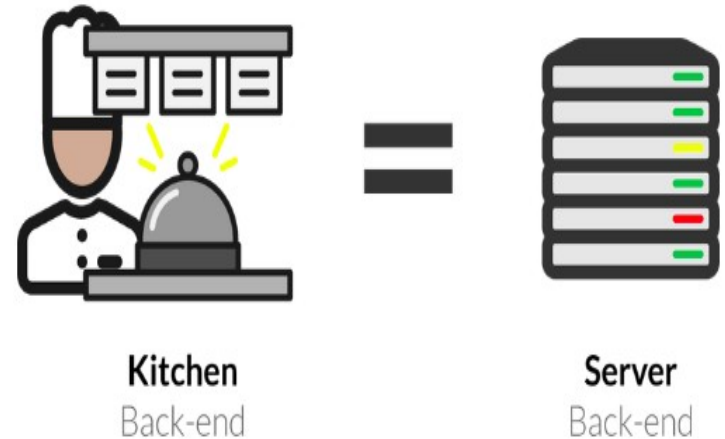
`https://iut-bm.univ-fcomte.fr/test.php`

L'URL ci-dessus indique au serveur iut-bm.univ-fcomte.fr d'exécuter le programme test.php et renvoyer sa sortie.

Pourquoi avons-nous besoin d'un serveur pour gérer les demandes de service Web?

Les serveurs sont des ordinateurs dédiés pour traiter les données de manière efficace et déléguant des demandes envoyées de nombreux clients (souvent à la fois).

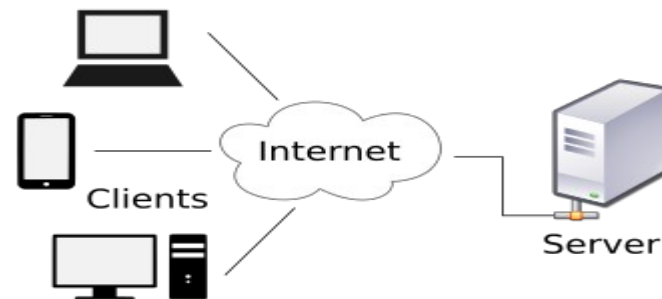
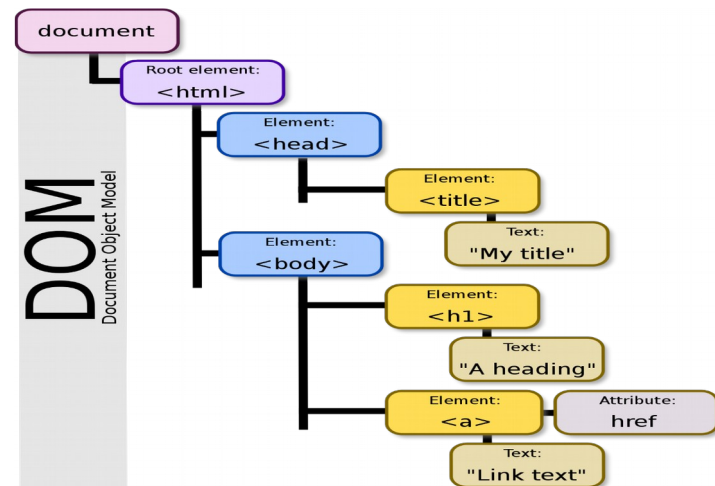
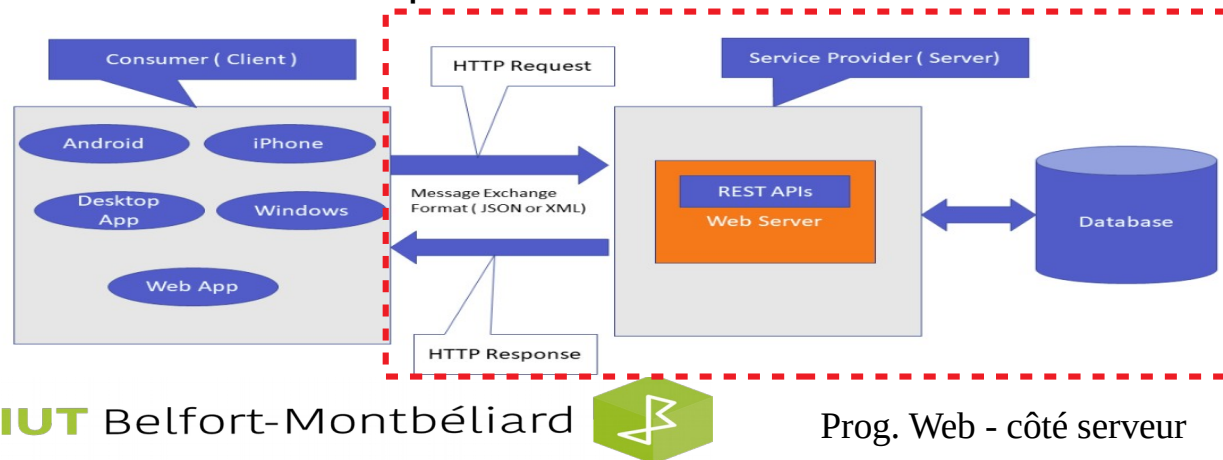
Ces tâches ne sont pas possibles (ou appropriées) dans le navigateur du client.



