

Laboratoire 2

Développement framework Côté serveur

Configuration du serveur **Node.js** et exploration

420-FCS-TT
420-4FS-TT

IMPORTANT !

Avant de commencer, écoutez la vidéo : <https://youtu.be/x4KPKcGr0hE>

2021-09-07 14:08

Objectifs du laboratoire

- Installer et configurer Express
- Gérer les routes avec Express
- Installer et configurer EJS
- Utiliser les vues partielles avec EJS
- Consolider son apprentissage versus l'exploration du laboratoire 1

Outils nécessaires :

- Logiciel d'édition *Visual Studio Code*
- *Node.js*



Par :
Fernand Bikatal Bi Tonye
ftonye@teccart.qc.ca

Correction/adaptation :
Marc Grenier
2020-01-13

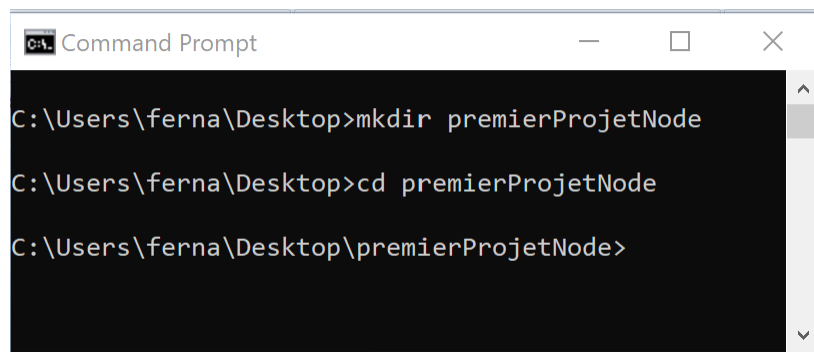
Table des matières

1. CRÉATION D'UN PROJET AVEC NODE	4
CRÉATION DU DOSSIER DE PROJET	4
INITIALISATION DU PROJET	4
INSPECTION ET MODIFICATION DU FICHIER JSON	7
2. UTILISATION D'UN FRAMEWORK AVEC NODE	8
INSTALLATION DE EXPRESS	8
3. UTILISATION D'UN « TEMPLATE » AVEC NODE	13
INTRODUCTION	13
INSTALLATION DE EJS	13
CONFIGURATION D'EXPRESS.....	14
CRÉATION D'UN TEMPLATE EJS.....	15
RÉPONDRE À UN GET AVEC UN TEMPLATE	15
4. TRAVAILLER AVEC DES FICHIERS STATIQUES	16
AJOUT DE FICHIERS « STATIC » AU « TEMPLATE »	16
ENVOYER DES PARAMÈTRES DU SERVEUR VERS LA VUE.....	ERREUR ! SIGNET NON DEFINI.
5. UTILISATION DES VUES PARTIELLES AVEC NODE	19
LES VUES PARTIELLES	19
6. EXÉCUTER DU JAVASCRIPT COTE SERVEUR AVEC NODE.JS	ERREUR ! SIGNET NON DEFINI.
PASSER UN TABLEAU DE DONNÉES COMPLEXES À UN TEMPLATE	ERREUR ! SIGNET NON DEFINI.

1. CRÉATION D'UN PROJET AVEC NODE

CRÉATION DU DOSSIER DE PROJET

Pour créer un projet Node, il faut tout d'abord créer un dossier qui contiendra les fichiers et dépendances de votre projet. À l'aide de l'invite de commande effectuez les manipulations suivantes :



```
Command Prompt

C:\Users\ferna\Desktop>mkdir premierProjetNode

C:\Users\ferna\Desktop>cd premierProjetNode

C:\Users\ferna\Desktop\premierProjetNode>
```

Le but étant de créer le dossier **premierProjetNode** sur le bureau de votre ordinateur et de naviguer dans ce dossier.

INITIALISATION DU PROJET

Une fois à l'intérieur du dossier, vous pouvez utiliser la commande **npm init**. Une série de questions vous sera alors posée, la première étant le nom de *package* que vous voulez utiliser.

```
npm
C:\Users\ferna\Desktop>mkdir premierProjetNode
C:\Users\ferna\Desktop>cd premierProjetNode
C:\Users\ferna\Desktop\premierProjetNode>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 json` 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: (premierprojetnode)
```

Tapez la touche **Enter**, un numéro de version vous sera proposé. Tapez à nouveau **Enter** et donnez la description, *Mon premier projet Node*, à votre projet.

```
npm
C:\Users\ferna\Desktop>mkdir premierProjetNode
C:\Users\ferna\Desktop>cd premierProjetNode
C:\Users\ferna\Desktop\premierProjetNode>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 json` for definitive documentation on these fields
and exactly what they do.

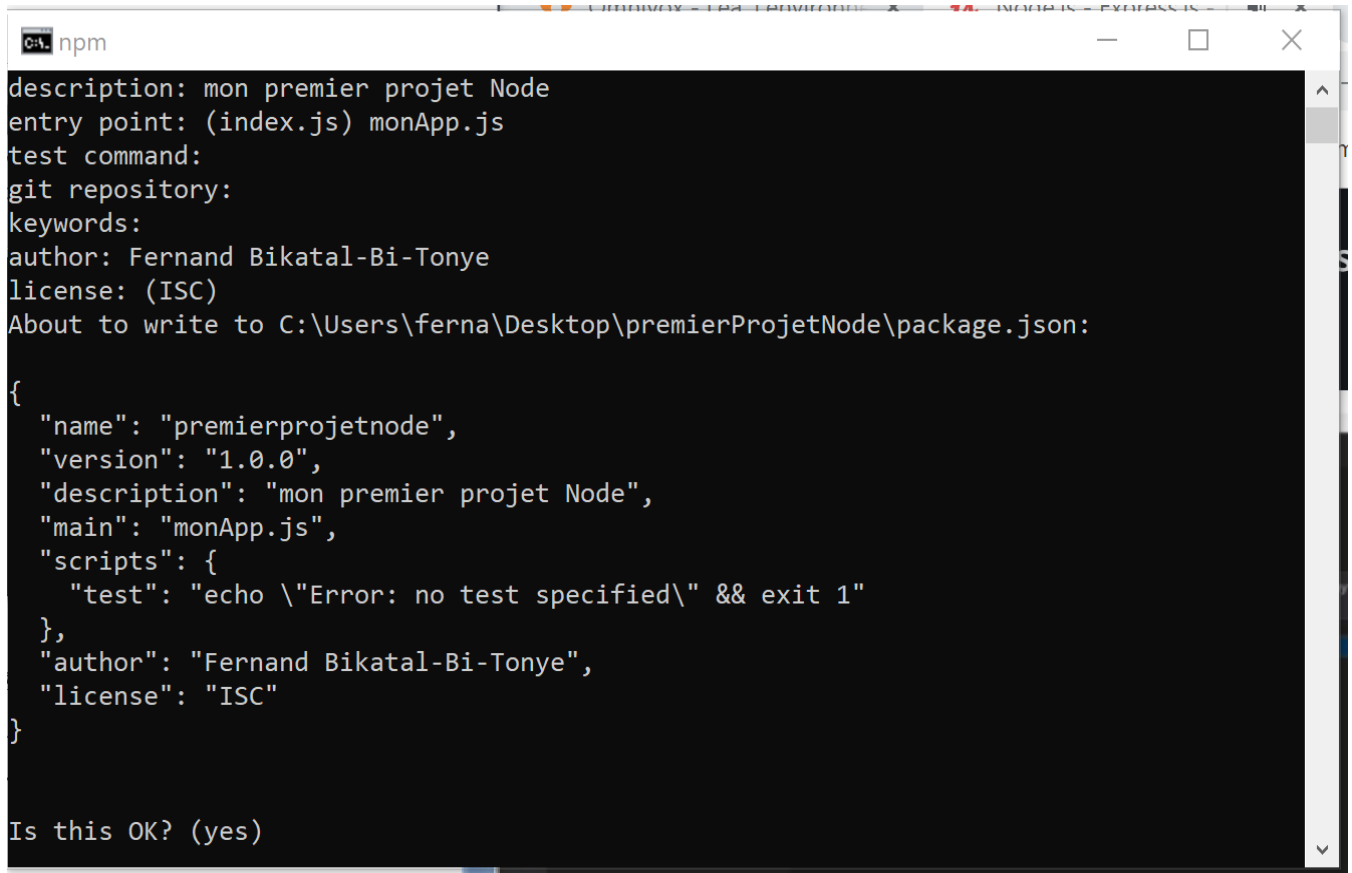
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (premierprojetnode)
version: (1.0.0)
description: mon premier projet Node
```

