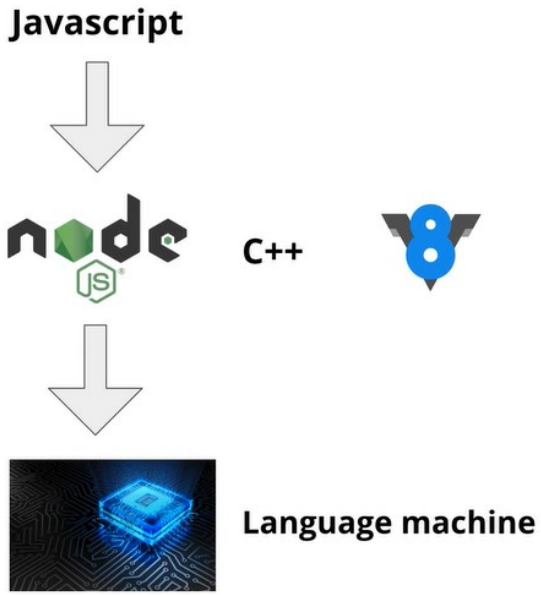
Présenté par : Sana EL ATCHIA

NodeJS qu’est-ce que c’est ?

- Ce n'est pas un langage car il utilise JavaScript.
- Ce n'est pas non plus un Framework car il ne fait pas qu'offrir des fonctionnalités en utilisant un langage.
- **C’est un environnement d'exécution open-source et multiplateforme pour le développement d'applications web côté serveur.**
- Il possède également une architecture événementielle capable d'effectuer des entrées/sorties asynchrones.
- Il s'agit d'un environnement serveur pour le JavaScript, construit en utilisant le moteur JavaScript de Google V8.
- Il a été créé en 2009 par Ryan Dahl.
- Il est écrit en C/C++ et en JavaScript.
- V8 : interpréteur JavaScript.



Installation de Node

L'installation de NodeJS et de NPM est simple en utilisant le paquet d'installation disponible sur le site officiel de NodeJS.

- Téléchargez le programme d'installation depuis le site web de NodeJS.

  https://nodejs.org/en/download/

- Exécutez le programme d'installation.
- Suivez les étapes du programme d'installation.
- Acceptez l'accord de licence et cliquez sur le bouton suivant.
- Redémarrez votre système/machine.

Utiliser l'interpréteur Node

- Vérifier l’installation de Node
  - Ouvrir le terminal et taper la commande suivante : **node –v** ou **node –version**

- Exécuter le code d'un fichier
  - Pour exécuter un fichier contenant du JavaScript avec Node, il vous suffit d'aller dans le dossier contenant votre fichier puis de taper : **node monScript.js**
  - L'interpréteur de Node va lire le contenu du fichier et exécuter toutes les instructions.

Qu’est ce que c’est un npm?

- Node Package Manager (NPM) est un outil en ligne de commande qui installe, met à jour ou désinstalle des paquets Node.js dans une application.

- Le npm couvre en fait trois choses différentes :

  - **Le gestionnaire de paquets officiel de Node.js**. Il est installé par défaut lorsque vous installez Node.js. Il est aujourd'hui plus globalement le gestionnaire de paquets le plus utilisé pour le JavaScript.

  - C'est le plus important dépôt de paquets (packages registry) au monde et contient à ce jours plus de 900000 paquets JavaScript !

  - **C'est un CLI**, c'est-à-dire une interface en ligne de commande, permettant de gérer les paquets dans un projet : les installer, les désinstaller, les mettre à jour etc. Il permet également de publier des paquets sur le dépôt npm.

