301 - Introduction

node.js

WIK-NJS301

Durée estimée : 6h

Intervenant : Jeremy Trufier <jeremy@wikodit.fr>

ynov

wikodit

WIK-NJS

Programme nodeJS

301 – Introduction
302 – Scripting et CLI
303 – Express.js
304 – MVC Frameworks
305 – Tests unitaires

1XX – 1er année (pas de notion d'algorithmie)
2XX – 2e année (notions d'algorithmie succintes)
3XX – 3e année (rappels et pratique, niveau moyen d'algorithmie)
4XX – 4e année (concepts avancés, niveau avancé d'algorithmie)
5XX – 5e année (approfondissement experts)

Au préalable

Qui l'utilise ?

Source: nodejs wiki

Téléchargement

- https://nodejs.org
- Version "**Current**"

Préparation

Tout d'abord, créez un dossier pour les cours nodejs

- Ouvrir un terminal ou PowerShell
- Naviguez vers ce dossier (avec la commande `cd dossier1/sousdossier/...`
- Créez un dossier appelé njs-301 (avec la commande `mkdir njs-301`)
- Naviguez vers ce dossier (`cd njs-301`)

Tout ce qui se passera dans ce cours
devra se passer dans ce dossier "njs-301" dans le terminal

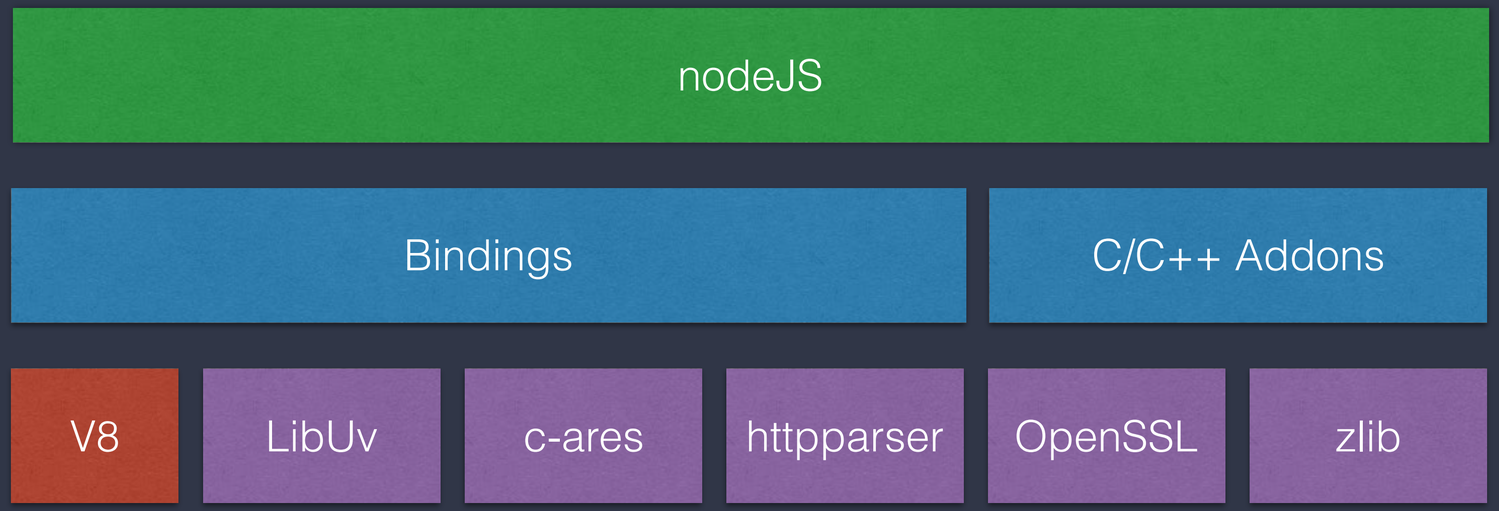
Installation Windows

Utilisateur OSX & Linux : Rien de plus à faire :)

Utilisateurs Windows :