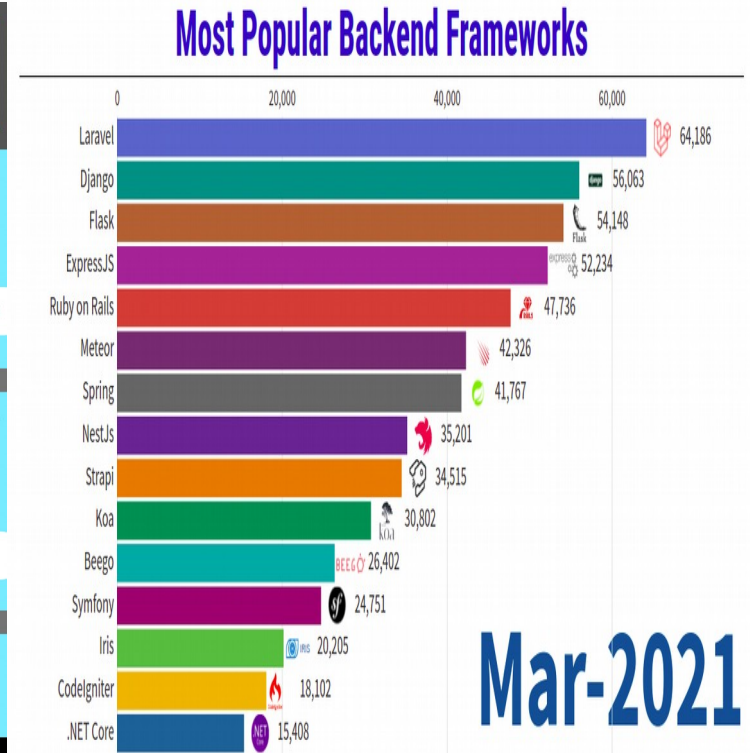
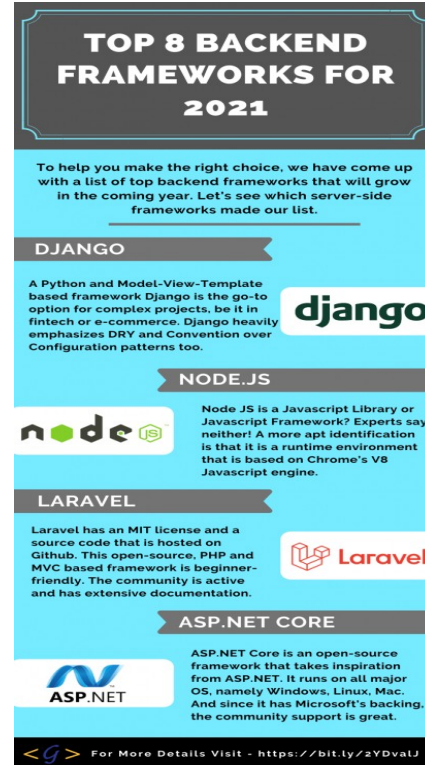
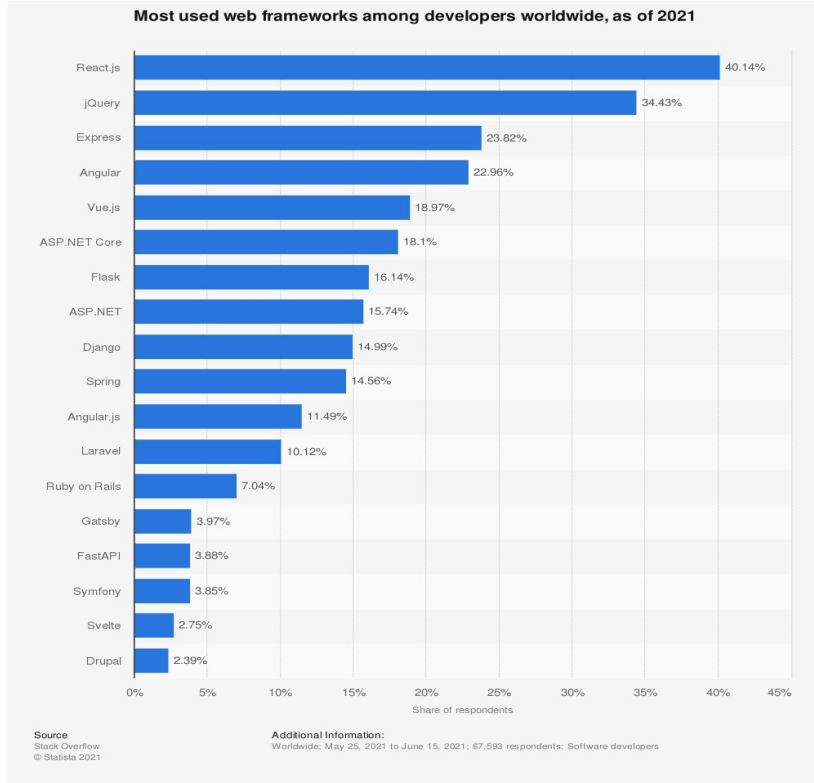
Programmation côté serveur