- Site officiel : https://www.npmjs.com

Utilisation de npm

- NPM est inclus dans l'installation de Node.js
- Vérifier l'installation de NPM en écrivant la commande suivante : **npm -v**
- Accéder à l'aide de NPM en écrivant la commande suivante : **npm help**

Utilisation de npm - Package.json

- Le package.json d'un projet ressemble à quelque chose comme ci-dessous :

- Pour créer le fichier **package.json** dans un projet, il suffit de taper la commande suivante :

  npm init

Ce fichier permet de :

- ❖ Lister les paquets utilisés par votre projet qui sont appelés ses dépendances.
- ❖ Spécifier précisément quelles versions des dépendances peuvent être installées.
- ❖ Permettre aux développeurs d'installer toutes les dépendances de votre projet en une commande

Package.json – composition (1/2)

- Les éléments les plus importants du package.json sont le **nom** et la **version**. Ils sont en fait obligatoires, et votre paquet ne s'installera pas sans eux.
- Le nom et la version forment ensemble un identifiant qui est supposé être complètement unique.
- Il est composé aussi de :
  - **Repository** : ce dépôt indique l'endroit où se trouve votre code. Les développeurs peuvent accéder à votre application et y contribuer.

```
"repository": {
    "type": "git",
    "url": "https://github.com/npm/npm.git"
}
```

  - **Scripts** : NPM fournit de nombreux scripts utiles comme **npm install**, **npm start**, **npm stop** etc.

```
{
    "scripts": {
        "start": "node server.js"
    }
}
```

Package.json – composition(2/2)

- **"start": "node server.js“** : s'il y a un fichier **server.js** à la racine de votre package alors npm utilisera par défaut la commande start pour node server.js.

- **Les dépendances** sont spécifiées dans un objet simple qui associe un nom de paquet à une gamme de versions.

  - Le nom de la version doit correspondre exactement à la version.
  - Si vous souhaitez installer la dernière version d'un fichier, il vous suffit de mettre latest à la place du nom de la version.
  - Le tilde (~) est utilisé pour indiquer "Approximativement équivalent à la version".

Utilisation de npm

NPM effectue l'opération en deux modes : global et local.

- **Global** : NPM effectue des opérations qui affectent toutes les applications Node.js sur l'ordinateur.
- **Local** : NPM effectue des opérations pour le répertoire local particulier qui affecte une application dans ce répertoire uniquement.

Utilisation de npm - Installer un paquet en local

- La commande suivante permet d'installer un paquet dans le dossier local de votre projet Node.js :

  npm install nom-du-paquet ou **npm i nom-du-paquet**

- Exemple :

  npm install express : cette commande permet d'installer ExpressJS dans votre dossier.

- Tout module installé à l'aide de NPM est installé dans le dossier node_modules et est ajouté comme une dépendance dans le fichier package.json

Utilisation de npm - Installer un paquet en global

- NPM installe les paquets globaux dans le dossier **\AppData\Roaming\npm\node_modules**
- La commande suivante permet d'installer un paquet de manière globale :

  npm install -g nom-du-paquet ou **npm i -g nom-du-paquet**

- Exemple :

  npm install – g express : cette commande permet d'installer ExpressJS de manière globale.

Utilisation de npm - Mettre à jour un paquet

- Pour mettre à jour le paquet installé localement dans votre projet Node.js, naviguez dans l'invite de commande ou dans la fenêtre du terminal jusqu'au dossier du projet et écrivez la commande de mise à jour suivante.

npm (-g) update

- Exemple :

npm update express : La commande suivante mets à jour le module ExpressJS existant à la dernière version.

Utilisation de npm - Désinstaller un paquet

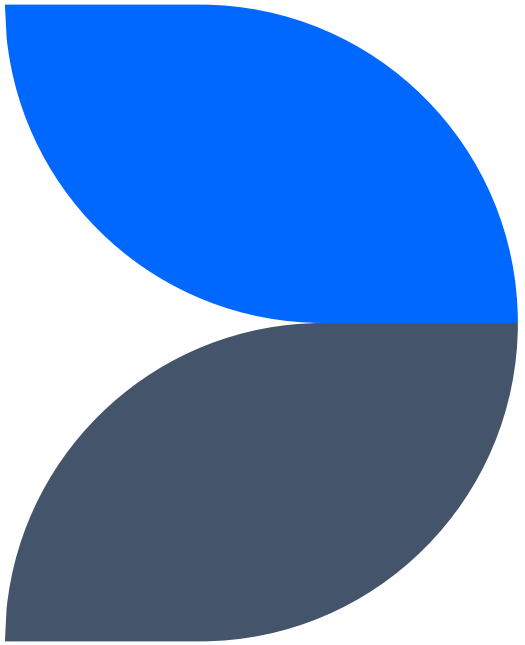
- La commande suivante permet de désinstaller un paquet de manière locale (/globale) :

  npm uninstall (-g) nom-du-paquet

- Exemple :

  npm uninstall – g express : cette commande permet désinstaller ExpressJS de manière globale.