- Ouvrir PowerShell avec clic droit + ouvrir en tant qu'administrateur
  - executer la commande suivante : npm install -g windows-build-tools
- Fermer PowerShell, et réouvrir en utilisateur standard
  - executer : npm install sqlite3
- Si ça ne fonctionne pas et qu'une erreur rouge concernant "CL.exe" apparait
  - Si vous avez Visual Studio, l'ouvrir, créér un projet Visual C++ pour télécharger les outils Visual C++ 2015
  - Rééssayez
  - Sinon, ouvrir "Invite de commande développeur" et reessayez
- Sinon... essayer ce qui est en note de ce slide
- Sinon... utilisez une VM

La mécanique



nodeJS s'appuie un maximum sur des fonctions systèmes écrites en C/C++

Source : http://abdelraoof.com/blog/2015/10/19/introduction-to-nodejs/

Les avantages

- Asynchrone
- Événementiel
- Javascript
- Communauté
- Gros volumes de connections

Le désavantage : peut-être très intensif au niveau CPU pour des tâches gourmandes

CLI

Command Line Interface

Commande `node`

Terminal ou PowerShell

Evaluation de script
Mode interactif
Code javascript
Output
Return

Pour quitter
Création d'un fichier
Execution d'un fichier JS

```
$ node -e "console.log('hello')"
$ node
> console.log('hello')
hello
undefined
> 1+3
4
> .exit
$ echo "console.log('Hello World')" > ./hello.js
$ node ./hello.js
Hello World
$
```

Un serveur Web

```
const http = require('http')

http.createServer((req, res) => {
  res.end('Bonjour à tous !')
}).listen(8080)
```

JavaScript ES6
Notions & Rappels

Les bases

constante

variable et objet

fonction nommée

```
const HELLO = 'Bonjour'

var currentUser = {
  firstname: 'Jean',
  lastname: 'Dubois'
}

function welcome(person){
  return `${HELLO} ${person.firstname} ${person.lastname}`
}

console.log(welcome(currentUser))
```

Convention utilisée : http://standardjs.com/rules.html#javascript-standard-style

Les types

- boolean
- number
- string
- undefined
- null
- object
- symbol (ES6)

Que remarquez-vous ?

```
typeof 'une chaine'
typeof true
typeof undefined
typeof { name: 'Jean' }
typeof [0, 1, 2, 3]
typeof null
typeof function(){}
```

Les conditions

Simple égalité

```
var a = '2'

if (a == 2) {
  console.log('a est égal à 2')
} else {
  console.log('a n\'est pas égal à 2')
}
```

Strict égalité : le type est pris en compte

```
var a = '2'

if (a === 2) {
  console.log('a est le chiffre 2')
} else {
  console.log('a n\'est pas le chiffre 2')
}
```

Condition ternaire

```
var a = '2'
console.log(a == '2' ? 'a égal 2' : 'a n\'est pas égal à 2')
```

Les conditions

Opérateurs de comparaison

<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égalité faible
===	Égalité stricte
!=	Inégalité faible
!==	Inégalité stricte

Opérateurs logiques

&&	ET logique
||	OU logique
!	NON logique

```
var a = '2'

if (typeof a != undefined && a != null) {
  if (a == 1) {
    console.log('a est 1')
  } else if (a >= 2 || a <= 5)) {
    console.log('a est entre 2 et 5 inclus')
  } else {
    console.log('a n'est pas entre 1 et 5')
  }
} else {
  console.log('a est null ou non défini')
}
```

Les fonctions

```
// Fonction nommée
console.log(sum(7, 4)) // => 11
function sum(x, y) {
  return x + y
}
```

```
// Fonction anonyme
console.log(diff(7, 4)) // => ERREUR
var diff = function (x, y) {
  return x - y
}
console.log(diff(7, 4)) // => 3
```

```
// Fonction anonyme avec flèche
var times = (x, y) => { return x * y }
```

Les objets et tableaux

```
// Les objets
var user = {
  firstname: 'Jean'
  lastname: 'Dubois'
}

var user2 = user
user2.firstname = 'Marc'

// Les tableaux
const LANGUAGES = [ 'fr', 'en', 'es' ]
user.language = language[0]

// Le résultat ??
console.log(user)
```