Auparavant, vous avez écrit du code qui est servi à l'utilisateur et exécuté dans le navigateur. JavaScript exécute le code qui manipule le DOM.

Les programmes côté serveur ne sont jamais donnés aux utilisateurs. Ils sont constamment exécutés sur le serveur, en attente des demandes de la même manière que JS écoute pour des événements. Ils dirigeront ensuite le code et fourniront une réponse au client.



Notre (nouveau) langage de programmation côté serveur: Node JS



Qu'est-ce que NodeJS ?

NodeJS est un environnement d'exécution permettant d'utiliser le JavaScript côté serveur. Grâce à son fonctionnement non bloquant, il permet de concevoir des applications en réseau performantes, telles qu'un serveur web, une API ou un job CRON.

Créé en 2009 par **Ryan Dahl**, ce runtime JavaScript open source et cross-platform avait pour but premier de remédier aux limites de la programmation séquentielle et des serveurs web (tels qu'Apache HTTP Server). **Ces technologies atteignaient effectivement leurs limites lorsqu'elles devaient gérer de très nombreuses connexions simultanées.**

L'objectif principal de Node.js est que l'interaction du système doit être non bloquante ou asynchrone.

Node.js: JS côté serveur

- Node.js utilise le même moteur JavaScript V8 open source que Chrome
- Node.js est un environnement d'exécution pour exécuter des programmes JS en utilisant les mêmes fonctionnalités de langage de base, mais en dehors du navigateur
- Lorsque vous utilisez Node, vous n'avez pas accès aux objets/fonctions du navigateur (par exemple, document, window, addEventListener)
- Au lieu de cela, vous avez accès aux fonctionnalités de gestion des requêtes HTTP, des E/S de fichiers et de l'interaction avec la base de données
- Cette fonctionnalité est essentielle pour créer des API REST!

Node.js, un langage utile et efficace

- Le moteur JavaScript V8 développé par Google, qui permet d'exécuter du code JavaScript à l'intérieur de Google Chrome et, grâce à Node, directement sur le serveur
- Open-Source avec une communauté de développeurs active
- Conçu pour une programmation efficace et asynchrone côté serveur
- Une boucle d'événements, appelée aussi event loop NodeJS, permettant d'exécuter plusieurs opérations simultanées de façon asynchrone et non bloquante en tirant profit des multiples fils d'exécution (multithreading) des noyaux des processeurs modernes

Pourquoi apprendre Node.js?

Node est basé sur JavaScript

Créé en 1995 par Netscape et longtemps cantonné à réaliser de petites animations sur les sites web, JavaScript est aujourd'hui le langage de développement web le plus populaire au monde. Cela peut s'illustrer par plusieurs statistiques dont les suivantes :

- JavaScript est le langage le plus utilisé par les répondants au Stack Overflow Developers Survey de 2019
- C'est également le langage le plus présent dans les contributions GitHub depuis au moins cinq ans
- Enfin, Javascript est le langage utilisé par une écrasante majorité de sites web pour le front-end mais est également de plus en plus populaire pour le back-end

Ces raisons font de JavaScript un langage incontournable et son utilisation à la base de Node.js est un avantage certain car tous les développeurs web en connaissent déjà au moins les bases. Il existe ainsi une grande communauté de développeurs JavaScript animée par une mentalité open source favorisant les échanges de connaissances et la montée en compétences.

Pourquoi apprendre Node.js?

