

Adel Latibi
adel.latibi@gmail.com

Plan



- Présentation
- Pourquoi NodeJS ?
- Installation de NodeJS
- Tester NodeJs
- Creation premier serveur
- Intégrer les fichiers HTML
- Passer des paramètres via l'URL
- Evénements
- Les Streams



Présentation

NodeJS est une plateforme construite sur **le moteur JavaScript V8 de Chrome** qui permet de développer des applications en utilisant du JavaScript. Il se distingue des autres plateformes grâce à une ***approche non bloquante permettant d'effectuer des entrées/sorties (I/O) de manière asynchrone.***



Pourquoi NodeJS ?

Avant de commencer à découvrir une nouvelle technologie, il est important d'en comprendre les spécificités. Pour comprendre nous allons partir de la description donnée sur le site officiel :

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Ce que n'est pas NodeJS



*NodeJS n'est pas un framework. Ce n'est pas un outil qui vous permettra de mettre en place une application web rapidement avec peu de code. C'est un outil plus bas niveau qui vous permettra de **communiquer avec le système à travers différentes librairies C++** et avec un langage familier. Comme vu précédemment, c'est un outil que l'on va sélectionner si on a besoin de gérer un grand nombre de demandes sur un seul thread en évitant les lenteurs dû à la nature synchrone d'autres langages.*



Installation de NodeJS

Pour télécharger **NodeJS** il vous suffit de vous rendre sur le [site officiel](#) et de choisir la version que vous souhaitez installer. Si vous souhaitez obtenir plus d'information sur la conséquence de la mention LTS (Long term support) n'hésitez pas à jeter un oeil sur le dépôt [GitHub](#).

Windows & MacOS

Pour Windows et MacOS l'installation se fait au travers d'un exécutable qui vous guidera dans les différentes étapes d'installation et configuration.

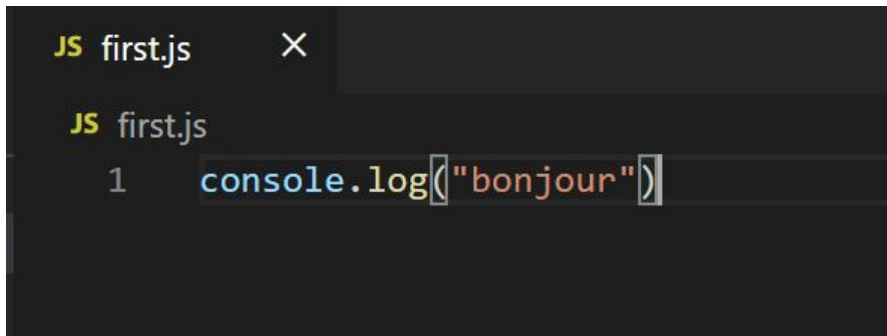
Linux

Sur linux, la méthode la plus simple reste de passer par le gestionnaire de paquets.



Tester NodeJs

Pour tester et voir si le NodeJs est bien installé on va créer un dossier **test** ensuite on met le code suivant:





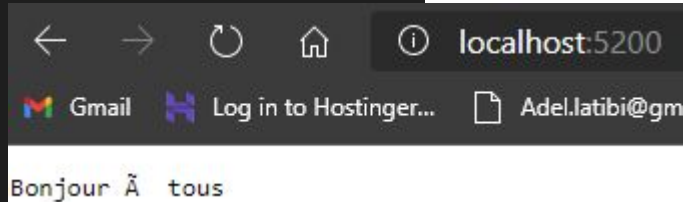
```
JS first.js  X  
JS first.js  
1 console.log("bonjour")
```

Pour exécuter ce code on tape la commande: **node first.js**

Notre premier serveur

1. Création d'un nouveau dossier *formation_nodejs*
2. Création d'un fichier *server.js*


```
JS server_first.js >  server.on('request') callback >  'Content-Type'
1  const http = require('http'); // importation module http
2  // creation un objet server
3  const server = http.createServer()
4  // evenement pour créer un serveur
5  server.on('request', (request,response) => {
6      // l'entete de la réponse http
7      response.writeHead(200, { 'Content-Type': 'text/plain' });
8      response.end("Bonjour à tous"); // la reponse au client
9  })
10 server.listen(5200) // le port
```



Dans l'exemple précédent on a vu comment créer un simple serveur avec la fonction `createServer()`

Mais le problème c'est que d'abord la réponse **http** n'est pas sous format **HTML**, l'encodage de texte n'est pas en **utf-8** pour les tous les caractères, pour ce faire:


JS server_first.js X

JS server_first.js >  server.on('request') callback

```
1  const http = require('http'); // importation module http
2  // creation un objet server
3  const server = http.createServer()
4  // evenement pour créer un serveur
5  server.on('request', (request, response) => {
6      // l'entete de la réponse http
7      response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8 ' });
8      response.end("<h1>Bonjour à tous</h1>"); // la reponse au client
9  })
10 server.listen(5200) // le port
```



Bonjour à tous



Maintenant pour simplifier notre code on a même pas besoin de créer la constante **server** ou d'utiliser l'événement `request` on peut directement faire:

```
1  const http = require('http'); // importation module http
2  // creation d'un serveur
3  http.createServer( (request,response) => {
4      // l'entete de la réponse http
5      response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8 ' });
6      response.end("<h1>Hello world</h1>"); // la reponse au client
7  }).listen(5200) // le port
```