Tapez **Enter** et un *entry point* vous sera proposé pour votre application. Inscrivez *monApp.js* et tapez **Enter** :

```
npm
Press ^C at any time to quit.
package name: (premierprojetnode)
version: (1.0.0)
description: mon premier projet Node
entry point: (index.js) monApp.js
```

Entrez ensuite votre nom comme auteur et laissez les autres champs libres pour le moment. Un aperçu du fichier descriptif JSON de votre application vous sera présenté.



```
C:\> npm

description: mon premier projet Node
entry point: (index.js) monApp.js
test command:
git repository:
keywords:
author: Fernand Bikatal-Bi-Tonye
license: (ISC)
About to write to C:\Users\ferna\Desktop\premierProjetNode\package.json:

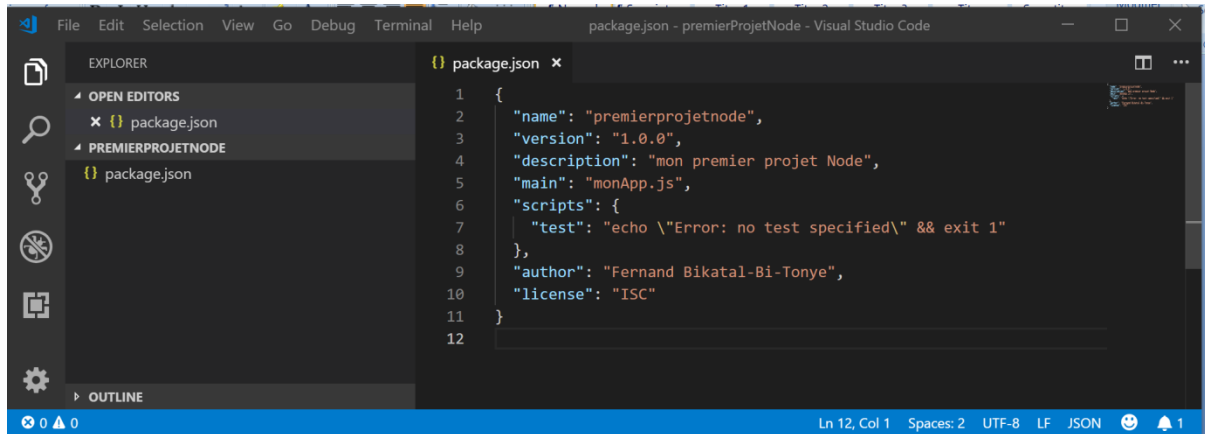
{
  "name": "premierprojetnode",
  "version": "1.0.0",
  "description": "mon premier projet Node",
  "main": "monApp.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Fernand Bikatal-Bi-Tonye",
  "license": "ISC"
}

Is this OK? (yes)
```

Tapez **Enter** pour confirmer.

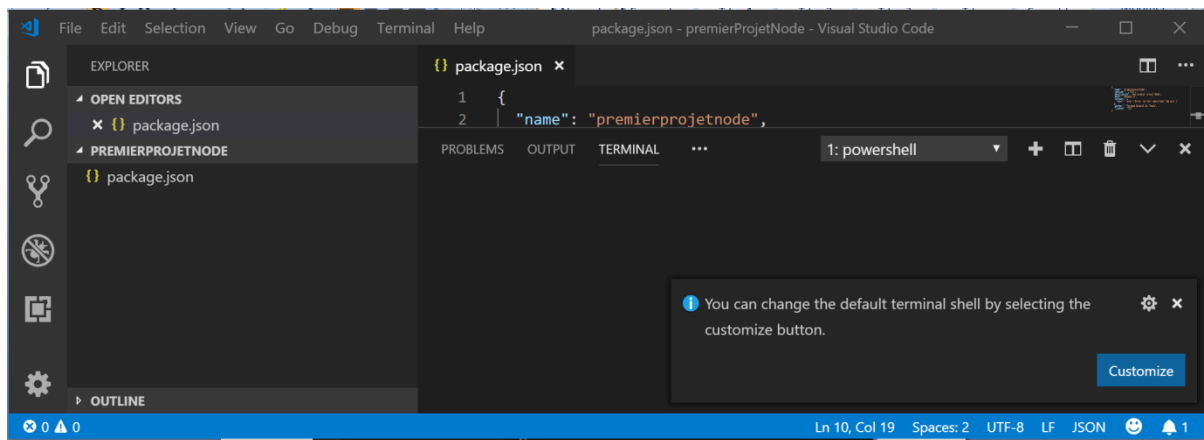
INSPECTION ET MODIFICATION DU FICHER JSON

Il existe plusieurs éditeurs de texte nous permettant de modifier des fichiers JSON, mais pour les besoins de ce cours, nous allons utiliser **Visual Studio code**. Ouvrez **Visual Studio Code**, choisissez, **File** ➔ **Open Folder** et sélectionnez votre répertoire de projet.