Les itérations (for)

```
const FRUITS = [ 'apple', 'orange', 'strawberry', 'blueberry' ]

for (var i = 0, l = FRUITS.length; i < l ; i++) {
  console.log(1, FRUITS[i])
}
```

```
for (var i = FRUITS.length; i--; ) {
  console.log(2, FRUITS[i])
}
```

```
for (var i = 0, l = FRUITS.length; i < l ; i++) {
  if (FRUITS[i] === 'apple') { continue }
  console.log(3, FRUITS[i])
  if (FRUITS[i] === 'strawberry') { break }
}
```

Les itérations (while)

```
var found = false, i = 0
while ( !found ) {
  if (FRUITS[i] === 'strawberry') {
    found = true
  }
  console.log(4, FRUITS[i])
  i++
}
```

La portée des variables

```
var test = 'a'

var func = function () {
  var test = 'b'

  if ( test === 'b' ){
    var test = 'c'
    console.log(1, test)
  }

  console.log(2, test)
}

func()
console.log(3, test)
```

```
let test2 = 'a'

var func2 = function () {
  let test2 = 'b'

  if ( test2 === 'b' ) {
    let test2 = 'c'
    console.log(1, test2)
  }

  console.log(2, test2)
}

func2()
console.log(3, test2)
```

TD : Le jeu du plus ou moins

Au lancement, le script choisit un nombre de 0 à 999.

Le but du jeu est de trouver ce nombre.

À chaque mauvaise réponse, le script indique simplement si le nombre est supérieur ou inférieur.

Memo & Tips

```
// Initialisation
const rl = require('readline').createInterface({
  input: process.stdin, output: process.stdout
})

// Pose une question à l'utilisateur
rl.question('Question ?', (answer) => {
  process.stdout.write("Tu as répondu : " + answer + "\n")
})

// Retourne un entier aléatoire entre 0 et 10
Math.floor(Math.random() * 10)

// RegExp qui retourne true si input contient 1 à 3 chiffres
/^\d{1,3}$/.test(input)

// Quitte le programme
process.exit()
```

Langage prototypé

```
// foo hérite du prototype de Object
foo = new Object()
foo.bar = 'toto'

// le code ci dessus est l'équivalent de
foo = { bar: 'toto' }
```

- Tout type non primitif est un objet
- Objets ont des prototypes
- Objets héritent les propriétés de leur prototype
- L'opérateur `new` créé une instance d'un objet en appelant la méthode `constructor` du prototype

```
// Création d'un prototype avec un constructeur
const Person = function (firstname, lastname) {
  this.firstname = firstname
  this.lastname = lastname
}

// Ajout d'une méthode au prototype de Person
Person.prototype.name = function() {
  return this.firstname + ' ' + this.lastname
}

// Utilisation de notre classe
p = new Person('Jean', 'Bon')
p.name() // => "Jean Bon"
```

Les classes ES6

```
class Employee {
  constructor (firstname, lastname) {
    this.firstname = firstname
    this.lastname = lastname
  }

  static createFromName (name) {
    const p = new Employee()
    p.name = name
    return p
  }

  get name () {
    return `${this.firstname} ${this.lastname}`
  }

  set name (name) {
    [ this.firstname, this.lastname ] = name.split(' ')
  }

  sayHello () {
    return `Bonjour ${this.firstname} !`
  }
}

let jeanEmployee = new Employee('Jean', 'Bon')
let emilieEmployee = Employee.createFromName('Emilie Bond')
```

Les classes ES6 : Héritage

```
class Boss extends Employee {
  get name () {
    return `Mr. ${super.name}`
  }

  stressOut (person) {
    return `Plus vite que ça ${person.firstname} !!`
  }
}

let michelBoss = new Boss('Michel', 'Dubois')
michelBoss.stressOut(jeanEmployee) // "Plus vite que ça Jean !!"
michelBoss.name // "Mr. Michel Dubois"
```

Gestionnaire de package

Ça sert à quoi ?