Serveur Node



Node.js – serveur web

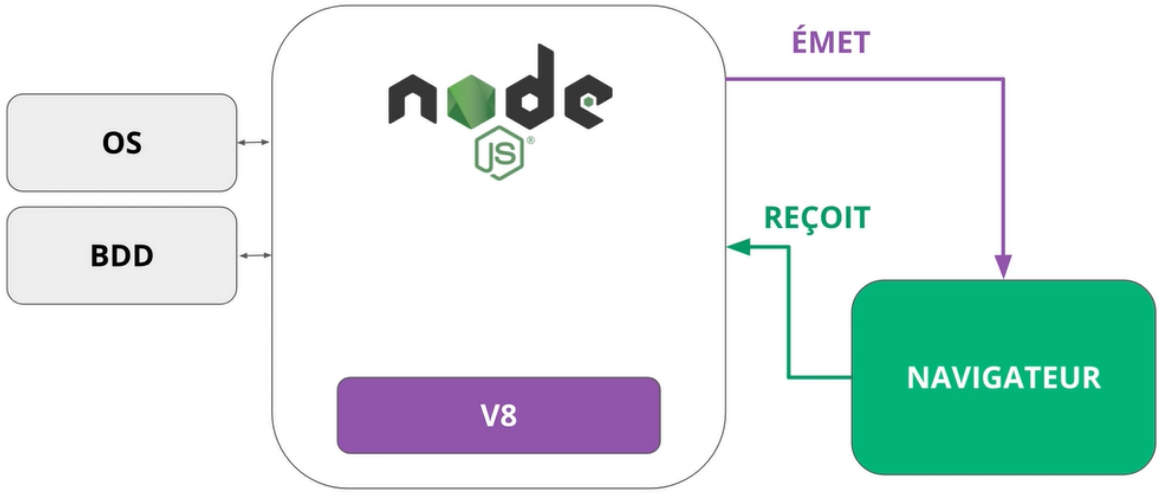
- Pour accéder aux pages web d'une application web, vous avez besoin d'un serveur web.

- Node.js offre la possibilité de créer votre propre serveur web qui traitera les requêtes HTTP de manière asynchrone.

Node.js – serveur web

- Un serveur Web est au minimum un programme qui peut parser des URLs et qui peut interagir avec des clients en utilisant le protocole Http.

- Fonctionnalités nécessaires pour un serveur web :
  - Réutilisabilité du code et modularité
  - Interactions avec des fichiers
  - Interaction avec des bases de données
  - Possibilité de recevoir des requêtes et d'envoyer des réponses avec les protocoles http et https



Qu’est ce que c’est un module?

- Le JavaScript n'avait aucun moyen natif d'organiser du code.

- Un système de module permet de séparer votre code en briques qui peuvent être utilisées dans d'autres modules en les important.

- Vous pouvez choisir ce que vous exportez, c'est-à-dire ce que vous souhaitez rendre disponible lorsque votre module est importé dans un autre module.

- Fonctionnement des modules Node :

  - Chaque fichier est traité comme étant un module séparé.
  - Chaque fichier, donc chaque module, peut être importé dans un autre fichier
  - Pour importer un module il suffit d'utiliser **require** en spécifiant le chemin relatif vers le fichier :

    ```
    const monModule = require('./monModule.js’);
    ```
  - Pour exporter des données, il suffit d’utiliser l’objet **module.exports**.

Créer un serveur HTTP avec Node.js

- Node.js permet de créer facilement un serveur web simple qui traite les demandes entrantes de manière asynchrone.