Intégrer les fichiers HTML

Pour l'instant on a vu comment le serveur répond avec une simple chaîne de caractères, mais en réalité, on utilise des fichiers HTML pour afficher notre page. Pour cela on importe le module **fs** (filesystem) [Voir la doc NodeJs](#) et on crée le fichier **index.html**

```
<> index.html >  html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4  |   <meta charset="UTF-8">
5  |   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6  |   <title>Formation NodeJs</title>
7  </head>
8  <body>
9  |   <h1>Bienvenue sur mon site</h1>
10 |   <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Modi nobis quis elige
11 </body>
12 </html>
```

JS server_second.js > ...

```
1  const http = require('http'); // importation module http
2  const fs = require('fs'); // importation module fs pour lire des fichier
3
4  http.createServer((request,response) => {
5    fs.readFile('index.html', (err, data) => {
6      // err si il y a un probelme sur le fichier
7      // data le contenu de fichier html
8      if (err) throw err; // lancer une exception
9      response.writeHead(200,
10     { 'Content-Type': 'text/html; charset=utf-8' });
11     response.end(data);
12   });
13 }).listen(5200)
```

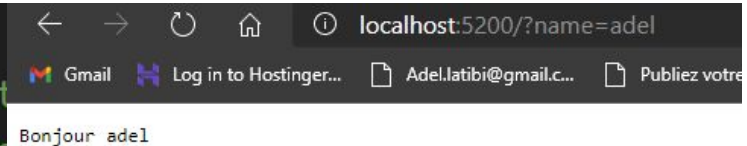
Passer des paramètres via l'URL

Si on a un URL de type: <http://localhost:5200?name=adel>

ici le paramètre est *name* qu'on passe au serveur, et c'est une requête de type *GET*

JS server.js > ...

```
1  const http = require('http'); // importat
2  const url = require('url'); // importati
3
4  http.createServer( (request,response) => {
5      let query = url.parse(request.url,true).query
6      response.writeHead(200)
7      response.end("Bonjour " + query.name)
8  }).listen(5200)
```



Evénements

```
JS server_event2.js > ...
1  const EventEmitter = require('events');
2  const monEcouleur = new EventEmitter()
3
4  monEcouleur.on("saute", (a,b) => {
5    console.log("j'ai sauté",a,b)
6  })
7
8  monEcouleur.emit("saute",10,20)
9  monEcouleur.emit("saute")
10 monEcouleur.emit("saute")
```

```
PS C:\Users\adell\projects\cours_nodejs> node server_event2.js
j'ai sauté 10 20
j'ai sauté undefined undefined
j'ai sauté undefined undefined
```

Evénements 2



```
JS server_event2.js > ...
1   const EventEmitter = require('events');
2   const monEcouteur = new EventEmitter()
3
4   monEcouteur.once("saute", (a,b) => {
5     console.log("j'ai sauté",a,b)
6   })
7
8   monEcouteur.emit("saute",10,20)
9   monEcouteur.emit("saute")
10  monEcouteur.emit("saute")
```


Example events

```
1  const HttpServer = require('http'); // importation module http
2  const EventEmitter = require('events'); // importation module events
3
4  const App = {
5    start: function (port) {
6      const emitter = new EventEmitter();
7      // creation d'un serveur
8      HttpServer.createServer( (request,response) => {
9        // l'entete de la réponse http
10       response.writeHead(200, { 'Content-Type': 'text/html; charset=utf-8 ' });
11       emitter.emit('root',response)
12       response.end(); // la reponse au client
13     }).listen(port) // le port
14
15     return emitter;
16   }
17 }
18
19
20 const app = App.start(5050)
21 app.on('root',(response)=>{
22   response.write("<h1>Hello world of events</h1>")
23 })
24
```


Les Streams

Le code ci-dessous c'est la méthode classique pour créer des copies des fichiers qui comporte des inconvénients.

```
1  const fs = require('fs')
2
3  fs.readFile('logo.png', (err, data) => {
4    if (err) throw err;
5    fs.writeFile('copy.png', data, (err) => {
6      if (err) throw err;
7      console.log('le fichier a bien été copié')
8    })
9  })
```

Les Streams 2

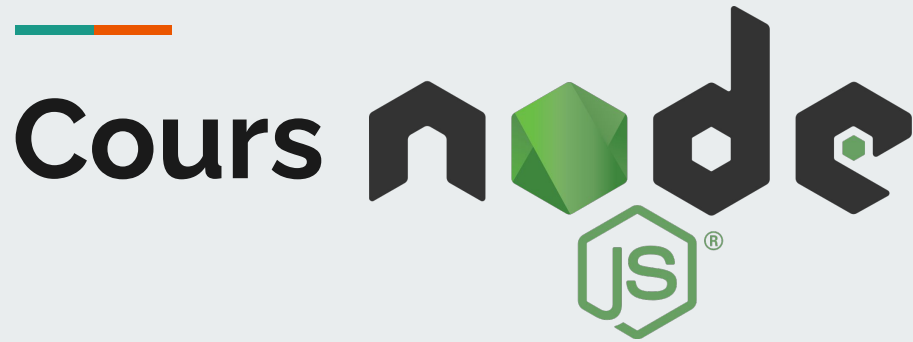
```
JS stream.js > [?] read
1  const fs = require('fs')
2
3  file = "logo.png"
4  const read = fs.createReadStream(file)
5  read.on('data', (chunk) => {
6    console.log("j'ai lu "+chunk.length)
7  })
8
9
10 read.on('end', ()=>{
11   console.log("j'ai fini de lire le fichier")
12 })
```

