
Express.js

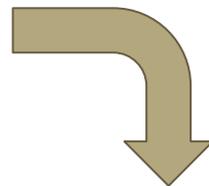
S.Gounane
ISIL 2018-2019

Le framework Express.js

- Express.js est un micro-framework pour Node.js.
- Il met à disposition des outils de base pour aller plus vite dans la création d'applications Node.js.
- Pas assez complet comme les framework php (laravel, symphony ..) python(django).
- Pour installer express.js
 - ***npm install express***

Express.js: firstApp

```
1 var http = require('http');
2 var url = require('url');
3
4 var server = http.createServer(function(req, res) {
5   var page = url.parse(req.url).pathname;
6   console.log(page);
7   res.writeHead(200, {"Content-Type": "text/plain"});
8   if (page == '/') {
9     res.write('Accueil');
10  }
11  res.end();
12 });
13
14 server.listen(3000);
```



```
1 var express = require('express');
2
3 var app = express();
4
5 app.get('/', function(req, res) {
6   res.setHeader('Content-Type', 'text/plain');
7   res.send('Accueil');
8 });
9
10 app.listen(3000);
```

Les routes

```
1 var express = require('express');
2 var app = express();
3
4 app.get('/', function(req, res) {
5     res.setHeader('Content-Type', 'text/plain');
6     res.write('Accueil');
7 });
8
9 app.get('/contact', function(req, res) {
10    res.setHeader('Content-Type', 'text/plain');
11    res.write('Nous contscter');
12 });
13
14 app.get('/cours/web/nodejs', function(req, res) {
15    res.setHeader('Content-Type', 'text/plain');
16    res.write('Node.js is greate');
17 });
18
19 app.use(function(req, res, next){
20    res.setHeader('Content-Type', 'text/plain');
21    res.status(404).send('Page introuvable !');
22 });
23
24 app.listen(3000);
```

Les routes dynamiques

Express permet de gérer des routes dont certaines portions peuvent varier.

En insérant une variable *:mavariabile* dans l'URL de la route

Ce qui crée un paramètre accessible avec *req.params.mavariabile*.

```
26 app.get('/capteurs/:id/:temp', function(req, res) {
27   res.setHeader('Content-Type', 'text/plain');
28   res.end('La temperature mesurerpar le capteur n°' +
29     req.params.id + ' est: ' + req.params.);
30 });
```

Les middlewares

- Ce sont des petits morceaux d'application qui rendent chacun un service spécifique.
- Les middlewares de base dans Express ne sont pas nombreux:
 - **compression** : permet la compression gzip de la page
 - **cookie-parser** : permet de manipuler les cookies
 - **cookie-session** : permet de gérer des informations de session (durant la visite d'un visiteur)
 - **serve-static** : permet de renvoyer des fichiers statiques contenus dans un dossier (images, fichiers à télécharger...)
 - **serve-favicon** : permet de renvoyer la favicon du site
 - etc.

les middlewares

- Ces middlewares sont interconnectés et peuvent communiquer entre eux en se renvoyant jusqu'à 4 paramètres
 - **err**: les erreurs
 - **req**: la requête du visiteur
 - **res**: la réponse à renvoyer (la page HTML et les informations d'en-tête)
 - **next**: un callback vers la prochaine fonction à appeler

Les middlewares

```
1 var express = require('express');
2 var favicon = require('serve-favicon'); // Charge le middleware de favicon
3
4
5 var app = express();
6 // Indiquer que le dossier /public contient des fichiers
7 //statiques (middleware chargé de base)
8 app.use(express.static(__dirname + '/public'));
9
10 // Activer la favicon indiquée
11 app.use(favicon(__dirname + '/public/favicon.ico'));
12
13 // Répondre a la requete
14 app.use(function(req, res){
15     res.send('Hello');
16 });
17
18 app.listen(3000);|
```

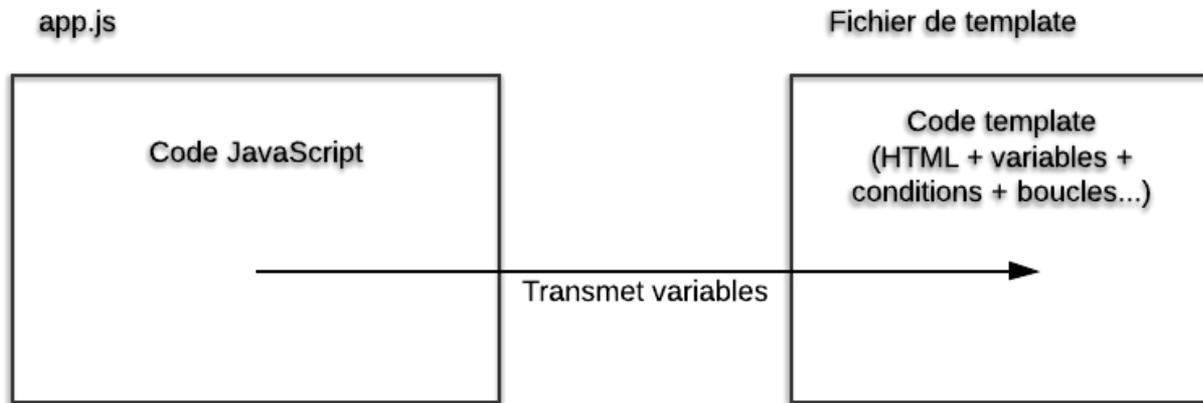
Les templates

- Envoyer un contenu Html en utilisant cette méthode n'est pas une tâche agréable
- Les templates permettent de produire du HTML et d'insérer au milieu du contenu, des variables (javascript).
- Il existe beaucoup de langages de templates: Mustache, Underscore, EJS, JSP, HandlebarsJS, pug (Jade)...

```
res.write('<!DOCTYPE html>'+
'<html>'+
'  <head>'+
'    <meta charset="utf-8" />'+
'    <title>Ma page Node.js !</title>'+
'  </head>'+
'  <body>'+
'    <p>Voici un paragraphe <strong>HTML</strong> !</p>'+
'  </body>'+
'</html>');
```

Les templates

- Principe:
 - depuis un fichier JavaScript, on appelle une template en lui transmettant les variables dont il a besoin pour construire la page.