- L'exemple suivant est un serveur web Node.js simple contenu dans le fichier server.js.

```
const http = require('http');
const server = http.createServer( (request, response) => {
  // Fonction de Callback pour lire la requête et écrire la réponse
});
server.listen(8080);
```

Exemple de base – créer un serveur

- Créer un fichier javascript server.js avec le code suivant :

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

// création d'un serveur node.js
const server = http.createServer(function (req, res) {
  res.setHeader("Content-Type", "text/plain");
  res.end("Bienvenue à la grande classe");
});

// port sur lequel on écoute notre serveur
server.listen(3000, "127.0.0.1", function () {
  console.log("serveur lancé ...");
});
```

Lancer le serveur Node.js (exemple)

- Exécuter le fichier server.js dans node à l'aide de la commande :

  node server

- Ouvrir le navigateur et entrer l'url http://127.0.0.1:3000/.

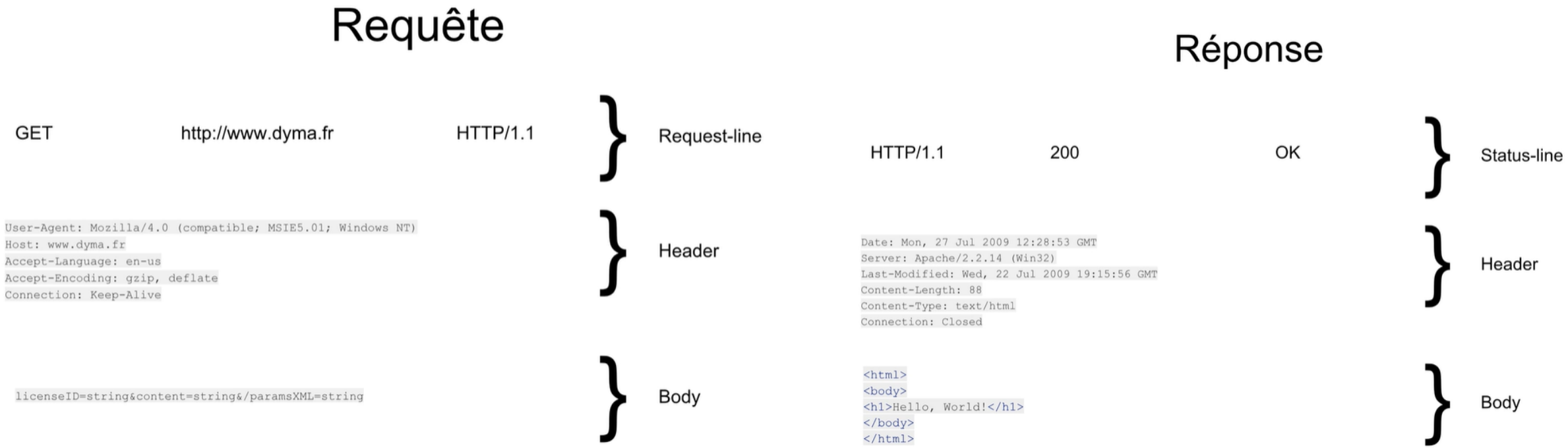
- Le navigateur affiche le message « Bienvenue à la grande classe » à l'écran.

Requêtes HTTP

La méthode http.createServer() , permet de créer le serveur, comprend des paramètres de **demande** et de **réponse** qui sont fournis par Node.js.

Request : est utilisé pour obtenir des informations sur la requête HTTP en cours, par exemple l'url, l'en-tête de la requête et les données.

Response : est utilisé pour envoyer une réponse à une demande HTTP en cours



Exemples de requêtes HTTP

```
const http = require("http");
// création du serveur web
const server = http.createServer(function (req, res) {
  // vérifier url au niveau de la requete courante
  if (req.url == "/") {
    //Définir l'entete de la réponse
    res.writeHead(200, { "Content-Type": "text/html" });