Le marché de l'emploi est en demande

Les avantages apportés par Node.js (accélération du développement, scalabilité, flexibilité, facilité pour créer des équipes fullstack, etc.) en font une solution de plus en plus présente dans les projets des entreprises à travers le monde

À mesure que l'adoption de Node JS en entreprise progresse, la demande en développeurs maîtrisant cette technologie sur le marché du travail augmente également

À titre d'exemple, la plateforme de recrutement spécialisée en IT Hired a annoncé qu'en France, les développeurs Node inscrits sur sa plateforme ont reçu en moyenne 6,5 offres d'entretien par mois en 2019 contre 4.2 pour les développeurs Android, par exemple

En 2020 et toujours dans l'Hexagone, le marché des développeurs NodeJS est même qualifié de "hautement pénurique", faisant des développeurs fullstack JS les développeurs les mieux payés.

The screenshot shows the homepage of the LEMONDE INFORMATIQUE website. The header includes the site's name, a search bar, and navigation links for ACTUALITÉS, DOSSIERS, ÉVÈNEMENTS, SERVICES, LIVRES-BLANCS, and PARTENAIRES. A secondary navigation bar lists various IT topics. The main content area features a large article titled 'Comment maîtriser les nouveaux risques ?' dated 'Mercredi 22 septembre 2021' with an 'Inscrivez-vous' button. Below this is a prominent blue banner for 'Développeur Javascript fullstack, la profession IT qui paye le mieux', attributed to 'Véronique Arène' and dated 'publié le 19 Février 2020'. To the left of the banner are social media icons for Facebook, Twitter, and LinkedIn. To the right, there is a section titled 'SUIVRE TOUTE L'ACTUALITÉ' with a 'Newsletter' sign-up button and text indicating it's for over 50,000 IT professionals.

Pourquoi apprendre Node.js?

La courbe d'apprentissage est abordable

Node JS est une technologie assez facile à prendre en main, notamment grâce aux nombreux frameworks disponibles qui simplifient le développement. S'il présente aussi certaines faiblesses (surtout pour l'exécution côté client), JavaScript reste un langage polyvalent permettant de répondre à de nombreux besoins. Il est également facile à apprendre, surtout depuis l'arrivée de la syntaxe ES6

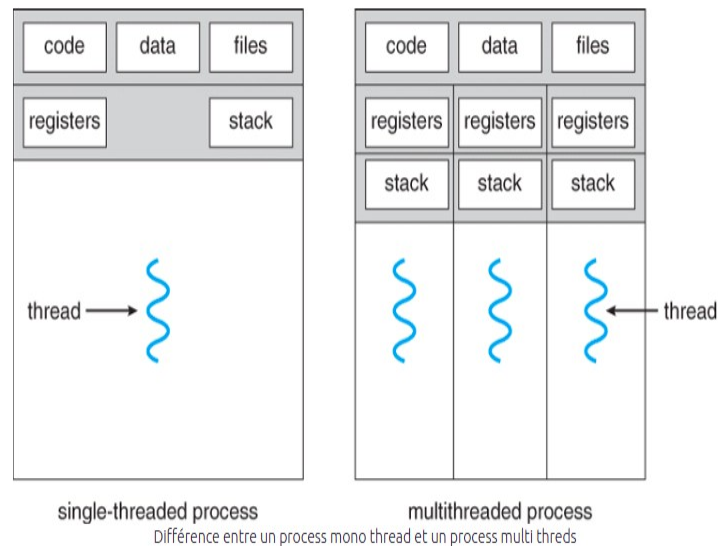
La communauté grandissante de développeurs Node et JavaScript facilite également l'apprentissage en mettant constamment à disposition de nouvelles ressources (cours, tutoriels, vidéos etc.)

L'utilisation de JavaScript comme langage de programmation côté serveur permet également d'abolir les barrières entre front-end et back-end, permettant de monter une équipe fullstack plus facilement