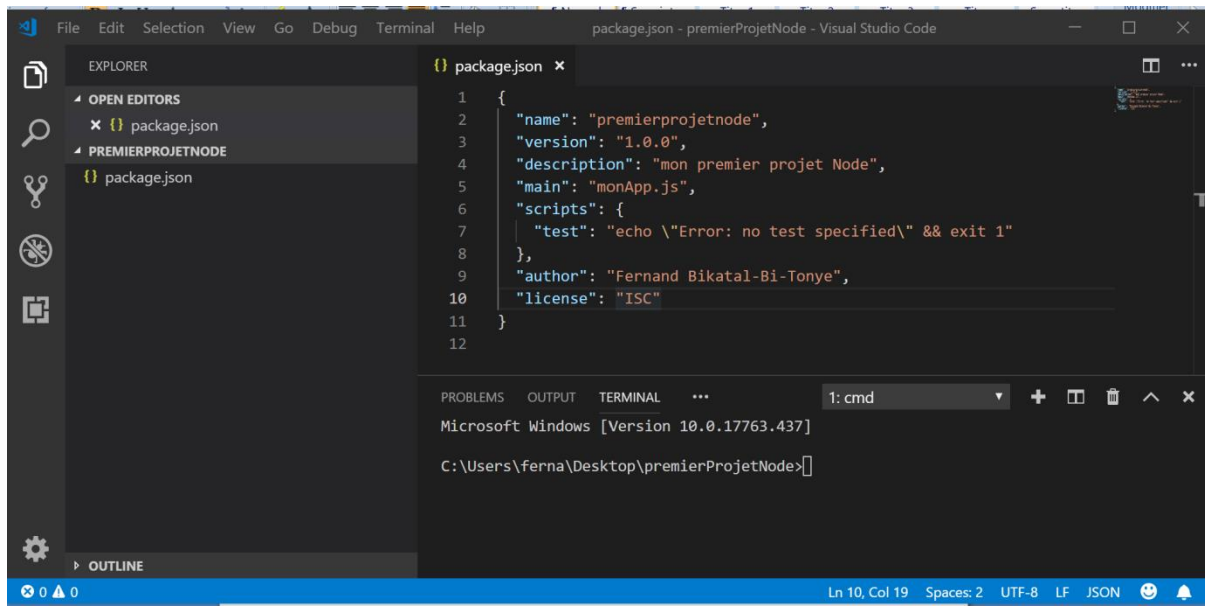
Un des avantages de cet éditeur de texte, est qu'il nous permet d'interagir directement avec l'invite de commande. Sélectionnez :

Terminal ➔ **New Terminal**



Si l'invite de commande **Powershell** apparaît, modifiez-le en appuyant sur le menu déroulant. Choisissez l'option **Command Prompt**.

(Lors de vos prochains projets, vous pourrez créer le dossier dans l'Explorateur Windows, ouvrir le dossier dans Visual Studio et faire le **npm init** à même le terminal).



2. Utilisation d'un framework avec Node

INSTALLATION DE EXPRESS

À l'aide de **NPM**, un gestionnaire de package, nous allons installer **Express**, un *framework* pour le développement d'application Node.js, avec la commande : **npm install express --g**
Nous obtiendrons le résultat suivant :

```
npm WARN premierprojetnode@1.0.0 No repository field.
+ express@4.16.4
updated 1 package and audited 121 packages in 1.441s
found 0 vulnerabilities
```


Le module est installé, mais n'est pas encore dans les dépendances du projet dans le fichier, **package.json** :

```
{ } package.json ●
1  {
2    "name": "testoptionsgets",
3    "version": "1.0.0",
4    "description": "Test",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Marc Grenier",
10   "license": "ISC"
11 }
12
```

Attention, fermez le fichier **package.json** avant de continuer.

Faites

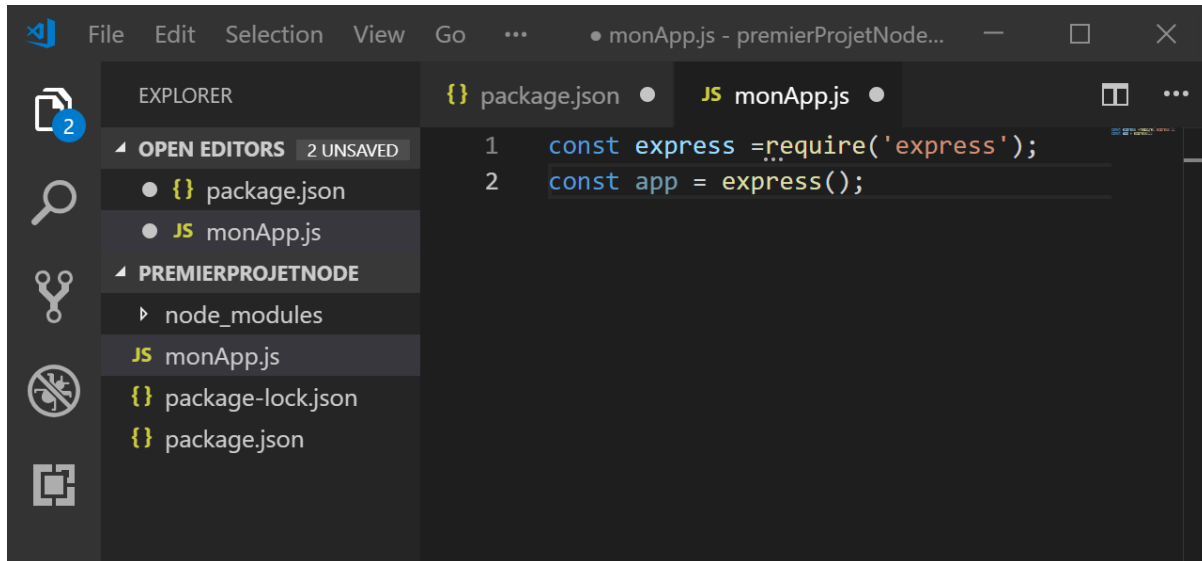
npm install express --save

Le module sera maintenant visible dans le fichier, **package.json** :

```
{ } package.json > ...
1  {
2    "name": "labo7",
3    "version": "1.0.0",
4    "description": "test du labo7",
5    "main": "MonApp.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1"
8    },
9    "author": "Marc",
10   "license": "ISC",
11   "dependencies": {
12     "express": "^4.17.1"
13   }
14 }
15
```

CRÉATION DU FICHIER **monApp.js**

À l'aide de **Visual Studio Code**, créez le fichier **monApp.js** et ajoutez-y les deux lignes de code suivantes :



La première ligne de code permet de récupérer le module **express** que nous avons installé avec la commande **npm install express**

La deuxième ligne de code permet d'utiliser ce module pour créer notre serveur référencé dans la variable **app**.