- Installation et réinstallation aisée des modules/library
- Gère les métadonnées d'un projet ou package
- Mise à jour des modules aisée
- Partage des dépendances avec l'équipe
- Verrouillage des versions des modules pour un projet donné

NPM
Node Package Manager

- Le plus grand nombre de package
- Compatible GIT
- Autour d'un fichier `package.json`
- Gestion des versions majeure, mineure, etc...
- Gestion de scripts
- Publication d'un nouveau package

Le package.json

<package.json>

```
{
  "name": "njs-301",
  "version": "0.0.1",
  "description": "Introduction à node.js",
  "main": "app.js",
  "scripts": {
    "start": "node app.js"
  },
  "author": "Jeremy Trufier <jeremy@wikodit.fr>",
  "license": "MIT",
  "dependencies": {
    "lodash": "^4.16.2"
  }
}
```

Gestionnaire de package

Nouveau dossier
Navigation dans le dossier
Initialisation interactive du projet
Installation de lodash

Lancement de node
Appel d'une fonction de lodash
Echec: `_` n'existe pas
On charge lodash

La méthode fonctionne

Terminal

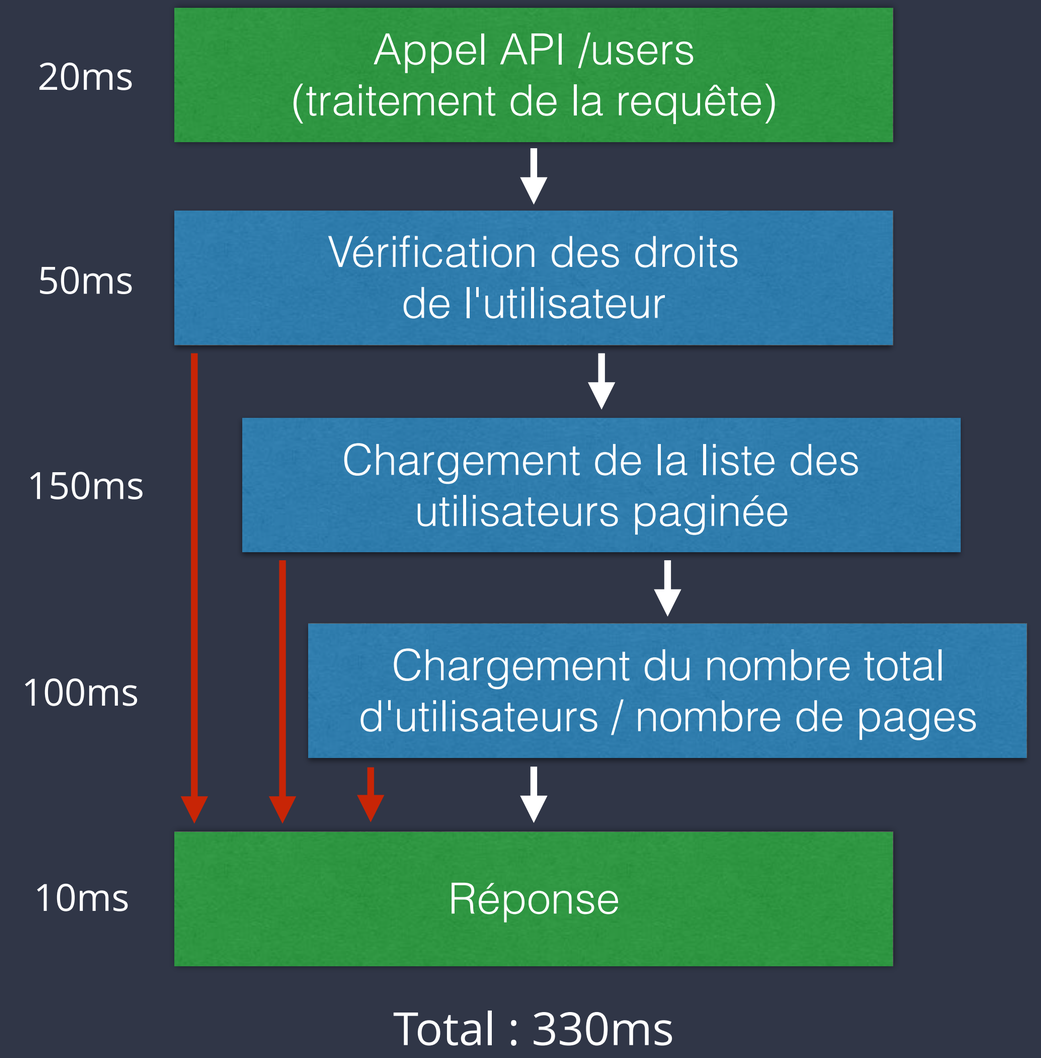
```
$ mkdir njs-301
$ cd njs-301
$ npm init
$ npm install --save lodash

$ node
> _.camelCase('Foo Bar')
TypeError: Cannot read property 'camelCase' of undefined
> var _ = require('lodash')
> _.camelCase('Foo Bar')
'fooBar'
```