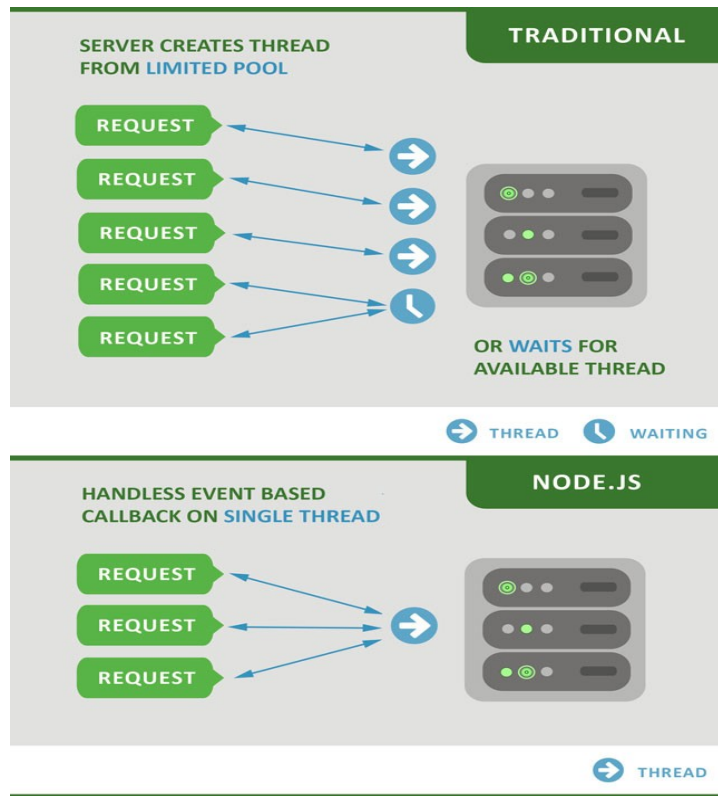
Node est très adapté pour les serveurs web

Node JS est **single-threaded**: il ne s'exécute que sur une seule instance (un thread ou fil d'exécution) du système d'exploitation, à l'inverse d'un serveur fonctionnant en PHP (avec Apache HTTP Server par exemple) qui est, lui, multi-threaded

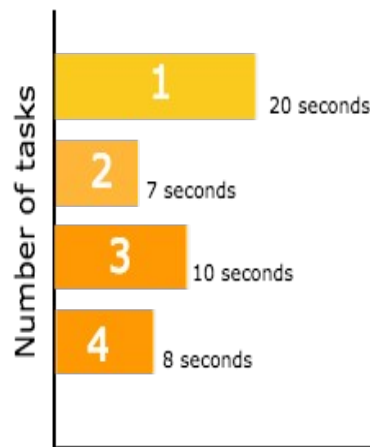
Cette caractéristique de Node lui permet de gérer les requêtes de façon non bloquantes. Là où un serveur PHP mobilise un thread pour réceptionner une requête et attendre qu'elle passe à travers tout le code pour retourner une réponse au client, le serveur Node va gérer chaque requête de manière asynchrone au sein d'un seul et unique thread. La requête est ainsi réceptionnée puis envoyée dans la callback queue où le thread de Node sert les réponses une par une



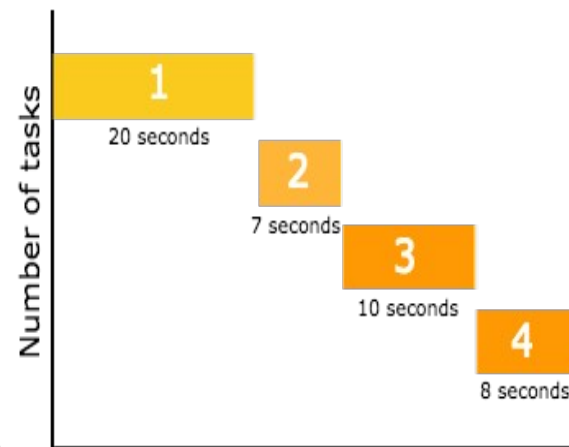
Node est très adapté pour les serveurs web