Les Streams 3 stats

```
JS stream_stats.js > ...
1  const fs = require('fs')
2
3  fs.stat("logo.png", (err, stat) => {
4    const total = stat.size
5    let progress = 0
6
7    const read = fs.createReadStream("logo.png")
8    read.on('data', (chunk) => {
9      progress += chunk.length
10     p = Math.round(100 * progress / total)
11     console.log("j'ai lu "+p+" %")
12   })
13
14   read.on('end', () => {
15     console.log("j'ai fini de lire le fichier")
16   })
17 })
18
```

Les Streams 4

```
JS stream_copy.js > ...
1  const fs = require('fs')
2
3  file = "logo.png"
4  const read = fs.createReadStream(file)
5  const write = fs.createWriteStream("logo_cp.png")
6  |
7  read.on('data', (chunk) => {
8  |    console.log("j'ai lu "+chunk.length)
9  |  })
10
11 read.pipe(write)
12
13 read.on('finish', ()=>{
14 |   console.log("j'ai fini de lire le fichier")
15 | })
```



Adel Latibi
adel.latibi@gmail.com

Plan



- Module & NPM
- lodash
- Ramda
- Express
- Gestion d'erreurs
- Micro-framework ExpressJs
- ExpressJs (Les routes)
- ExpressJs (Moteur de template)
- ExpressJs (POST)
- ExpressJs (GET)
- ExpressJs (Cookies)
- ExpressJs (Sessions)
- ExpressJs Connexion avec MySQL
- ExpressJs (Les middlewares)

Modules et NPM



Jusqu'à maintenant nous avons écrit l'ensemble de notre code dans un seul fichier JavaScript, cependant dans cas réel on va diviser notre application sur plusieurs modules.

NodeJS dispose d'un système permettant de découper notre application sous forme de modules. Il est ainsi possible de créer un fichier JavaScript séparé qui va disposer d'une portée locale. Toutes les variables qui y sont définies ne sont disponibles qu'au sein de ce fichier et ne seront pas accessibles depuis l'extérieur.

```
server > JS server.js > ...  
1  const hello = require("./hello")  
2
```


```
server > JS hello.js  
1  console.log("Bonjour à tous")
```


Exporter les Variable, fonctions (1)

```
server > JS server.js > ...  
1  const object = require("./hello")  
2  
3  
4  console.log(object.nom)  
5  console.log(object.age)  
6  console.log(object.add(15,5))  
7
```

```
server > JS hello.js >  add  
1  let nom = 'Mark Sloan'  
2  let age = 35  
3  
4  function add(a,b){  
5      return a+b  
6  }  
7  
8  module.exports = {  
9      nom, age, add  
10 }
```


Exporter les Variable, fonctions (2)

```
server > JS server.js > [?] object
1  const object = require("./hello")
2  
3
4  console.log(object.nom)
5  console.log(object.age)
6  console.log(object.add(15,5))
7
```

```
server > JS hello.js >  add
1  exports.nom = 'Mark Sloan'
2  exports.age = 35
3
4  exports.add = function (a,b){
5      return a+b
6  }
```

Exporter les Variable, fonctions (3)

```
server > JS app.js > [App] App > start
1  const HttpServer = require('http')
2  // importation de l'evenement
3  const EventEmitter = require('events');
4  const FS = require('fs')
5  // importation URL
6  const URL = require('url')
7
8  exports.App = {
9    start : function (port,page){
10      const emitter = new EventEmitter();
11      HttpServer.createServer((request,response)=>{
12        FS.readFile(page,"utf8",(err,data)=>{
13          if(err) throw err;
14          // header http
15          response.writeHead(200,{ 'Content-type': 'text/html; charset=utf-8'});
16          // body http const URL: typeof import("url")
17          let query = URL.parse(request.url,true).query
18          emitter.emit('server',response,data,query)
19          response.end();
20        })
21      }).listen(port)
22      return emitter;
23    }
24  }
```



```
server > JS server.js > [⌘] app
```

```
1  const server = require("./app")
2  const app = server.App.start(5050, "home.html")
3  app.on('server', (response, data, query) => {
4      let html = data.replace("{{name}}", query.name)
5      html = html.replace("{{age}}", query.age)
6      response.write(html)
7  })
```

Npm



NodeJS dispose d'une communauté assez importante qui partage ses modules au travers du gestionnaire de paquet NPM. Il permet de télécharger rapidement un module ainsi que ses différentes dépendances.

On commencera par initialiser **npm**

via la commande: ***npm init***,

ce qui aura pour effet de créer un fichier

package.json qui permettra de suivre

les modules à importer.

```
PS C:\Users\adell\projects\cours_nodejs\server> npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (server)
version: (1.0.0)
description: Example de serveur en NodeJs
entry point: (index.js) server.js
test command:
git repository:
keywords:
author: Adel Latibi
license: (ISC)
About to write to C:\Users\adell\projects\cours_nodejs\server\package.json:
```

Npm

Une fois notre projet initialisé il est possible de [télécharger](#) un module simplement via la commande

```
npm install --save <module>
```

Maintenant on va installer **lodash** le module le plus populaire de **npm** :

```
C:\Users\adell\projects\cours_nodejs\server>npm i -S lodash
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN server@1.0.0 No repository field.

+ lodash@4.17.20
added 1 package from 2 contributors and audited 1 package in 2.084s
found 0 vulnerabilities
```