    // Définir le conetenu de la réponse
    res.write("<h1>la page d'acceuil</h1>");
    res.end();
  } else if (req.url == "/etudiant") {
    res.writeHead(200, { "Content-Type": "text/html" });
    res.write("<h1>Afficher la liste des stagiaires</h1>");
    res.end();
  } else if (req.url == "/saisiEtudiant") {
    res.writeHead(200, { "Content-Type": "text/plain" });
    res.write("Ajouter un stagiaire");
    res.end();
  } else if (req.url == "/data") {
    res.writeHead(200, { "Content-Type": "application/json" });
    res.write(JSON.stringify({ message: "une réponse JSON" }));
    res.end();
  } else {
    res.end("Erreur !");
  }
});
server.listen(3000);
console.log("le serveur web Node.js est lancé ...");
```

Envoyer une réponse dans un fichier HTML

- Node.js inclut le module fs pour accéder au système de fichiers physiques.
- Le module fs est responsable de toutes les opérations d'E/S de fichiers asynchrones ou synchrones.

Envoyer une réponse dans un fichier HTML

- Node.js offre la possibilité d'interagir avec le système de fichiers (espace de stockage) qui supporte le serveur.

- Le module nommé fs (pour File System) doit être inclus au début du fichier server.js au même titre que le module http comme ceci:

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

- Il est possible aux scripts de Node.js d'accéder aux fichiers du serveur, c'est à dire, ils peuvent mener les opérations suivantes:

  - ✓ Lecture des fichiers
  - ✓ Modifier le contenu des fichiers
  - ✓ Créer de nouveaux fichiers
  - ✓ Renommer les fichiers
  - ✓ Supprimer les fichiers

Système de fichiers Node.js – Lecture asynchrone

- La méthode **fs.readFile()** permet de lire un fichier physique de manière asynchrone.

  fs.readFile(filename ,[option] ,callback)

  - filename : Chemin complet et nom du fichier sous forme de chaîne.
  - options : Le paramètre options peut être un objet ou une chaîne qui peut inclure l'encodage et l'indicateur.

    =>L'encodage par défaut est utf8 et l'indicateur par défaut est "r".

  - callback : Une fonction avec deux paramètres err et fd. Elle sera appelée lorsque l'opération readFile sera terminée.

Système de fichiers Node.js – Lecture asynchrone Exemple

- L'exemple suivant permet la lecture asynchrone du fichier acceuil.text existant :

```
var fs = require("fs");
fs.readFile("acceuil.txt", "utf8", function (err, data) {
  if (!err) {
    console.log(data);
  } else {
    console.log(err);
  }
});
```

- L'exemple ci-dessus lit acceuil.txt de manière asynchrone et exécute la fonction de rappel lorsque l'opération de lecture est terminée.

- Cette opération de lecture déclenche une erreur ou se termine avec succès. Le paramètre err contient des informations sur l'erreur, le cas échéant. Le paramètre data contient le contenu du fichier spécifié

Système de fichiers Node.js – Lecture synchrone Exemple

- Utiliser la méthode **fs.readFileSync()** pour lire le fichier de manière synchrone, comme indiqué ci-dessous

```
var fs = require("fs");
var data = fs.readFileSync("acceuil.txt", "utf8");
console.log(data);
```

Système de fichiers Node.js – Ecrire dans un fichier

- Utiliser la méthode **fs.writeFile()** pour écrire des données dans un fichier.
- Si le fichier existe déjà, il écrase le contenu existant, sinon il crée un nouveau fichier et y écrit les données.

fs.writeFile(filename, data, [option], callback)

- ✓ filename : chemin complet et nom du fichier sous forme de chaîne.
- ✓ data : Le contenu à écrire dans un fichier.
- ✓ options : Le paramètre options peut être un objet ou une chaîne de caractères qui peut inclure l'encodage...

  =>L'encodage par défaut est utf8 et l'indicateur par défaut est "r".
- ✓ callback : Une fonction avec deux paramètres err et fd. Elle sera appelée lorsque l'opération d'écriture sera terminée.

Système de fichiers Node.js – Ecrire dans un fichier Exemple

- L'exemple suivant permet de créer un nouveau fichier appelé ecrire.txt et y écrit « Bienvenue à la grande classe" de manière asynchrone.

```
var fs = require("fs");
// ajouter un fichier
fs.writeFile(
  "ecrire.txt",
  "Bienvenue à la grandeclasse",
  "utf8",
  function (err) {
    if (!err) {
      console.log("ok");
    } else {
      console.log(err);
    }
  }
);
```

- De la même manière, utilisez la méthode fs.appendFile() pour ajouter le contenu à un fichier existant.

```
var fs = require("fs");
fs.appendFile("ecrire.txt", " CFA", "utf8", function (err) {
  if (!err) {
    console.log("ok");
  } else {
    console.log(err);
  }
});
```