Node



PHP



Environnement de développement



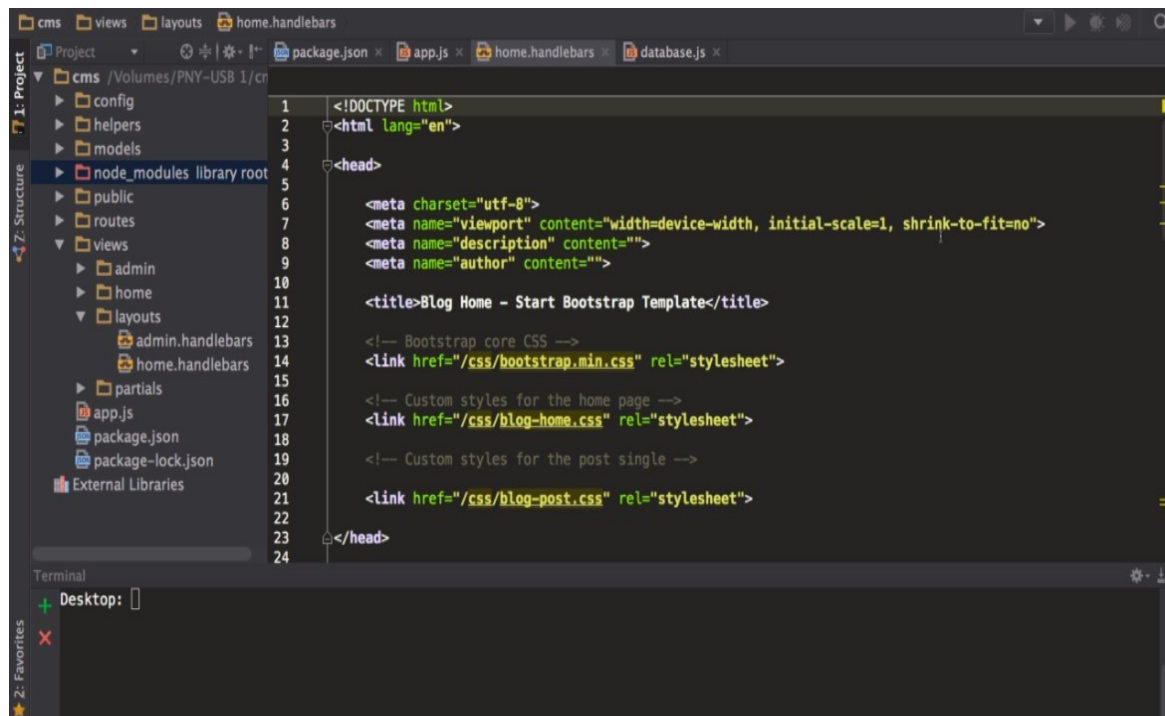
Sublime Text



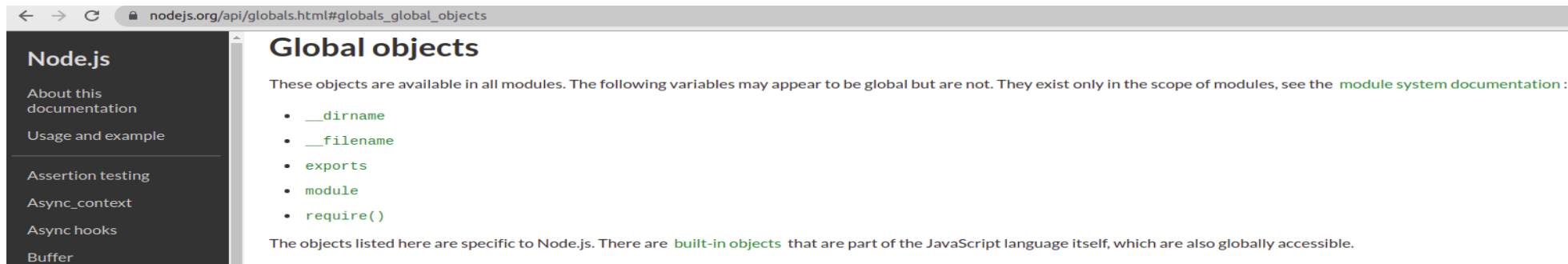
WebStorm



Visual Studio Code



Global object



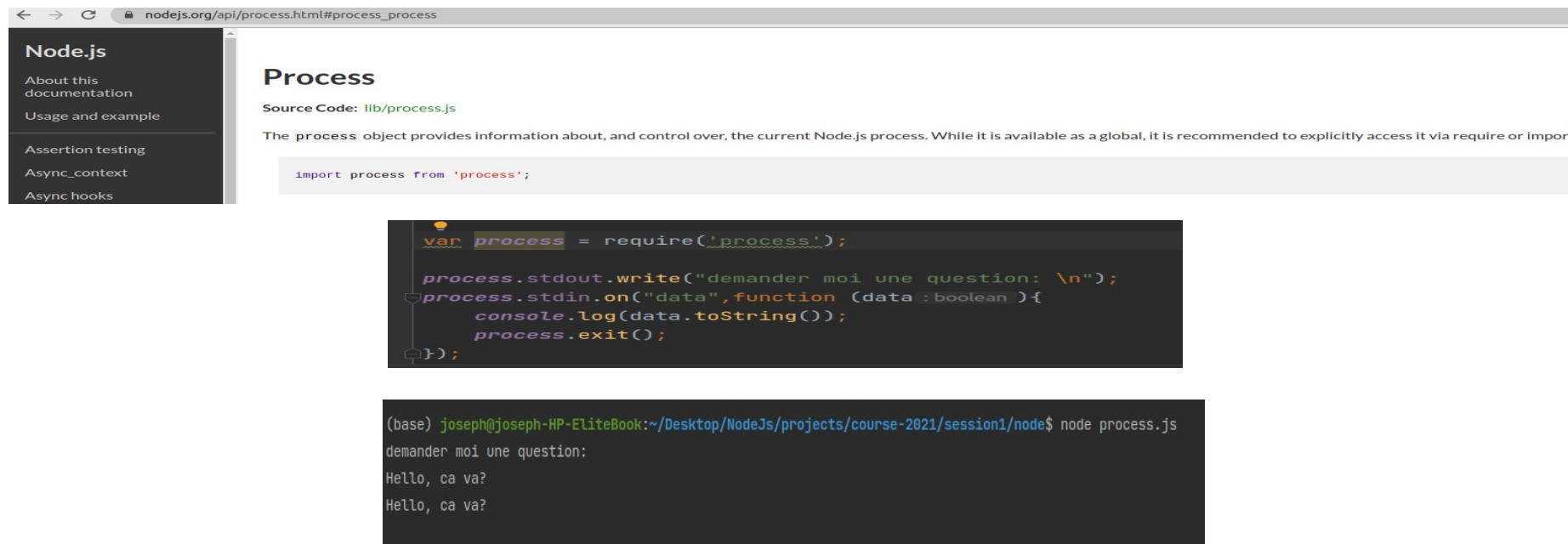
The screenshot shows the Node.js documentation page for global objects. The left sidebar contains a navigation menu with links: Node.js, About this documentation, Usage and example, Assertion testing, Async_context, Async hooks, and Buffer. The main content area is titled "Global objects" and contains the following text: "These objects are available in all modules. The following variables may appear to be global but are not. They exist only in the scope of modules, see the [module system documentation](#):" followed by a bulleted list of variables: `__dirname`, `__filename`, `exports`, `module`, and `require()`. Below the list, it states: "The objects listed here are specific to Node.js. There are [built-in objects](#) that are part of the JavaScript language itself, which are also globally accessible."

les objets globaux dans NodeJS sont de nature globale et ils sont disponibles dans tous les modules. Nous n'avons pas besoin d'inclure ces objets dans notre application, nous pouvons plutôt les utiliser directement. Ces objets sont des modules, des fonctions, des chaînes et l'objet lui-même

```