Asynchrone et événements

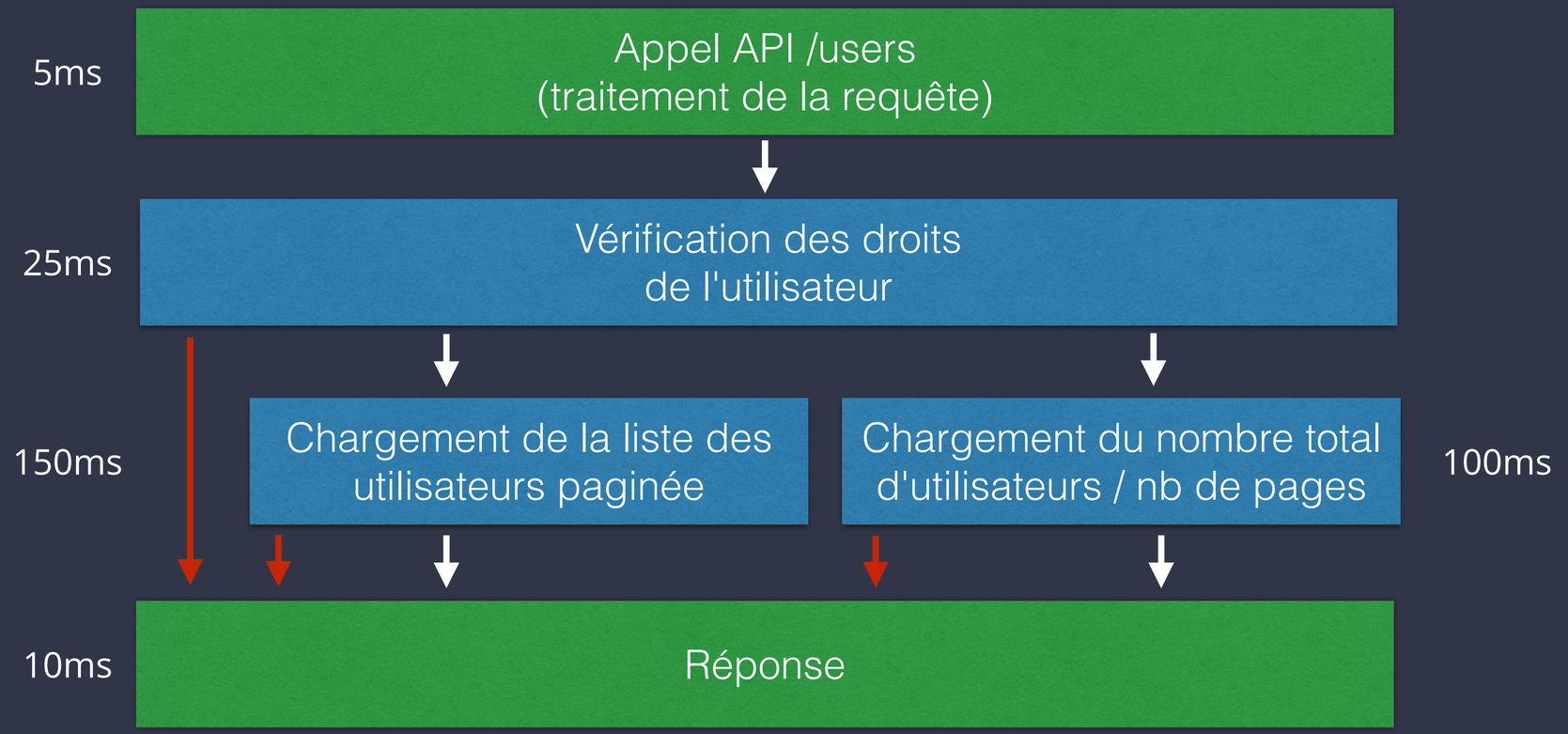
Le code synchrone

- Procédural
- En pause lors des I/O (bloquant)
- Dans certains langage : création d'un nouveau thread
- Temps de traitement total = temps de chaque partie additionnée



