Sommaire

Qu'est-ce que Node.JS

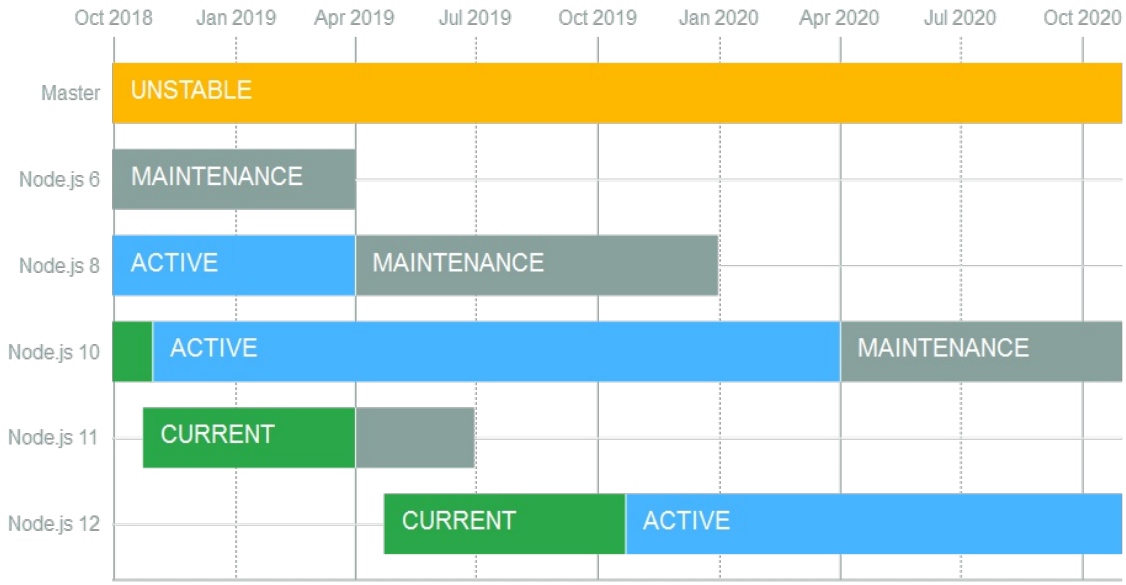
Qu'est-ce que Node.JS

- Javascript côté server
- Moteur V8 de Webkit
- Open Source
- Multiplateforme (ou presque)
- Mono thread

- Asynchrone
- I/O non bloqués

Qu'est-ce que Node.js

- Versions standards et LTS



Qu'est-ce que Node.js

Release	Status	Codename	Initial Release	Active LTS Start	Maintenance LTS Start
6.x	**Maintenance LTS**	[Boron][]	2016-04-26	2016-10-18	2018-04-30
8.x	**Active LTS**	[Carbon][]	2017-05-30	2017-10-31	April 2019
10.x	**Current Release**	Dubnium	2018-04-24	October 2018	April 2020
11.x	**Pending**		2018-10-23		

Qu'est-ce que Node.js

- Deux types de version :
  - LTS : Support éttendu : Ce sont les versions les plus stables
  - Les autres : versions dites "beta" (test de nouvelles fonctionnalités, …)

Qu'est-ce que Node.JS

- serveur http

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

http.createServer((req, res) => {
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.end('Hello World!');
}).listen(8080);
```

Utilisation

- scripts bash
- compilations diverses (babel, …)
- applications multiplateformes desktop (electron, NW)
- applications mobiles (react-native, ionic, native-script)
- serveurs web (express, hapi, koa)
- serverless (chromeless, …)
- …

Gestionnaire de paquets

Gestionnaire de paquets

- NPM
  - officiel
  - npmjs.org

```
npm install -S hapi
```

- Yarn
  - facebook
  - plus rapide

```
yarn add hapi
```

Gestionnaire de paquets

- NPX
  - Permet de lancer des binaires installés via npm
  - Peu importe s'ils sont au sein du projet ou global

```
./node_modules/.bin/eslint
/usr/bin/eslint

npx eslint
```

Package.json

- coeur de l'application
- paquets utilisés
- commandes de lancement
- version Node.JS requise