Spécifions maintenant le routage de base de notre site. Les fureteurs font des requêtes **get** par défaut, quand vous demandez un site web. Si notre utilisateur demande notre site web, soit **localhost :3000**, par une requête **get**, quelle ressource lui sera retournée ? Voir ici-bas :

```
{} package.json JS monApp.js ×
Labo 7 > labo7Node > JS monApp.js > ...
1  const express = require('express') //utilise express
2  const app = express() //crée une instance de express dans app
3
4  app.get('/', function(req, res) { //req demande la page web
5      res.send('Bonjour le monde!'); //res.send renvoie Bonjour le monde dans res
6  });
7
8  app.listen(3000, function() { //le serveur attend les requête sur le port 3000
9      console.log("j'écoute le port 3000!");
10 });
```

Les deux premières lignes importent (**require()**) le module Express et créent une application Express (**var app = express();**). Cet objet, qui est traditionnellement nommé **app**, dispose de méthodes pour acheminer les requêtes http.

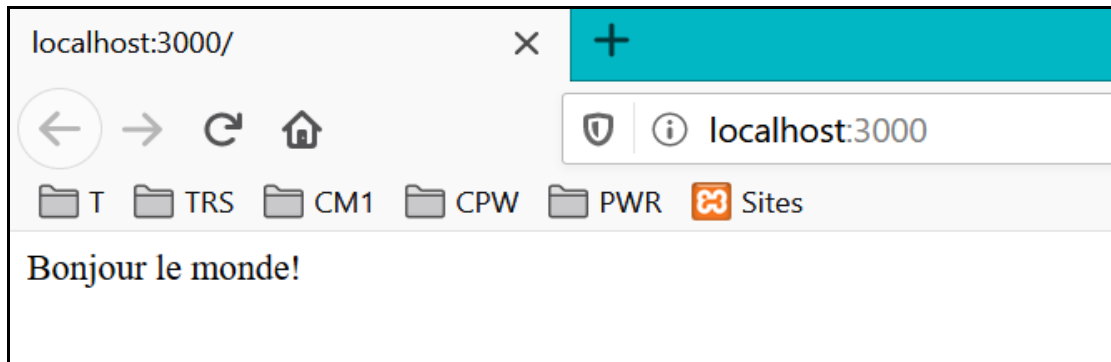
La partie centrale du code (commençant par **app.get**) montre une définition de route. La méthode **app.get()** spécifie une fonction de rappel (**callback**) qui sera invoquée chaque fois qu'il y a une requête HTTP GET avec le chemin ('/') relatif à la racine du site. La fonction de rappel prend **req** et **res** comme arguments. **res.send** envoie simplement en réponse la chaîne "Bonjour le monde !"

Le dernier bloc démarre le serveur sur le port '3000' et envoie un commentaire sur la console.

Faites démarrer le serveur ainsi avec la commande, **node monApp.js** :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
C:\Users\ferna\Desktop\premierProjetNode>node monApp.js
j'écoute le port 3000
█
```

Avec le serveur en cours d'exécution, vous pouvez accéder à **localhost:3000** dans votre **Mozilla**, pour voir l'exemple de réponse renvoyée.



Pour arrêter le serveur, il suffit d'utiliser la combinaison des touches **CTRL + C**.

Utilisez maintenant la notation **abrégée** de la fonction de *callback* pour utiliser un code simplifié :