PUG

1. Pug est un moteur de templates implémenté en JavaScript
2. Il permet de générer dynamiquement du HTML
3. Il est minimaliste et basé sur les indentations.
4. Pug était auparavant connu sous le nom de Jade

Pug: Exemple

- Installer **pug** dans le dossier de votre projet
 - `$npm install pug`
- Dans l'application express il faut définir le moteur de templates utilisé et le chemin d'accès aux templates
 - `app.set("view engine", "pug");`
 - `app.set("views", "path/to/views");`
- Finalement on appelle la méthode *render(template,params)* de l'objet *response*

Pug: Exemple

```
const { application } = require("express")
const express=require("express")
```

app.js

```
const app=express()
app.set("view engine","pug")
app.set("views","./templates")
app.use((req, res)=>{
  let data={
    name: "pug"
  }
  res.render("index",data)
})
app.listen(3000);
```

```
doctype html
html
  head
    title my #{name} test
  body
    h1 hi from #{name}!!!
```

templates/index.pug



localhost:3000

hi from pug!!!



view-source:http://localhost:3000/

```
1 <!DOCTYPE html ><html><head><title>my pug test </title></head><body> <h1>hi from pug!!! </h1></body></html>
```

Classes et ID

- Pour attribuer une classe à un élément html, on utilise le "."
 - `h1.titre ...` => `<h1 class="titre"> ... </h1>`
- Pour attribuer un ID à un élément html, on utilise le "#"
 - `h1#titre ...` => `<h1 id="titre"> ... </h1>`
- *Si on ne précise pas l'élément Html pug utilise la balise <div>*
 - `.titre ...` => `<div class="titre"> ... </div>`
 - `#titre ...` => `<div id="titre"> ... </div>`
- *Pour ajouter un attribut à un élément on utilise les parenthèses "(attribut=valeur)"*
 - `img.garde(src="logo.png", alt="mon logo")` => ``

Texte

- On peut ajouter un texte à une template par plusieurs façons:

```
p Pug rocks!
```

```
P
```

```
| You are logged in as
```

```
| user@example.com
```

```
P.
```

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor  
incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud  
exercitation ullamco laboris nisi ut aliquip ex ea commodo consequa
```

Commentaires

- On peut ajouter des commentaires à une template par plusieurs façons:

```
// My wonderful navbar
nav#navbar-default
// - My wonderful navbar (silent comment)
nav#navbar-default
//
    My wonderful navbar
    It is just so, awesome!
nav#navbar-default
nav#navbar-default // My wonderful navbar
```

Javascript dans pug

- Une caractéristique importante de pug est la possibilité d'exécuter du javascript dans ses templates.
- On peut insérer des variables, des objets, des boucles, des conditions ...ect
- Pour utiliser js dans pug il faut distinguer entre
 - **Buffered code** (le code tamponné) :
 - **Unbuffered code** (le code non tamponné)

unbuffered et buffered code

- unbuffered code (le code tamponné) :
 - commence par (-)
 - N'ajoute rien à la sortie, mais ses valeurs peuvent être utilisé dans le reste de la template
- buffered code (le code non tamponné)
 - Commence par (=)
 - Le javascript est évalué et rendu en sortie

```
//-unbuffered code
- const name = "said"

//- On peut maintenant utiliser name dans le reste de la template pug

//-buffered code
p= 'Hi my name is: ' + name
p= 'my age is: ' + 2*10 +'years old'

//-Pour des raison de securité le buffered code est échappé (escaped)
p= '<script>alert("Hi")</script>'

//- Donne en sortie: <p>&lt;script&gt;alert(&quot;Hi&quot;)&lt;/script&gt;</p>p>
```

Interpolation

- L'interpolation est le processus de remplacer des gabarits (placeholders) par des valeurs des expressions js.
- L'interpolation se fait par plusieurs méthodes:
 - Le buffered code (déjà vu)
 - L'utilisation de `#{expression js}`

```
- const name = "said"
p Hi #{name}
p Hi #{name.toUpperCase()}
p Hi #{name.charAt(0).toUpperCase() + name.slice(1)}
//on peut omettre #{ } pour affecter la valeur d'une var à un attribut d'un élément
img(src="portrait.png", alt=name)
```

Les boucles

- Pour parcourir les éléments d'un tableau dans pug on utilise le mot clé *each*

```
-const days=["lundi", "mardi", "mercredi", "jeudi", "vendredi",  
"vendredi", "samedi", "dimanche"]  
  
ol  
  
  each day in days  
    li= day  
  
  else  
    li le tableau est vide
```

Les boucles

- On peut aussi parcourir les éléments d'un objet

```
-  
  
const employee = {  
  'name': 'said',  
  'email': 'said@upm.ma',  
  'age': 21  
}  
  
ul  
  
each value, key in employee  
  li= `${key}: ${value}`  
  
each l'objet est {}
```

Conditionnels

- Les conditionnels offrent un moyen très pratique de rendre différents HTML en fonction du résultat d'une expression JavaScript :

```
-  
  
const student = {  
  'name': 'said',  
  'email': 'said@upm.ma',  
  'age':12  
}  
  
if student.age<11  
  p kid  
else if student.age<15  
  p junior  
else  
  p senior
```