function imprimerBonjour() {  
    console.log( "Bonjour, tout le monde!");  
}  
  
// Appelez maintenant la fonction ci-dessus après 2 secondes  
setTimeout(imprimerBonjour, timeout: 2000);
```

Process object

L'objet de processus est un objet global dans Node. Il est accessible de n'importe où; L'objet process fournit les flux d'entrée/sortie standard stdin, stdout et stderr (comme en C/C++)



The screenshot displays the Node.js documentation for the `process` module. On the left, a sidebar lists navigation options: "Node.js", "About this documentation", "Usage and example", "Assertion testing", "Async context", and "Async hooks". The main content area is titled "Process" and includes the source code link `lib/process.js`. A descriptive text states: "The `process` object provides information about, and control over, the current Node.js process. While it is available as a global, it is recommended to explicitly access it via `require` or `import`:". Below this, a code block shows the import statement: `import process from 'process';`.

Below the documentation, a code editor displays a JavaScript script:

```
var process = require('process');

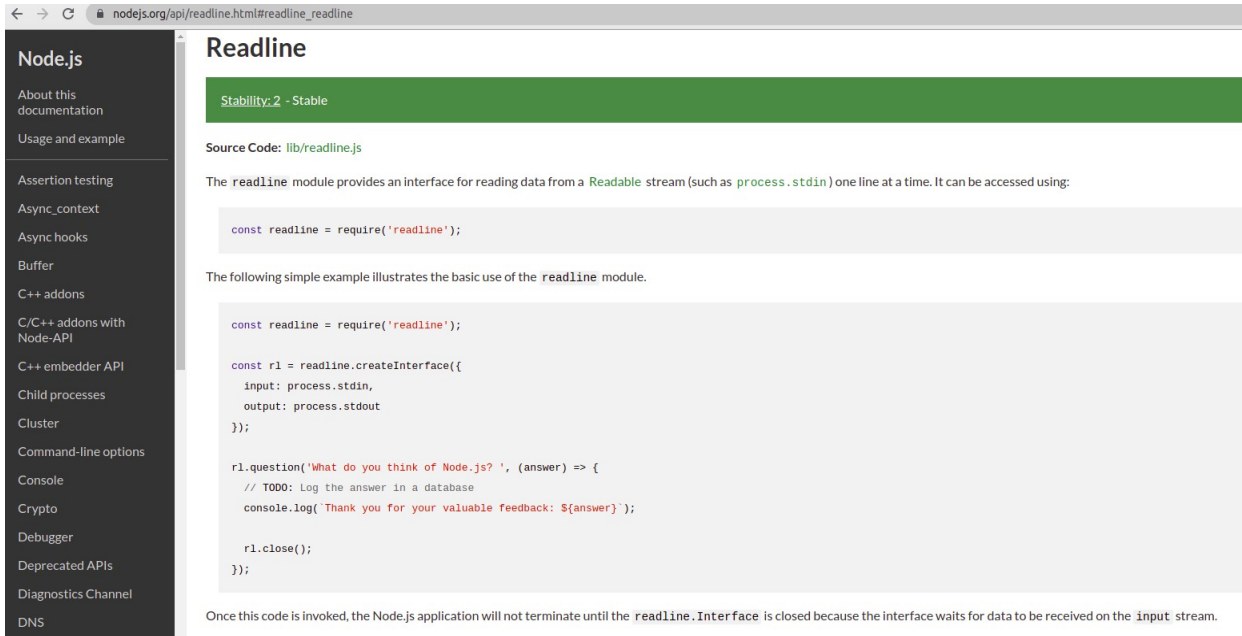
process.stdout.write("demander moi une question: \n");
process.stdin.on("data", function (data : boolean ){
  console.log(data.toString());
  process.exit();
});
```