Remarque:

Si vous voulez que le serveur se redémarre automatiquement une fois le fichier **js** sauvegardé installer **nodemon** et remplacer la commande **node** avec **nodemon**: `npm install -g nodemon`

NPM (lodash) examples

```
1  const _ = require("lodash")
2
3  let words = ['sky', 'wood', 'forest', 'falcon',
4              'pear', 'ocean', 'universe'];
5
6  let fel = _.first(words);
7  let lel = _.last(words);
8
9  console.log(`First element: ${fel}`);|
10 console.log(`Last element: ${lel}`);
```

```
server > JS server.js > [?] c2
1  const _ = require("lodash")
2
3  let nums = [1, 2, 3, 4, 5, 6, 7, 8, 9];
4
5  let c1 = _.slice(nums, 2, 6);
6  console.log(c1);
7  💡
8  let c2 = _.slice(nums, 0, 8);
9  console.log(c2);
```

Vous retrouverez la suite de l'exemple sur [zetcode](#)

NPM (ramda) examples

[ramda](#) est un module souvent utilisé sur NodeJs, contient plusieurs fonctions utiles pour le programmeur

server > JS server.js > ...

```
1  const R = require('ramda');
2
3  const users = [
4    { name: 'John', age: 25 },
5    { name: 'Lenny', age: 51 },
6    { name: 'Andrew', age: 43 },
7    { name: 'Peter', age: 81 },
8    { name: 'Anna', age: 43 },
9    { name: 'Albert', age: 76 },
10   { name: 'Adam', age: 47 },
11   { name: 'Robert', age: 72 }
12 ];
13
14 console.log(R.pluck('age', users));
15 console.log(R.pluck('name', users));
```

```
1  const R = require('ramda');
2
3  const users = [
4    { name: 'John', city: 'London', born: '2001-04-01' },
5    { name: 'Lenny', city: 'New York', born: '1997-12-11' },
6    { name: 'Andrew', city: 'Boston', born: '1987-02-22' },
7    { name: 'Peter', city: 'Prague', born: '1936-03-24' },
8    { name: 'Anna', city: 'Bratislava', born: '1973-11-18' },
9    { name: 'Albert', city: 'Bratislava', born: '1940-12-11' },
10   { name: 'Adam', city: 'Trnava', born: '1983-12-01' },
11   { name: 'Robert', city: 'Bratislava', born: '1935-05-15' },
12   { name: 'Robert', city: 'Prague', born: '1998-03-14' }
13 ];
14
15 let res1 = R.filter(R.where({ city: R.equals('Bratislava') }))(users);
16 console.log(res1);
```


NPM (express)

Coder des serveurs Web en Node pur

est possible, mais long. L'utilisation

du framework Express simplifie

la programmation de serveur web.

Pour installer express:

```
npm install --save express
```

```
let express = require('express')
let app = express()

app.get('/', (request, response) => {
  response.send('Bonjour')
})
app.get('/demo', (request, response) => {
  response.send('Bonjour je suis la démo')
})
app.post('/', (request, response) => {
  // Traitement des données
})

app.listen(8080)
```


NodeJs Gestion d'erreurs

Nous allons apprendre à créer des objets *Error* et à lancer et gérer des erreurs dans *Node.js*

Modifications futures liées aux meilleures pratiques en matière de gestion des erreurs.

```
2  var err = new Error("The error message");
3  console.log(err.message); //affiche: The error message
4  console.log(err); // affiche: le stack
5
```

```
2  try {
3      var a = 1;
4      throw new Error("Some error message"); // lancer l'erreur
5      console.log(a); //this line will not be executed;
6  } catch (error) {
7      console.log(error.message); //will be the above thrown error
8  }
```

ExpressJs

[ExpressJS](#) est une librairie qui vous permettra de créer une application Web plus simplement qu'avec l'objet http directement. Elle fournit un ensemble de méthode permettant de traiter les requêtes HTTP et fournit un système de middleware pour étendre ses fonctionnalités.

```
JS server.js  X  JS app.js
server > JS server.js > app.get('/') callback
1  const app = require('./app');
2  app.get('/', (request, response) => {
3    response.send("<h1>Bonjour express</h1>")
4  })
5
6  app.listen(3000)
```

```
JS server.js  JS app.js  ●
server > JS app.js > ...
1  const express = require('express');
2
3  const app = express();
4
5  module.exports = app;
```

ExpressJs (Les routes)

Comme beaucoup de frameworks web, ExpressJS se présente comme un routeur où l'on va déclarer les routes supportées par notre application ainsi que le traitement à effectuer lorsque cette dernière est rencontrée.


```
1  const app = require('./app');
2  // la route /
3  app.get('/',(request,response)=>{
4    |    response.send("<h1>Bonjour express</h1>")
5  |  })
6  // la route demo
7  app.get('/demo',(request,response)=>{
8    |    response.send("<h1>salut tu es sur la page demo</h1>")
9  |  })
10 // la route contact
11 app.get('/contact',(request,response)=>{
12 |    response.send("<h1>salut tu es sur la page demo</h1>")
13 |  })
14 app.listen(3000)
```