Le code asynchrone

- Parallélisation
- Optimisation
- Thread unique
- Event-Loop



Total : 190ms

nodeJS est asynchrone !

Synchrone

```
const fs = require('fs')
let filepath = '/etc/passwd'

try {
  fs.accessSync(filepath, fs.constants.R_OK)
  console.info(`I can read ${filepath}`)
} catch (e) {
  console.error(`No access to ${filepath}`)
}

console.log('This is written last!')
```

Asynchrone

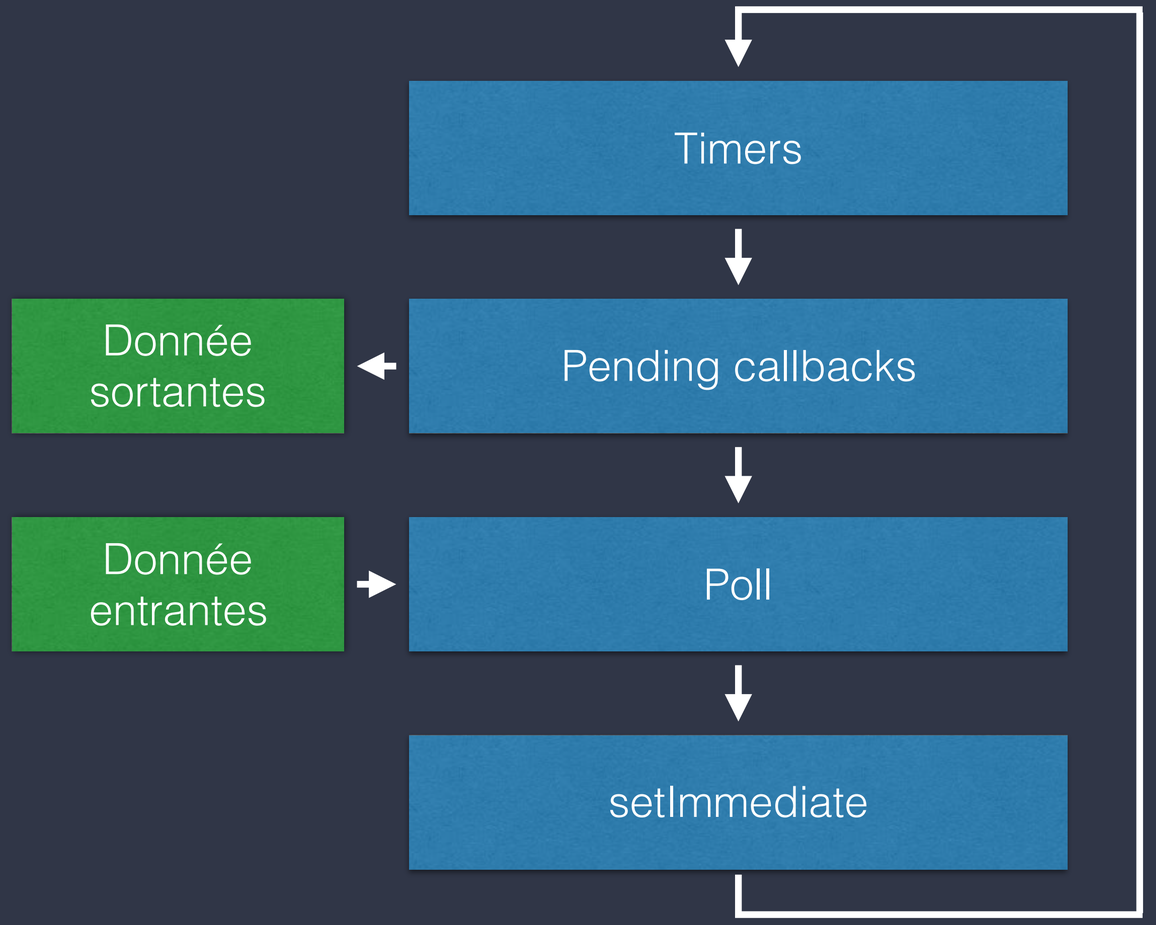
```
const fs = require('fs')
let filepath = '/etc/passwd'

fs.access(filepath, fs.constants.R_OK, (err) => {
  if (err) {
    console.error(`No access to ${filepath}`)
  } else {
    console.info(`I can read ${filepath}`)
  }
})

console.log('This is written first!')
```

