

## TME 6 – Node et Express

### TME 6 – Node et Express

Ce TME a pour objectif d'aborder le fonctionnement de base d'Express :

- installation
- routage
- réponse à des requêtes basiques
- gestion du JSON

#### 5.1 Installation

Nous n'allons pas faire de React dans le cadre de ce TME, car nous abordons la partie serveur. Pour cela, il faut mettre en place le code qui permettra de répondre aux requêtes HTTP.

1. créez un répertoire nommé TME6
2. **placez-vous dans ce répertoire (cd TME6)**
3. tapez `npm init` pour initialiser un nouveau projet dans ce répertoire
4. tapez `npm install express`. Si l'installation ne se passe pas correctement, supprimez d'autres répertoires `node_modules` créés par ailleurs (par exemple lors du TME précédent). Ne vous inquiétez pas, l'essentiel est de ne jamais supprimer le fichier `package.json` qui contient tout le nécessaire pour installer un projet et re-télécharger les modules node nécessaires.

Vous pouvez regarder rapidement ce qui a été installé dans le répertoire `node_modules`.

Pour plus de facilité, nous allons aussi installer le module `nodemon`. Ce module permet de ne pas avoir à *tuer* node à chaque modification d'un fichier du projet en redémarrant automatiquement le serveur.

1. tapez `npm install nodemon`
2. dans le fichier `package.json`, ajoutez la ligne `"start": "nodemon index.js"`, (sans oublier la virgule, c'est au format JSON !) juste avant la ligne commençant par `"test"`
3. créez un fichier `index.js`, contenant la ligne `require('./app/index1.js');`
4. créez un fichier `index1.js` dans le répertoire `app`. C'est dans ce fichier que vous allez placer le code du TME

Cela donne la possibilité de conserver ce fichier pour garder une trace des exercices successifs, et de reprendre le TME dans un autre fichier, par exemple `app/index2.js`, en écrivant dans `index.js` la ligne `require('./app/index2.js');`.

#### 5.2 Site de base

Dans ce premier exercice, nous n'utiliserons que des méthodes GET, ce qui permet de visualiser le résultant dans une fenêtre de navigateur.

1. Affichez le texte "Tout va à merveille" en réponse à une requête sur le port 8000 de localhost. Précisez l'encodage de la réponse.
2. Affichez le texte "Cette page n'existe pas" en cas d'erreur 404.
3. Affichez le texte "Tout va à merveille pour cette page 1" sur la page localhost :8000/test1
4. Affichez le texte "Tout va à merveille pour cette page xxx" sur la page localhost :8000/essaixxx, où xxx désigne une chaîne de caractères quelconque.

Importez maintenant le module `path` (en ajoutant `const path = require('path');`). Créez dans le répertoire `app` un répertoire `public` dans lequel vous placerez le fichier `test.html` suivant :

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="utf-8" />
    <title>Fichier statique</title>
</head>
<body>
    <p>Page test</p>
</body>
</html>
```

En vous servant notamment de la documentation de la méthode `sendFile` et de cette ressource, définissez une route permettant d'envoyer ce fichier index.html en réponse à un GET de l'URL `http://localhost:8000/test.html`.

```
./index.js

require('./app/index1.js')

./app/index.js

1 const express = require('express');
2 const path = require('path');

3 const app= express();
4 const port=8000;

5 app.get('/', (req, res) => {
6     res.setHeader('Content-Type', 'text/plain;charset=UTF-8');
7     res.send('Tout va à merveille');
8 }
9 .get('/test1', (req, res) => {
10     res.setHeader('Content-Type', 'text/plain;charset=UTF-8');
11     res.end('Tout marche à merveille pour cette page 1');
12 }
13 .get('/essai:num', (req, res) => {
14     res.setHeader('Content-Type', 'text/plain;charset=UTF-8');
15     res.end('Tout marche à merveille pour cette page '+req.params.num);
16 }
17 .get('/test.html', (req, res) => {
18     res.sendFile(path.join(__dirname, '/public/test.html'));
19 }
20 .use(function (req, res, next) {
21     res.setHeader('Content-Type', 'text/plain;charset=UTF-8');
22     res.status(404).send("Cette page n'existe pas.");
23 });
24

25 app.listen(port, function() {
26     console.log('Le serveur écoute le port '+port);
27 });
28 }
```

### 5.3 Utilisation de Router et de static

La situation précédente a comme inconvénient d'exposer les chemins sur le serveur pour accéder aux fichiers statiques. Il vaut mieux utiliser le `middleware static`.

- utilisez `Router` pour réécrire les routes précédentes
- en consultant la documentation d'`Express`, utilisez le `middleware static` pour éviter d'avoir à expliciter le chemin lors du routage.