```
Labo 7 > labo7Node > JS monApp.js > ...
1  const express = require('express') //utilise express
2  const app = express() //crée une instance de express dans app
3
4  app.get('/', (req, res)=> { //req demande la page web
5      res.send('Bonjour le monde!'); //res.send renvoie Bonjour le monde dans res
6  });
7
8  app.listen(3000, ()=> { //le serveur attend les requête sur le port 3000
9      console.log("j'écoute le port 3000!");
10 });
```

Il ne nous reste plus qu'à redémarrer le serveur et à réessayer.

3. Utilisation d'un « template » avec Node

INTRODUCTION

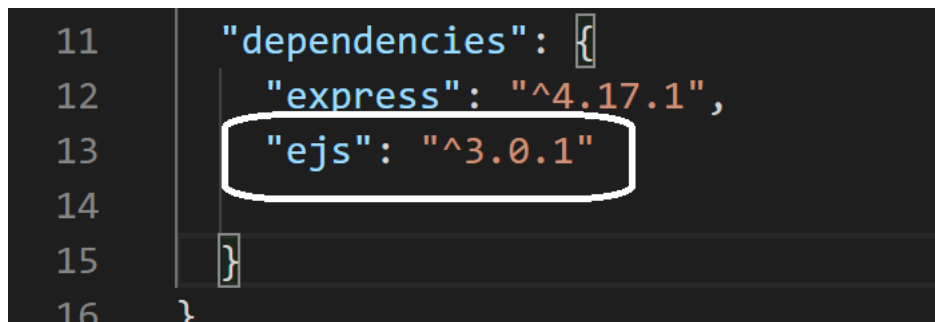
Présentement, notre application **Node.js** est en mesure de servir des requêtes de type **http get** en retournant au client des chaînes de caractères (strings) ou des balises html. Dans ce document, nous allons présenter un **Template Engine** nommé **EJS** qui permettra à notre serveur de sélectionner une vue (**view**) et d'envoyer les informations devant être présentées dans un document html.

INSTALLATION DE EJS

Attention, si le fichier, **package.json**, est ouvert, fermez-le avant de continuer.

- 1) Pour installer EJS, il faut naviguer dans le répertoire de votre projet et tapez la commande
npm install ejs --g
- 2) Pour ajouter EJS dans votre fichier, **package.json**, tapez la commande :
npm install ejs --save

Assurez-vous de la présence de EJS dans le fichier **package.json** avant de continuer :



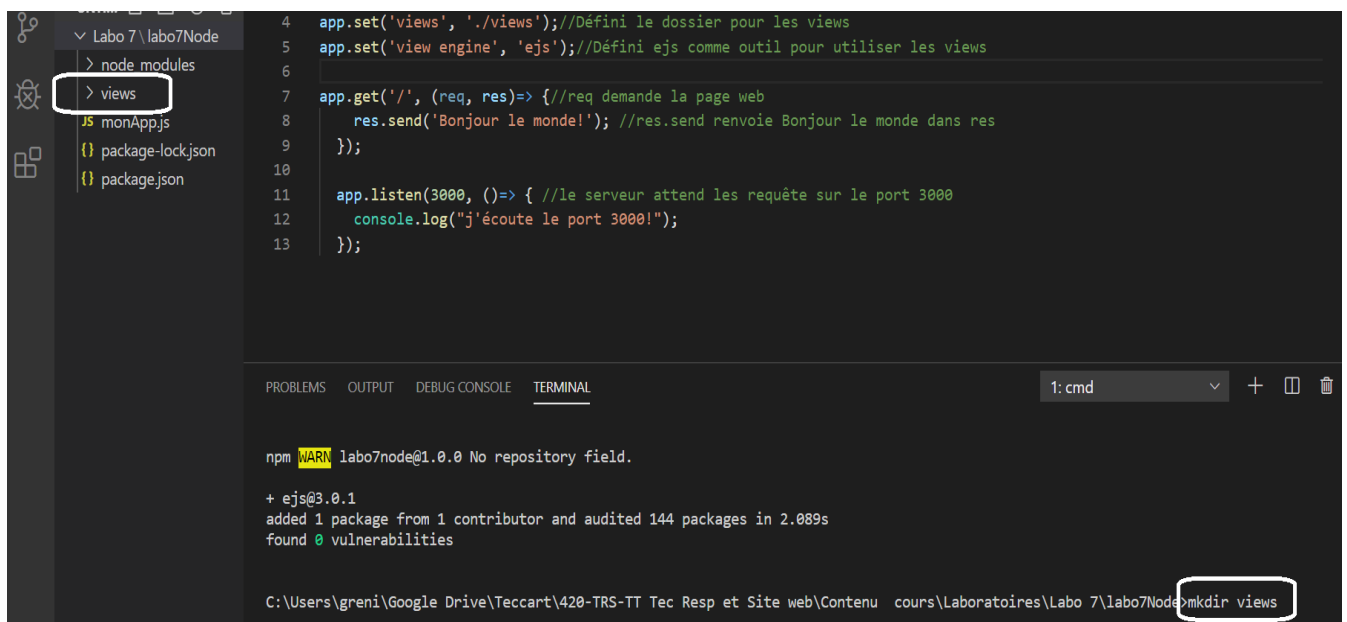
```
11  "dependencies": {  
12    "express": "^4.17.1",  
13    "ejs": "^3.0.1"  
14  },  
15  
16 }
```

CONFIGURATION D'EXPRESS

Une fois ce module installé, il vous faut faire quelques configurations supplémentaires au niveau de notre serveur **Node.js**. Il faudra spécifier dans quel répertoire se trouveront nos vues et indiquer le **Template Engine** devant être utilisé par le serveur :

```
1  const express = require('express') //utilise express
2  const app = express() //crée une instance de express dans app
3
4  app.set('views', './views');//Défini le dossier pour les views
5  app.set('view engine', 'ejs');//Défini ejs comme outil pour utiliser les views
6
7  app.get('/', (req, res)=> { //req demande la page web
8      res.send('Bonjour le monde!'); //res.send renvoie Bonjour le monde dans res
9  });
10
11  app.listen(3000, ()=> { //le serveur attend les requête sur le port 3000
12      console.log("j'écoute le port 3000!");
13  });
```

Remarquez qu'en plus des lignes 4 et 5 dans le fichier **monApp.js**, nous devons aussi ajouter un dossier **views** dans le dossier racine du projet et ce, à l'aide de la commande **mkdir views** :



The screenshot shows the Visual Studio Code interface. On the left, the file explorer shows the project structure with a folder named 'views' highlighted. The code editor in the center displays the same JavaScript code as the previous block. At the bottom, the terminal window shows the output of running 'npm install' and the command 'mkdir views' being executed in the project directory.

```
npm WARN labo7node@1.0.0 No repository field.
+ ejs@3.0.1
added 1 package from 1 contributor and audited 144 packages in 2.089s
found 0 vulnerabilities

C:\Users\greni\Google Drive\Teccart\420-TRS-TT Tec Resp et Site web\Contenu cours\Laboratoires\Labo 7\labo7Node>mkdir views
```

CRÉATION D'UN TEMPLATE EJS

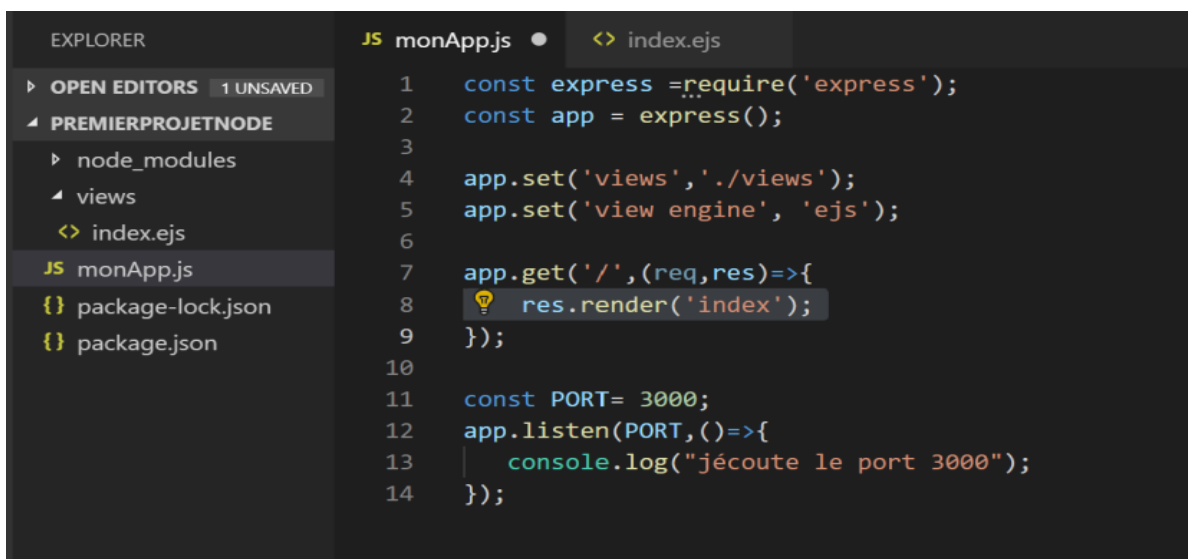
Un *template* de type **ejs** est en réalité un fichier contenant du code html et des balises de script particulières permettant de récupérer et d'afficher les informations provenant du serveur **Node.js**.

Créons **index.ejs** dans le dossier **views**.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4  |   <meta charset="utf-8">
5  <title> test EJS</title>
6
7  </head>
8  <body>
9  |
10 |   <h2>Bonjour EJS</h2>
11 |
12 |
13 </body>
14 </html>
```