ExpressJs (Moteur de templating ejs)

installer avec: `npm install ejs --save`

```
server > JS server.js > ...
1  const app = require('./app');
2  // importer ejs et l'inclure dans express
3  app.set('view engine', 'ejs')
4  // la route /
5  app.get('/', (request, response) => {
6    |   response.render('pages/index', {test: 'salut'})
7  })
8  // la route demo
9  app.get('/demo', (request, response) => {
10 |   response.send("<h1>salut tu es sur la page demo</h1>")
11 | })
12 // la route contact
13 app.get('/contact', (request, response) => {
14 |   response.send("<h1>salut tu es sur la page demo</h1>")
15 | })
16 app.listen(3000)
```

Ensuite on va créer *view > pages > index.ejs*



```
server > views > pages > <> index.ejs >  html >  body >  h1 >  ?  
1 <!DOCTYPE html>  
2 <html lang="en">  
3 <head>  
4 |   <meta charset="UTF-8">  
5 |   <meta name="viewport" content="width=device-width, initial-scale=1.0">  
6 |   <title>Home</title>  
7 </head>  
8 <body>  
9 |   <h1><%=test%></h1>  
10 </body>  
11 </html>
```

ExpressJs ajouter les fichiers statiques

1. On crée un dossier **public** dans la racine de projet ensuite **css** ensuite **style.css**
2. Ensuite on met la balise **link** pour importer le fichier css dans html
3. Ensuite on précise le dossier **public** en comme le dossier pour chercher les fichiers statiques

```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <link rel="stylesheet" href="css/style.css">
```

```
// set static folder
app.use(express.static('public'))
```

Requête GET

```
// la route /
app.get('/', (request, response) => {
  response.render('pages/index', {test: 'Message', query: request.query})
})
```



```
<body>
  <h1><%=test%></h1>
  <% if(locals.query.name && locals.query.age){ %>
    <p>
      Bonjour je suis <b><%= query.name %></b> j'ai <b><%=query.age%></b> ans
    </p>
  <% } %>
```


Formulaire POST

Pour pouvoir utiliser Opération Post on doit d'abord installer Body-Parser: *npm i --save body-parser*

```
server > JS app.js > ...
1  const express = require('express');
2  let bodyParser = require('body-parser')
3
4  const app = express();
5  // importer ejs (moteur de template) et l'inclure dans express
6  app.set('view engine','ejs')
7  // set static folder (gestion des fichiers statiques)
8  app.use(express.static('public'))
9  // middleware
10 app.use(bodyParser.urlencoded({ extended: false }))
11 app.use(bodyParser.json())
12
13 module.exports = app;
```


Formulaire POST

server > JS server.js >  app.post('/') callback >  message

```
1  const app = require('./app');
2
3  // la route /
4  app.get('/',(request,response)=>{
5    |   response.render('pages/index',{test:'Message'})
6  | })
7  // la route /
8  app.post('/',(request,response)=>{
9    |   if(request.body.message === undefined || request.body.message === ''){
10   |     response.render('pages/index',{test:'Message',error:"vous n'avez pas entré de message"})
11   |   }else{
12   |      response.render('pages/index',{test:'Message',message:request.body.message})
13   |   }
14 | })
15 app.listen(3000)
```

Formulaire POST

```
9  <body>
10  <h1><%=test%></h1>
11  <form action="/" method="post">
12      <textarea name="message" id="" cols="30" rows="10"></textarea><br>
13      <input type="submit" value="Envoyer">
14      <% if(locals.error){ %>
15          <b>Erreur: <%=error%></b>
16      <% }%>
17
18      <% if(locals.message){ %>
19          <b>message: <%=message%></b>
20      <% }%>
21
22  </form>
23  </body>
```

Cookies



npm install cookie-parser

```
const cookieParser = require('cookie-parser')
```

```
app.use(cookieParser())
```

```
4 // 1a route /
5 ▼ app.get('/',(request,response)=>{
6   // Cookies that have not been signed|
7 ▼   if(request.cookies.name===undefined){
8     response.cookie('name',"adel",{maxAge:1000})
9   }
```

Sessions



[mysql Docs](#)

installer: *npm install express-session*

```
4  const session = require('express-session')
```

```
18  app.use(session({
19    secret: '&é"(-è_çà)23s6d5f4sd6f8z7er9@',
20    resave: false,
21    saveUninitialized: true,
22    cookie: { secure: false } // no https
23  })))
```

```
if(request.session.username){
  console.log(request.session.username)
}else{
  request.session.username = "adel"
}
```

Connexion Nodejs avec MySQL

Comme d'habitude pour pouvoir connecter nodejs avec la base de données on va installer un module:

```
npm i --save mysql
```

```
server > JS bdd.js > [?] connection > 🔑 user
1  var mysql      = require('mysql');
2  var connection = mysql.createConnection({
3    host      : 'localhost',
4    user      : 'symfony',
5    password  : '',
6    database  : 'nodejs'
7  });
8
9  connection.connect();
10
11 module.exports = connection
12
```

server > JS server.js >  app.post('/') callback

```
1  const app = require('./app');
2  const connection = require('./bdd')
3  // la route /
4  app.get('/',(request,response)=>{
5      response.render('pages/index',{test:'Message'})
6  })
7  // la route /
8  app.post('/',(request,response)=>{
9      if(request.body.message === undefined || request.body.message === ''){
10         response.render('pages/index',{test:'Message',error:"vous n'avez pas entré de message"})
11     }else{
12         connection.query('INSERT INTO message set ?', { text: request.body.message}, function (error, results, fields)
13             if (error) throw error;
14             // ...
15         });
16         //connection.end();
17         response.render('pages/index',{test:'Message',message:request.body.message})
18     }
19 })
20 app.listen(3000)
```