```
{
```

```
  "private": true,
  "name": "Test APP",
  "description": "This is an example package.json",
  "version": "0.0.0-this-does-not-matter",
  "license": "MIT",
  "scripts": {
    "dev": "npx poi",
  },
  "devDependencies": {
    "less": "2.7.2",
    "less-loader": "4.0.5",
  },
  "dependencies": {
    "lodash": "4.17.4",
    "moment": "2.18.1",
  }
}
```

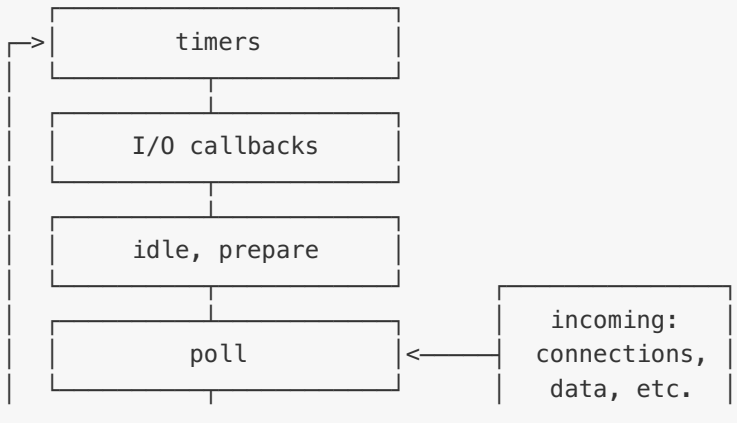
Package.json

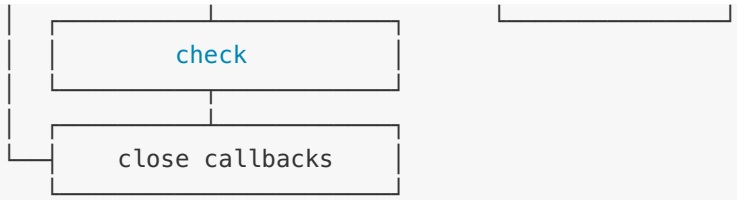
- **scripts** : scripts à lancer via `npm run`
- **devDependencies** : Dépendances servant uniquement pour la phase de développement
- **dependencies** : dépendances du projet
- **private** : Si à `true`, le projet ne sera jamais publié sur npm si vous lancez `npm publish` par mégarde

---

EventLoop

EventLoop





EventLoop (détails)

- **timers** : lance les callbacks programmés par `setTimeout` et `setIntervale`
- **I/O callbacks** : lance tous les autres callbacks
- **idle, prepare** : utilisé par le coeur de Node.JS
- **poll** : Créer de nouveaux événements d'entrée/sortie. Le cas échéant, peut bloquer l'exécution du script.
- **check** : invoque les callbacks de `setImmediate`
- **close callbacks** : `socket.on('close')`, ...

EventLoop (détails)

- Permet l'asynchrone
- Possibilité de forcer l'exécution au prochain passage avec `process.nextTick()`
- Fonctions avec callback pas toujours asynchrones selon ce qu'elles font
- Promesses toujours Asynchrones

EventLoop (exemples)

```
const funcSync = callback => {

    const fields = {
        foo : 'foo'
    };
    callback(fields);
};

console.log('a');
```

```
funcSync(field => console.log); // funcSync(field => console.log(field));
console.log('fin');
```

```
a
{foo: "foo"}
fin
```

EventLoop (exemples)

```
const funcSync = callback => {

    const fields = { foo : 'foo'};
    process.nextTick(() => {

        callback(fields);
    });
};

console.log('a');
funcSync(field => console.log);
console.log('fin');
```

```
a
fin
{foo: "foo"}
```

EventLoop (exceptions)

- **Attention** : tous les callbacks ne sont pas asynchrones de base :

```
const a = [1, 2, 3, 4];

a.forEach((value) => {

    console.log(value);
```

```
});

console.log('after forEach');
```

- Les fonctions des types de base (String, Object, Array, …) sont toujours synchrones.

Les événements

Les événements

- Utilisés au travers de la classe `EventEmitter`
  `javascript const EventEmitter = require('events').EventEmitter; const myEventNamespace = new EventEmitter();`
- Callbacks effectués à des moments précis
- Appels via `event.emit('eventName', ...params)`
- Interceptés via `event.on('eventName', callbackFunc)`

Les promesses

.then & .catch

```
myPromiseFunction()
   .then((val) => console.log(val))
   .catch(TypeError, err => console.error)
   .catch(ReferenceError, err => console.error)
   .catch(err => console.error('Error non gérée', err));
```

async & await

```
try {

  const val = await myPromiseFunction();
  console.log(val);
}
```

```
catch(error){

  if(error instanceof TypeError){
    console.error('Type Error');
  }

  if(error instanceof ReferenceError){
    console.error('Reference error');
  }

  console.error(error);
}
```

Les promesses

- Asynchrones
- Capturent les erreurs
- Transformation fonction callback en promesses via `util.promisify`

```
const Util = require('util');
const Fs   = require('fs');

const ReadFileAsync = Util.promisify(Fs.readFile);

(async () => {

  try {
    const text = await ReadFileAsync(filePath, { encoding: 'utf8' });
    console.log(text);
  }
  catch(error){

    console.error(error);
  }
})();
```

Les promesses

- Norme A+ : Utilisation de libs compatibles
- Manipulation des Arrays en asynchrone :
  - `Promise.map`
  - `Promise.each`

- Fonctions étendues compatibles A+ au sein de la library `bluebird`

Websockets

socket.io

- Websocket : connexion "permanente"
- Communication client/serveur via les événements
- Aucun timeout
- L'événement appelé peut avoir un callback
  `javascript socket.emit('myeventWithCallback', (result) => { });`
- Gestion de Rooms
- Permet l'envoi d'events en broadcast

Mise en production

- Serveurs Web :
  - PM2 : Permet de lancer le serveur comme un service et de le relancer automatiquement
  - Docker : via un système d'orchestration (Rancher, …)
- Module :
  - NPM et Yarn
- Scripts :
  - Services functions (AWS lambda, …)

Plan de travail : Outils

- Utilisation de NVM avec la dernière version LTS de Node.JS en date (8.X)
- Utilisation de Webstorm (licences gratuites étudiants)
- Repository Git pour le suivi du projet
- TD et TP séparés en 2 parties :
  - Apprentissage via nodeschool.io
  - Mise en pratique

Organisation

- TD : Introduction
- TP1 : Analyse, Debug et Eslint
- TP2 : GIT et NPM
- TP3 : Hapi
- TP4 : Hapi : Swagger et Joi
- TP5 : Objection et migrations
- TP6 : Envoi d'email
- TP7 : RabbitMQ et microservice d'email

Objectifs

- Comprendre Node.JS
- Comprendre les websockets
- Utiliser Hapi.JS
- Faire une API REST
- Interragir avec PostgreSQL
- Approche microService