RÉPONDRE À UN GET AVEC UN TEMPLATE

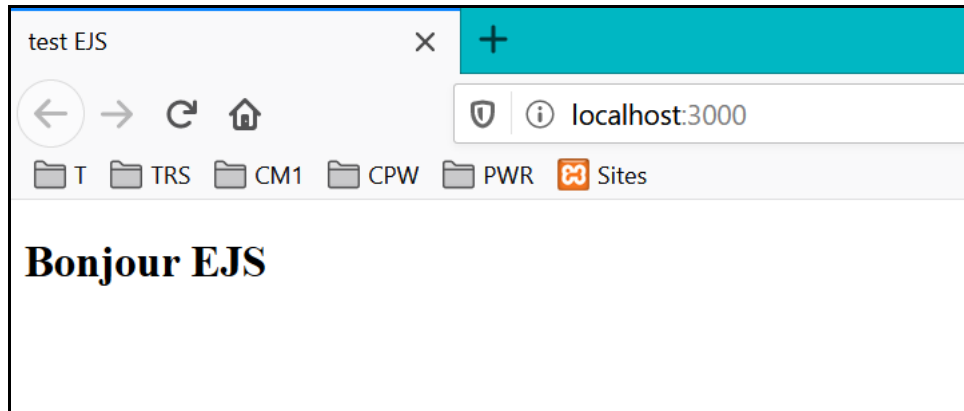
Lorsque nous voulons que le serveur utilise un *template* pour effectuer sa réponse au client, il doit utiliser la méthode **render()** au lieu de **send()**.



```
EXPLORER
└─ OPEN EDITORS 1 UNSAVED
  └─ PREMIERPROJETNODE
    └─ node_modules
      └─ views
        └─ index.ejs
      JS monApp.js
      {} package-lock.json
      {} package.json

JS monApp.js
1  const express = require('express');
2  const app = express();
3
4  app.set('views', './views');
5  app.set('view engine', 'ejs');
6
7  app.get('/', (req, res) => {
8    res.render('index');
9  });
10
11 const PORT = 3000;
12 app.listen(PORT, () => {
13   console.log("j'écoute le port 3000");
14 });
```

Si on lance à nouveau le serveur, voici le résultat que nous devrions obtenir :

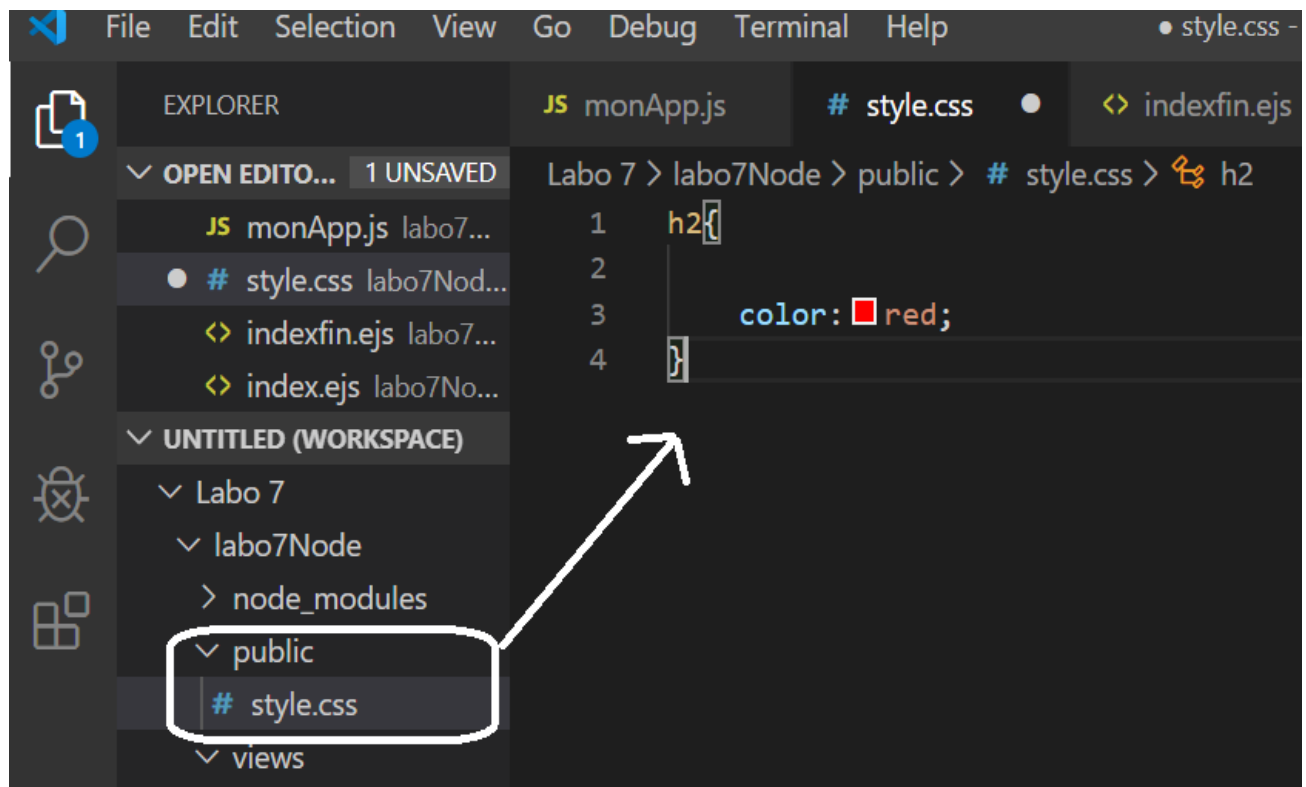


4. Travailler avec des fichiers statiques

AJOUT DE FICHIERS « STATIC » AU « TEMPLATE »

Pour être en mesure de créer une application web digne de ce nom, il vous faut être en mesure d'exploiter des fichiers statiques comme des feuilles de style ou des scripts. Le serveur **Node.js** met à notre disposition des *middlewares* nous permettant de charger les fichiers statiques à l'aide de la méthode **use**.