ExpressJs (Les middlewares)

Les middlewares permettent d'effectuer un traitement avant celui défini par les routes. On trouvera de nombreux middleware disponible sur **NPM** comme par exemple **body-parser** permettant de gérer les données postées par un formulaire par exemple.

```
var express = require('express');
var app = express();

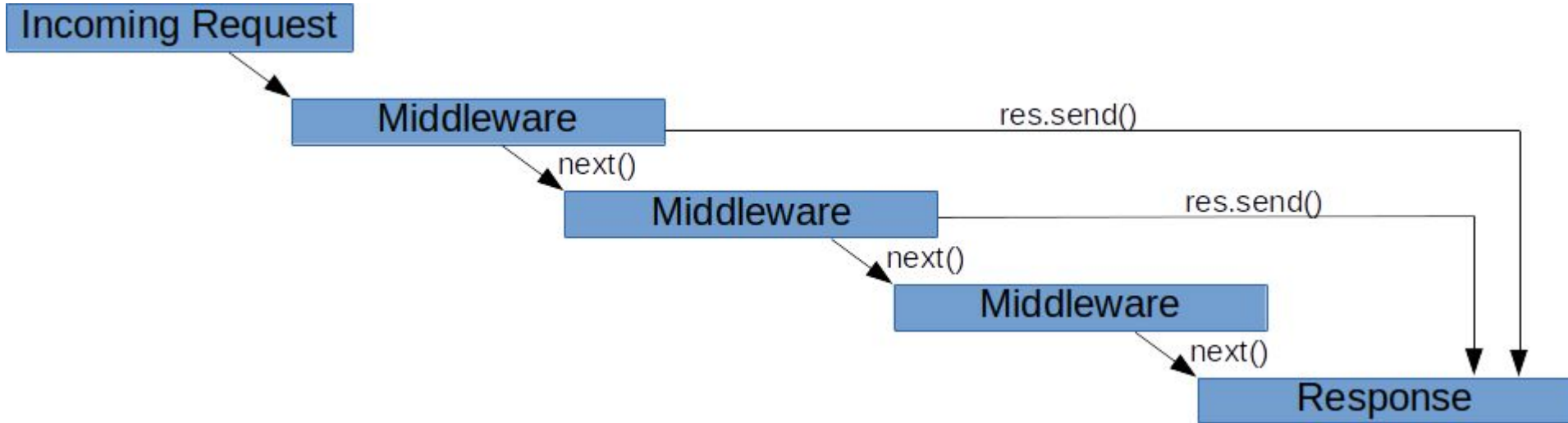
app.get('/', function(req, res, next) {
  next();
});

app.listen(3000);
```

The diagram illustrates the usage of Express.js middlewares with the following annotations:

- `express`: Méthode HTTP à laquelle la fonction middleware s'applique.
- `app`: Chemin (route) auquel la fonction middleware s'applique.
- `function(req, res, next)`: Fonction de middleware.
- `req`: Argument de **demande** HTTP à la fonction middleware, appelé "req" par convention.
- `res`: Argument de **réponse** HTTP à la fonction middleware, appelé "res" par convention.
- `next()`: Argument de rappel à la fonction middleware, appelée "next" par convention.

L'ordre dans les middlewares est important



Les middlewares Examples



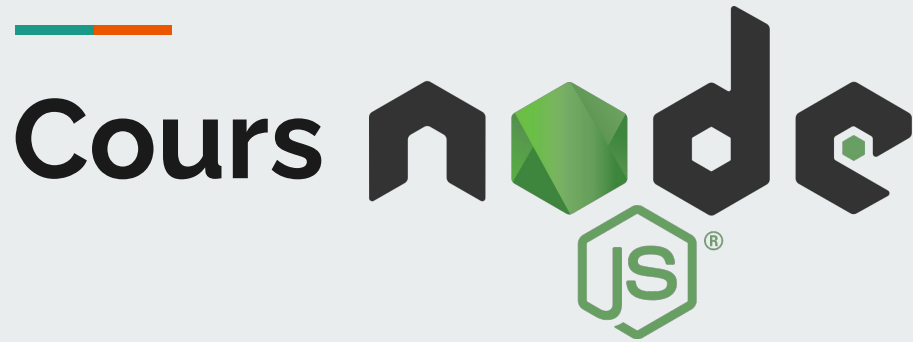
```
25  app.use((req, res, next) => {
26      console.log('Requête reçue !');
27      next();
28  });
29
30  app.use((req, res, next) => {
31      console.log("Time:", Date.now());
32      next();
33  });
34
```

Exercice



Créer une page **login(email, password)**, **inscription (nom, prénom, email, username, password, age, ville, adresse)** et **monCompte**:

- une fois l'utilisateur inscrit la page se redirige vers **login**
- une fois l'utilisateur connecté la page se redirige vers **monCompte**



Adel Latibi
adel.latibi@gmail.com



Plan



- Framework **AdonisJs**
- Installation AdonisJs
- 1er Application
- Comprendre l'architecture
- AdonisJs (Les routes)
- AdonisJs (Controller)
- Adonis (Request)
- AdonisJs (Response)
- AdonisJs (POST)
- AdonisJs (GET)
- Adonis (View)
- AdonisJs (Moteur de template)
- AdonisJs Connexion avec MySQL
- AdonisJs (Cookies)
- AdonisJs (Sessions)

A short horizontal bar with a teal segment on the left and an orange segment on the right.