Event loop

- Event Driven Programming
- Grande stabilité
- Peut gérer de grosses charges
- Jamais en pause ou en attente



Utiliser les évènements

register-event.js

```
process.stdin.setEncoding('utf8');

process.stdin.on('readable', () => {
  var chunk = process.stdin.read()
  if (chunk !== null) {
    process.stdout.write(`stdin datas: ${chunk}`)
  }
})

process.stdin.on('end', () => {
  process.stdout.write('end')
})

console.log('Program started')
```

Terminal

```
$ echo 'Hello world' | node register-events.js
Program started
stdin datas: Hello world
end
```

Rendre une fonction Asynchrone

```
function divide (a, b, callback) {
  process.nextTick(() => {
    try {
      if (b == 0) { throw new Error('Can not divide by 0') }

      let result = a / b

      if(isNaN(result)) { throw new Error('Can\'t divide those') }

      callback(null, result)
    } catch (e) {
      callback(e)
    }
  })
}

let showResults = (err, result) => console.log(err, result)

divide(10, 4, showResults)
divide(5, 0, showResults)
console.log('Written first')
```

Emettre des évènements

```
const EventEmitter = require('events')

class Divider extends EventEmitter {
  constructor (divisor) {
    super()
    this.divisor = divisor
  }

  run (numerator) {
    process.nextTick(() => {
      this.emit('start')

      try {
        let result = numerator / this.divisor
        if(isNaN(result)) {
          throw new Error('Can\'t divide those')
        }

        this.emit('result', result)
      } catch (e) {
        this.emit('error', e)
      }

      this.emit('end')
    })
  }
}
```

```
let divideByTwo = new Divider(2)

divideByTwo.run(3)

divideByTwo.on('start', () => {
  console.log('started')
})
divideByTwo.on('result', (r) => {
  console.log(`result is ${r}`)
})
divideByTwo.on('end', () => {
  console.log('ended')
})

console.log('still printed first')

// still printed first
// started
// result is 1.5
// ended
```

Les promesses

Le problème des callbacks

- Utile pour de simples opérations
- Continuous Passing Style (CPS)
  - Séquence de fonctions compliquée
  - Parallèlisation encore pire
  - Gestion d'erreur catastrophique

CPS en série

```
currentUser.checkAccess('users', function(err, access) {
  if (err) { return console.error(err) }

  Users.find().skip(offset).limit(limit).exec(function(err, users){
    if (err) { return console.error(err) }

    response.data = users

    Users.count({}, function(err, count){
      if (err) { return console.error(err) }

      response.meta.count = count
      console.log(response)
    })
  })
}
```

CPS en parallèle

```
let stuffToBeFinished = 0

stuffToBeFinished += !!Users.find().skip(offset).limit(limit).exec(function(err, users){
  if (err) { return console.error(err) }

  response.data = users
  finishedStuff()
})

stuffToBeFinished += !!Users.count({}, function(err, count){
  if (err) { return console.error(err) }

  response.meta.count = count
  finishedStuff()
})

function finishedStuff () {
  if (--stuffToBeFinished) { return }
  console.log(response)
}
```

Les solutions

Des librairies :

- async.js
- async
- chainsaw
- each
- flow-js
- ...

ou

Les Promises

Les Promises ES6 en séquentiel

```
currentUser.checkAccess('users').then((access) => {
  return Users.find().skip(offset).limit(limit)
}).then((users) => {
  response.data = users
  return Users.count({})
}).then((count) => {
  response.meta.count = count
}).then(() => {
  console.log(response)
}).catch((err) => {
  console.error(err)
})
```

Et donc ?