Ajoutez un répertoire nommé **public** au projet avec un fichier **CSS** à l'intérieur. Assurez-vous que ce fichier **CSS** permette de mettre la balise **<h2>** en rouge :



Assurez-vous que le répertoire **public** sera **utilisable** (donc **app.use**) dans le projet

```
1 const express = require('express') //utilise express
2 const app = express() //crée une instance de express dans app
3
4 app.set('views', './views');//Défini le dossier pour les views
5 app.set('view engine', 'ejs');//Défini ejs comme outil pour utiliser les views
6 app.use('/public', express.static('public'));// permet l'utilisation de fichiers statiques dans public
7
8 app.get('/', (req, res) => {
9   res.render('index.ejs');
10 });
11
12
13
14 app.listen(3000, ()=> { //le serveur attend les requête sur le port 3000
15   console.log("j'écoute le port 3000!");
16 });
```

N'oubliez pas d'ajouter le lien vers le fichier CSS dans *index.ejs* :

```
<> index.ejs ×
views > <> index.ejs > html > body > h2
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5 <meta charset="utf-8">
6 <title> test EJS </title>
7 <link rel = "stylesheet" type = "text/css" href=" ../public/style.css">
8 </head>
```

Rechargez la page pour être certain que cela fonctionne :

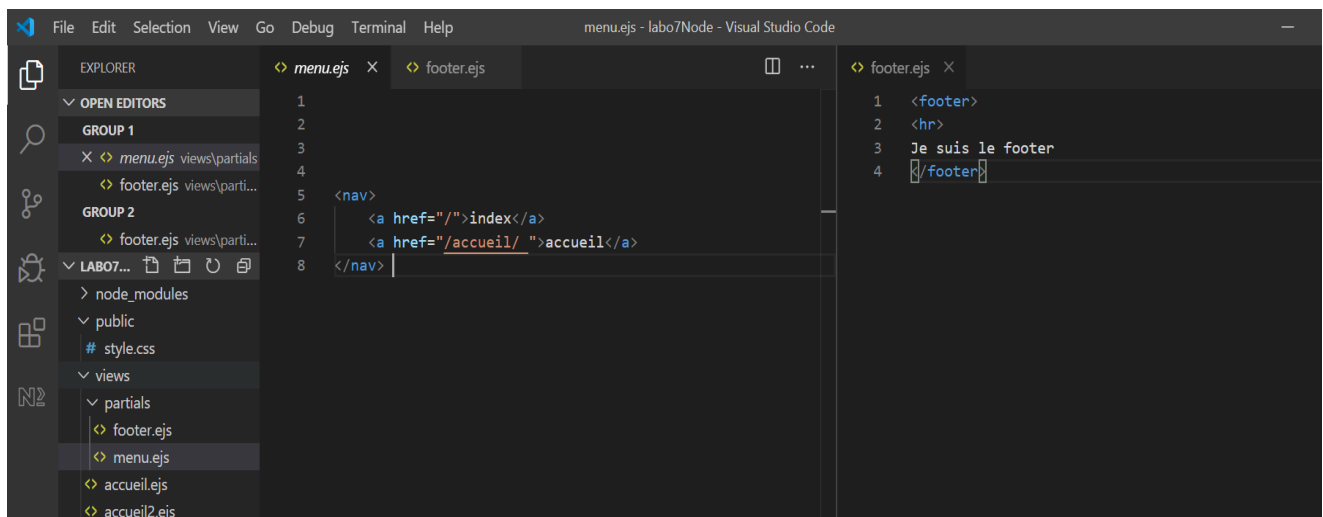


5. Utilisation des vues partielles avec Node

LES VUES PARTIELLES

Plusieurs technologies web côté serveur permettent de générer une **vue** et ce en assemblant des morceaux de plusieurs autres **vues**. Nous allons donc présenter ici le concept des **partials**.

Nous allons d'abord créer les fichiers, **menu.ejs** et **footer.ejs** qui fourniront les éléments partiels que nous ajouterons aux fichiers principaux (**accueil.ejs** et **index.ejs**) plus tard :



Remarquez bien les changements qui ont été apporté à l'arborescence de projet, avec l'ajout du dossier **partials** sous le dossier **views**.

Nous allons maintenant récupérer ces deux **partials**, dans un nouveau fichier, **accueil.ejs** :

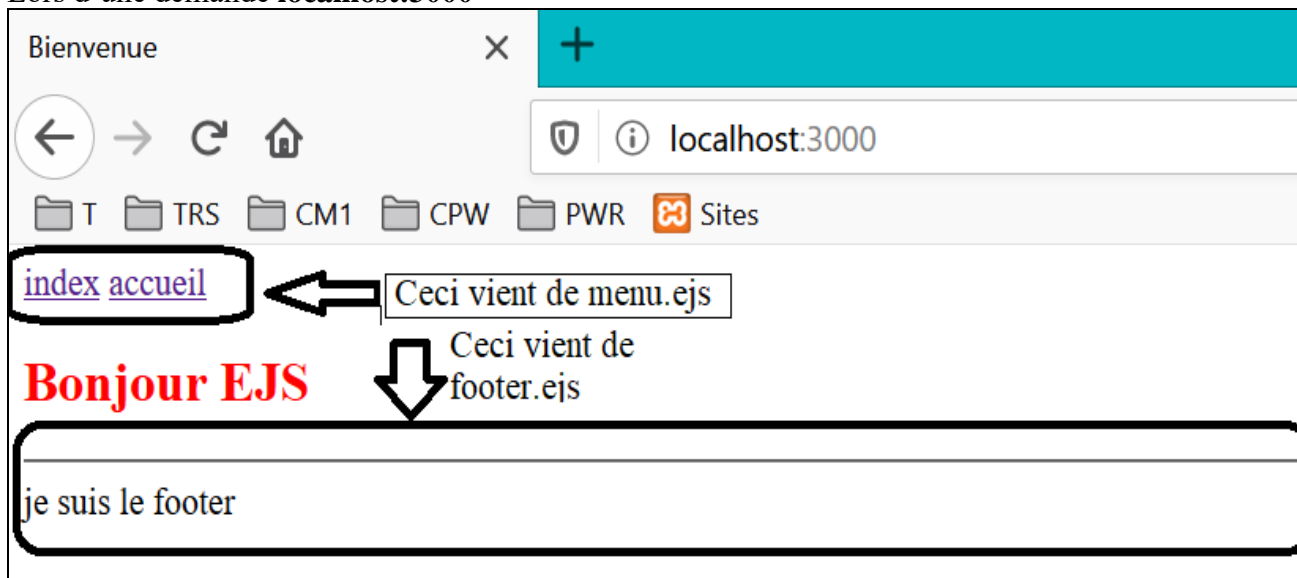
```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <meta charset="utf-8">
5   <title> <%=pageTitle%> </title>
6
7   <link rel = "stylesheet" type ="text/css" href=" ../public/style.css">
8
9
10 </head>
11 <body>
12   <%-include('./partials/menu')%>
13
14   <div>Bienvenue Marc Grenier</div>
15   <%-include('./partials/footer')%>
16
17 </body>
18 </html>
```

ATTENTION ! Ne pas mettre d'espaces, le code doit être identique pour les *include*.