At the bottom, a terminal window shows the execution of the script:

```
(base) joseph@joseph-WP-EliteBook:~/Desktop/NodeJs/projects/course-2021/session1/node$ node process.js
demander moi une question:
Hello, ca va?
Hello, ca va?
```

Readline

Le module Node.js Readline est essentiellement un wrapper autour de “**process standard output**” et “**process standard input**”; un wrapper donne plus de fonctionnalités et quelque chose de plus facile à comprendre et c'est juste beaucoup plus facile à faire



The screenshot shows the Node.js documentation for the `readline` module. The page title is "Readline" and it indicates "Stability: 2 - Stable". The source code is listed as `lib/readline.js`. The text explains that the `readline` module provides an interface for reading data from a `Readable` stream (such as `process.stdin`) one line at a time. It can be accessed using:

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

The following simple example illustrates the basic use of the `readline` module.

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

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});

rl.question('What do you think of Node.js? ', (answer) => {
  // TODO: Log the answer in a database
  console.log('Thank you for your valuable feedback: ${answer}');

  rl.close();
});
```

Once this code is invoked, the Node.js application will not terminate until the `readline.Interface` is closed because the interface waits for data to be received on the `input` stream.

```
const readline = require('readline');
var RL = readline.createInterface(process.stdin, process.stdout);

RL.question('quel est ton nom? ', (nom :string) => {
  RL.setPrompt(`${nom}, quel age as tu? |`);
  RL.prompt();
  RL.on('line', (age :string) => {
    console.log(`tu as ${age} ans`);
    RL.close();
  });
});
```


Custom events

Nous pouvons créer des événements personnalisés tels que des notifications pour les utilisateurs lorsqu'ils se connectent à un tableau de bord ou déclencher un e-mail lorsqu'un utilisateur s'inscrit à une application

Donc disons que si on veut émettre un événement, on crée un événement qui affiche un message par exemple à chaque fois que l'utilisateur se connecte

```
const events = require('events');
let emitter = new events.EventEmitter();
emitter.on( event: 'login', listener: (message)=>{
    console.log(`Message : ${message}, Vous vous êtes connecté au système`);
});
emitter.emit( event: 'login', args: 'Joseph Azar');
```


Module

Le module dans Node.js est une fonctionnalité simple ou complexe organisée en un ou plusieurs fichiers JavaScript qui peuvent être réutilisés dans l'application Node.js

Chaque module de Node.js a son propre contexte, il ne peut donc pas interférer avec d'autres modules ni polluer la portée globale. De plus, chaque module peut être placé dans un fichier .js distinct dans un dossier distinct

Myfirstmodule.js

```
exports.myDateTime = function () {  
    return Date();  
};
```

moduleExample.js

```
var http = require('http');  
var dt = require('./myfirstmodule');  
  
http.createServer(function (req, res) {  
    res.writeHead(200, {'Content-Type': 'text/html'});  
    res.write("The date and time are currently: " + dt.myDateTime());  
    res.end();  
}).listen(8080);
```

Child processes

Le module **child_process** nous permet d'**accéder aux fonctionnalités du système d'exploitation** en exécutant n'importe quelle commande système à l'intérieur d'un processus enfant

Nous pouvons contrôler ce flux d'entrée de processus enfant et écouter son flux de sortie. Nous pouvons également contrôler les arguments à transmettre à la commande du système d'exploitation, et nous pouvons faire ce que nous voulons avec la sortie de cette commande