Framework AdonisJs

- C'est un framework (Complet) serveur **NodeJs**
- AdonisJs utilise l'architecture MVC
- AdonisJs reprend l'architecture de **Laravel**
- Son ORM support: (Postgres, MySQL, SQLITE, MANGODB ... etc)



Installation AdonisJs

- Pour installer AdonisJS, nous avons besoin de *nodejs* et *npm*
- Pour installer AdonisJs: *npm i -g @adonisjs/cli*
 - Node.js >= 8.0.0
 - npm >= 3.0.0



1er Application AdonisJs

Pour créer une nouvelle application AdonisJs on utilise la commande suivante:

adonis new <application_name>

une fois l'application est cloné pour démarrer le serveur on utilise la commande suivante:

- ***cd <application_name>***
- ***adonis serve --dev***

Cette application va utiliser le port appliquer sur le fichier **.env**



Comprendre l'architecture 1/2

Maintenant qu'on a installer AdonisJs et créé notre 1er application, on va voir l'architecture de notre Framework:

- **app**: ce dossier contient la logique de notre application (Controllers, Models ...)
- **config**: ce dossier est utilisé pour définir les configurations de l'application
- **database**: ce dossier utilisé pour stocker les fichiers relatives aux base de données
- **public**: ce dossier les fichier statiques (js,css,img...)

```
.
├─ app/
│   └─ ...
├─ config/
│   ├── app.js
│   └─ auth.js
│       └─ ...
├─ database/
│   ├── migrations/
│   └─ seeds/
│       └─ factory.js
└─ public/
```




Comprendre l'architecture 2/2

- **resources:** ce dossier va contenir les vues de notre application (ici la vue en langage HTML mais extension est en **.edge**)
- **start:** ce dossier utilisé pour stocker les fichiers (**app.js**, **kernel.js** et **routes.js**) qui sont chargés lors de démarrage de notre application
- **test:** ce dossier utilisé pour stocker tous les testes de l'application adonisJS

```
├─ resources/  
  └─ ...  
    └─ views/  
├─ start/  
  └─ app.js  
  └─ kernel.js  
  └─ routes.js  
└─ test/
```

AdonisJs (Les routes) (1/2)

Pour pouvoir ajouter des routes dans AdonisJs, on utilise le fichier *start > routes.js*

la route la plus basique dans adonis:

```
1 Route.get('/', () => 'Hello Adonis')
```

start/routes.js

On peut aussi ajouter la route avec un controller.méthode comme suite:

```
1 Route.get('posts', 'PostController.index')
```

start/routes.js

AdonisJs (Les routes) (2/2)

On peut enregistrer des routes avec des multiples opération (POST,GET,PUT...):

```
1 Route.route('/', () => {  
2   //  
3 }, ['GET', 'POST', 'PUT'])
```

Ou spécifier directement l'opération à utiliser:

```
1 Route.get(url, closure)  
2 Route.post(url, closure)  
3 Route.put(url, closure)  
4 Route.patch(url, closure)  
5 Route.delete(url, closure)
```

AdonisJs (Controller)

Le contrôleur l'un des pilier de l'architecture MVC, qui traite les actions de l'utilisateur, modifie les données du modèle et de la vue.

Pour créer un contrôleur dans AdonisJs on utilise la commande suivante:

```
1 # HTTP Controller
2 > adonis make:controller User --type http
```

```
JS routes.js × JS UserController.js ●
app > Controllers > Http > JS UserController.js > ...
1  'use strict'
2
3  class UserController {
4
5  }
6
7  module.exports = UserController
8  |
```

AdonisJs (Controller)

Pour utiliser le contrôleur on doit l'injecter lors de l'enregistrement de la route dans *start/routes.js*:

```
app/routes.js
1 // app/Controllers/Http/UserController -> index()
2 Route.get(url, 'UserController.index')
3
4 // app/Controllers/Http/Admin/UserController -> store()
5 Route.post(url, 'Admin/UserController.store')
6
7 // app/MyOwnControllers/UserController -> index()
8 Route.post(url, 'App/MyOwnControllers/UserController.index')
```

```
app/Controllers/Http/UserController.js
1 'use strict'
2
3 class UserController {
4   index ({ request, response }) {
5     //
6   }
7 }
8
9 module.exports = UserController
```



AdonisJs (Request)

AdonisJs passe la requête http en tant qu'objet

```
3 class UserController {  
4  
5   index = ({request, view}) => {  
6  
7   }  
8 }
```

AdonisJs (Response)

Pour renvoyer une réponse http (text)

```
3 class UserController {
4
5   index = ({response})=>{
6     response.send('hello world')
7   }
8 }
9
10 module.exports = UserController
```

pour retourner un json:

```
3 class UserController {
4
5   index = ({response})=>{
6     response.json(
7       {
8         name: "Taylor",
9         age: '22'
10      }
11    )
12  }
13 }
```

AdonisJs (Response -> Redirects)

Pour rediriger vers un autre url

```
1 response.redirect('/url')
```

Pour rediriger via controller en ajoutant des paramètres:

```
4 // via controller method  
5 response.route('UserController.show', { id: 1 })
```


Adonis (View)