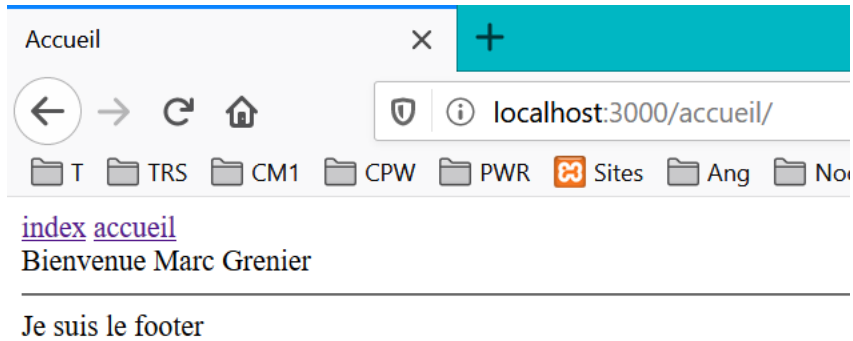
Faites les deux mêmes ajouts dans le fichier **index.ejs** !

Ainsi :

Lors d'une demande **localhost:3000**



Lors d'une demande **localhost:3000/accueil/**



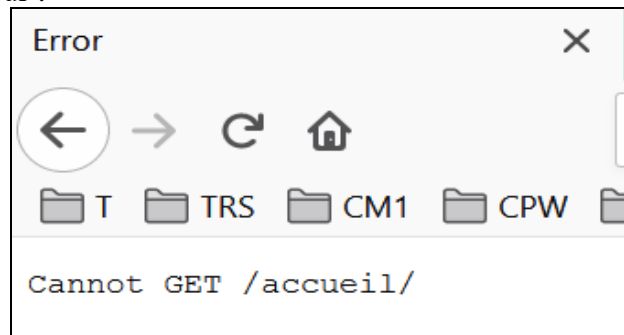
Cependant, lors d'un clic sur le lien, accueil, ou lors d'une demande **localhost :3000/accueil**, un message d'erreur s'affiche, car la route

```
app.get('/acceuil/',(req,res)=>{
```

prévu pour le lien de la vue partielle (**menu.ejs**), n'existe pas encore :



Voici le message d'erreur :

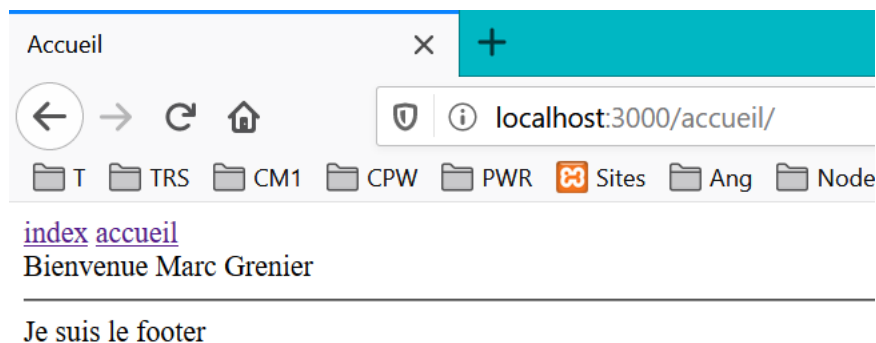


Ajoutez la route suivante pour régler le problème :

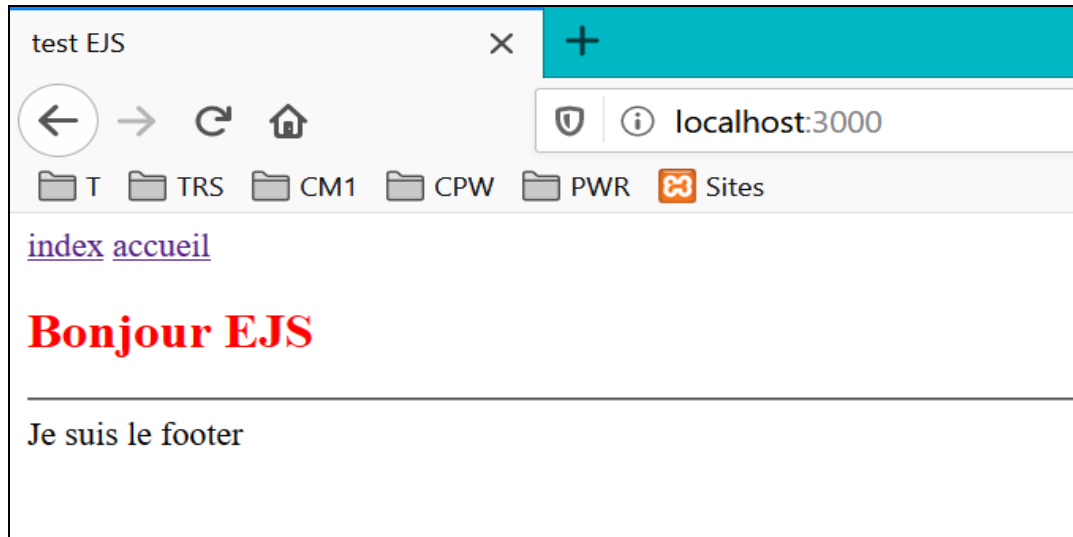
```
6
7 app.get('/accueil/',(req,res)=>{
8
9   res.render('accueil',{pageTitle:'Accueil'});
10 });
11
```

NOTE : Remarquez que les extensions **.ejs** ont été retirées des routes, car elles sont les extensions par défaut.

Lors d'un clic sur **accueil**, vous devriez maintenant obtenir la page suivante :



Lors d'un clic sur index :



-FIN-