- Chaînage
- Retours uniformes
- Organisation du code aisée
- Code en série ou en parallèle facile
- Asynchrone
- Erreurs regroupées

Les Promises ES6 en parallèle

```
currentUser.checkAccess('users').then((access) => {
  promises = [
    Users.find().skip(offset).limit(limit),
    Users.count({})
  ]
  return Promise.all(promises)
}).then((results) => {
  response.data = results[0]
  response.meta.count = results[1]
  console.log(response)
}).catch((err) => {
  console.error(err)
})
```

Les Promises ES6 : Création

```
class User {
  constructor (pseudo, permissions) {
    this.pseudo = pseudo
    this.permissions = permissions
  }

  checkAccess (resource) {
    return new Promise((resolve, reject) => {
      if (~this.permissions.indexOf(resource)) {
        resolve(true)
      } else {
        reject(new Error('Forbidden'))
      }
    })
  }
}

let asterix = new User('asterix', ['users', 'comments'])
asterix.checkAccess('comments').then((access) => {
  console.log(access)
}).catch((err) => {
  console.error(err)
})
```

TD : Livre d'or

Afficher une simple page avec :

- *Une liste de commentaires*
  - *Pseudo*
  - *Message*
  - *Date*

- *Un formulaire*
  - *Pseudo*
  - *Message*
  - *Bouton Envoyer*

On s'attarde au côté fonctionnel pas de style

Serveur Web

```
const http = require('http')
const qs = require('querystring')

// Simple serveur WEB
http.createServer((req, res) => {
  console.log(req.method) // GET ou POST
  console.log(req.url) // '/livre-d-or'

  // SOIT une page HTML
  res.writeHead(200, {'Content-Type': 'text/html'})

  // SOIT une redirection vers  "/livre-d-or"
  res.writeHead(302, {'Location': '/livre-d-or'})
  // Écriture d'une page HTML
  res.write(`
    <html>
      <body>HTML Page</body>
    </html>
  `)
  res.end() // Toutes les données on été envoyée
}).listen(8081)
```

TD : Livre d'or (aide)

Récupération données POST du formulaire

```
// Il suffit d'écouter deux évènement sur l'objet req :
// - 'data' => Renvoi les données du formulaire au fur à mesure qu'elles arrivent
// - 'end' => Appelé une fois que toutes les données sont arrivées
req.on('data', (data) => { console.log(data) })

// qs permet de parser des données :
let params = qs.parse('foo=3&bar=5')
console.log(params.foo) // => 3
```

Stockage des données

```
const db = require('sqlite')

// Ouvre une session SQLite stockée en mémoire vive (Promise)
db.open(':memory:')

// Simple SQL (Promise)
db.run("CREATE TABLE IF NOT EXISTS comments (pseudo, comment, createdAt)")

// Tableau de tous résultats (Promise)
db.all("SELECT * FROM comments WHERE pseudo = ?", 'toto')

// Premier résultat (Promise)
db.get("SELECT * FROM comments WHERE pseudo = ?", 'toto')
```

Package NPM requis : sqlite

TD : Livre d'or (aide)

Exemple de HTML en sortie :

```
<html>
    <body>
      <h1>Livre d'or</h1>
      <div class="comments">
            <p class="comment">
              Jean le 10/05/2024 : Oui ça marche bien
            </p>
            <p class="comment">
              Emilie le 09/05/2024 : Bonjour, c'est super !
            </p>
      </div>
      <form method="post">
        <label for="pseudo">
          Pseudo :
          <input type="text" id="pseudo" name="pseudo" />
        </label>
        <br />
        <label for="comment">
          Commentaire :
          <textarea id="comment" name="comment"></textarea
        </label>
        <br />
        <input type="submit" value="Envoyer" />
      </form>
    </body>
  </html>
```

Félicitations !!

Cours WIK-NJS-301 burned :)