Tous les vues sont stockées dans le dossier *resources/views* avec l'extension *.edge*,

pour créer une vue on utilise la commande

```
C:\Users\adell\projects\cours_nodejs\blog>adonis make:view blog  
✓ create resources\views\blog.edge
```

```
4  
5  ✓   index = ({request,response,view})=>{  
6      |   return view.render("blog")  
7      |   }  
8  }  
9
```

AdonisJs (Moteur de template edge)

pour passer des variables de *contrôleur* vers *la vue*:

```
index = ({request,response,view})=>{  
  const name= "Eve"  
  return view.render("blog",{name: name})  
}
```

```
<body>  
  <h1>Blog Nodejs</h1>  
  {{ name }}
```

On peut accéder à l'objet request dans html en utilisant:

```
1 The request URL is {{ request.url() }}
```

Ajouter les fichiers css dans la vue

Relative path (to CSS files in the `public` directory):

```
1  {{ style('style') }}
```

```
1  <link rel="stylesheet" href="/style.css" />
```

Ajouter les fichiers js dans la vue

Relative path (to JavaScript files in the `public` directory):

```
1  {{ script('app') }}
```

```
1  <script type="text/javascript" src="/app.js"></script>
```

Ajouter des images dans la vue

Returns path of a file relative to the `public` directory:

```
1 
```

```
1 
```

AdonisJs (GET)



Pour récupérer les requêtes GET:

```
2
3  class UserController {
4
5      index = ({request, view})=>{
6          const GET = request.get()
7          console.log(GET)
8      }
9  }
10
11  module.exports = UserController
12
```

AdonisJs (POST)



Pour récupérer les information du formulaire POST:

```
index = ({request,response,view})=>{  
  const POST= request.post()  
  return view.render("blog",{post: POST})  
}
```

AdonisJs (POST)

Pour la méthode POST on doit ajouter dans formulaire `{{csrfField()}}`

```
11 | <h2>message: {{post.msg}}</h2>
12 | <form action="/blog" method="post">
13 |   {{ csrfField() }}
14 |   <label for="msg">Message:</label> <br>
15 |   <textarea name="msg" id="msg" cols="30" rows="10"></textarea> <br>
16 |   <input type="submit" value="send">
17 | </form>
```


AdonisJs (request.all())

la fonction all() de l'objet request retourne un objet union de méthodes post() et get()

```
3  class UserController {  
4  
5      index = ({request, view}) => {  
6          const ALL = request.all()  
7          console.log(ALL)  
8      }  
9  }  
10  
11  module.exports = UserController  
12
```

AdonisJs Connexion avec MySQL

Pour pouvoir connect le framework AdonisJs avec une base de données on doit modifier le fichier `.env`:

Ensuite créer la base de données dans phpmyadmin

Et enfin lancer les migration avec la commande:

adonis migration:run

```
1  HOST=127.0.0.1
2  PORT=3333
3  NODE_ENV=development
4  APP_URL=http://${HOST}:${PORT}
5  CACHE_VIEWS=false
6  APP_KEY=iPW8wP2bB9HuJBnE0muvABGLz1ksH1kR
7  DB_CONNECTION=mysql
8  DB_HOST=127.0.0.1
9  DB_PORT=3306
10 DB_USER=symfony
11 DB_PASSWORD=
12 DB_DATABASE=adonis
13 SESSION_DRIVER=cookie
14 HASH_DRIVER=bcrypt
```

```
C:\Users\adell\projects\cours_nodejs\blog>adonis migration:run
migrate: 1503248427885_user.js
migrate: 1503248427886_token.js
Database migrated successfully in 372 ms
```

Adonis (Model)



Pour générer un fichier model on utilise la commande

```
C:\Users\adell\projects\cours_nodejs\blog>adonis make:model blog
✓ create app\Models\Blog.js
```

```
4   const Model = use('Model')
5
6   class Blog extends Model {
7   }
8
9   module.exports = Blog
10
```

Adonis (Model)



La commande suivante va créer le model blog et le fichier migration de blog

```
C:\Users\adell\projects\cours_nodejs\blog>adonis make:model blog --migration
✓ create  app\Models\Blog.js
✓ create  database\migrations\1607569596282_blog_schema.js
```

Pour appliquer les migrations dans la base de données on utilise la commande suivante:

```
C:\Users\adell\projects\cours_nodejs\blog>adonis migration:run
migrate: 1607569596282_blog_schema.js
Database migrated successfully in 199 ms
```



Adonis (Model) creation article

Pour créer une ligne dans la table blog dans la base de données:

```
const Blog = use('App/models/blog')
```

```
const blog = new Blog()  
blog.title = "python"  
blog.content = "blablabla..."  
blog.save()
```

Adonis (Model) Afficher la liste des articles

```
3  const Blog = use('App/models/blog')  
4  const Database = use('Database')
```

```
index = async ({request,response,view})=>{  
  
  const posts = await Database  
    .table('blogs')  
    .orderBy('id', 'desc')  
  
  console.log(posts)
```



AdonisJs (Cookies)

```
if(request.cookie('username')){  
  console.log(request.cookie('username'))  
}else{  
  response.cookie('username', "shadowkiller")  
}
```



AdonisJs (Sessions)

```
index = async ({request,session,response,view})=>{  
  if(session.get('email')){  
    console.log("session:",session.get('email'))  
  }else{  
    session.put('email',"adel.latibi@gmail.com")  
  }  
  return view.render("blog")  
}
```




Références

1. Grafikart
2. openclassroom
3. [NodeJs Docs](#)
4. [DevDocs](#)