```
./app/index.js
```

```

1 const express = require('express');
2 const path = require('path');
3
4 const router = express.Router();
5
6 const app = express();
7 const port = 8000;
8
9 app.use(express.static(path.join(__dirname, 'public')));
10
11 router.get('/', (req, res) => {
12     res.setHeader('Content-Type', 'text/plain; charset=UTF-8');
13     res.send('Tout va à merveille');
14 })
15 .get('/test1', (req, res) => {
16     res.setHeader('Content-Type', 'text/plain; charset=UTF-8');
17     res.end('Tout marche à merveille pour cette page 1');
18 })
19 .get('/essai:num', (req, res) => {
20     res.setHeader('Content-Type', 'text/plain; charset=UTF-8');
21     res.end('Tout marche à merveille pour cette page ' + req.params.num);
22 })
23 .get('/test.html', (req, res) => {
24     res.sendFile('test.html');
25 })
26 .use(function (req, res, next) {
27     res.setHeader('Content-Type', 'text/plain; charset=UTF-8');
28     res.status(404).send("Cette page n'existe pas.");
29 });
30
31 app.use('/', router);
32
33 app.listen(port, function() {
34     console.log('Le serveur écoute le port ' + port);
35 });

```

## 5.4 Réception et décodage d'une requête

En vous inspirant des exemples donnés en cours, mettre en place un serveur écoutant le port 8000, et répondant à une requête POST :

- postée à l'URL `http://localhost:8000/api`
- possédant un body en JSON valant `{"name": "test"}`

Le résultat est l'affichage dans la console du serveur du contenu de la propriété `name`

Tester la requête avec Postman. Que faut-il ajouter aux instructions précédentes ? Pourquoi ?

Complétez la requête en envoyant cette fois-ci dans le body

```
{
    "name": "test",
    "nb1": 3,
    "nb2": 5
}
```

À l'aide de la méthode `json` de l'objet réponse `res`, renvoyer au client le JSON

```
{
    "addition": 8,
    "prop": "hello"
}
```

où `addition` est calculé à partir de `nb1` et `nb2`.

Ne pas oublier d'envoyer `end()` pour que le client n'attende pas indéfiniment la fin de la réponse.

```

1 const express = require ('express');
2
3 const router = express.Router();
4
5 const app = express();
6 const port = 8000;
7
8 router.use(express.json());
9
10 router.get('/', (req, res) => {
11     res.setHeader('Content-Type', 'text/plain;charset=UTF-8');
12     res.send("Serveur à l'écoute");
13 })
14 .post('/api', (req, res) =>{
15     console.log(req.body.name);
16     res.json({addition: Number(req.body.nb1)+Number(req.body.nb2),prop: "hello"});
17     res.end();
18 })
19 .use(function (req, res, next) {
20     res.setHeader('Content-Type', 'text/plain;charset=UTF-8');
21     res.status(404).send("Cette page n'existe pas.");
22 });
23
24 app.use('/', router);
25
26 app.listen(port, function() {
27     console.log('Le serveur écoute le port '+port);
28 });

```

## 5.5 Gestion des erreurs de requête

Complétez le code précédent pour :

- renvoyer le code d'erreur 400 et le message "Requête invalide : name nécessaire" si la propriété `name` n'est pas indiquée dans le body de la requête
- renvoyer le code d'erreur 400 et le message "Requête invalide : nb1 et nb2 doivent être des nombres" si `nb1` ou `nb2` n'est pas un nombre

```

1 const express = require ('express');
2
3 const router = express.Router();
4
5 const app = express();
6 const port = 8000;
7
8 router.use(express.json());
9
10 router.get('/', (req, res) => {
11     res.setHeader('Content-Type', 'text/plain;charset=UTF-8');
12     res.send("Serveur à l'écoute");
13 })
14 .post('/api', (req, res) =>{
15     // Pour plus de lisibilité, on peut aussi définir const {name, nb1, nb2}=req.body;
16     try {
17         if (!req.body.name ) {
18             res.status(400).json({
19                 status: 400,

```

```

20             "message": "Requête invalide : name nécessaire"
21         });
22         return;
23     }
24     if (!(typeof req.body.nb1) == 'number') || !(typeof req.body.nb2) == 'number')) {
25         res.status(400).json({
26             status: 400,
27             "message": "Requête invalide : nb1 et nb2 doivent être des nombres"
28         });
29         return;
30     }
31     console.log(req.body.name);
32     res.json({addition: Number(req.body.nb1)+Number(req.body.nb2),prop: "hello"});
33     res.end();
34 }
35 catch (e) {
36     // Toute autre erreur
37     res.status(500).json({
38         status: 500,
39         message: "erreur interne",
40         details: (e || "Erreur inconnue").toString()
41     });
42 }
43 }
44 .use(function (req, res, next) {
45     res.setHeader('Content-Type', 'text/plain; charset=UTF-8');
46     res.status(404).send("Cette page n'existe pas.");
47 });
48
49 app.use('/', router);
50
51 app.listen(port, function() {
52     console.log('Le serveur écoute le port '+port);
53 });

```

## 5.6 Utilisation d'une session

Nous allons voir comment utiliser le *middleware* express-session.

1. installez le module express-session
2. en vous inspirant de [cet exemple de base](#), testez si une variable de session `user` est définie.
  - Si ce n'est pas le cas, l'initialiser avec par exemple votre prénom, et afficher dans le navigateur un message de bienvenue.
  - Si c'est le cas, afficher "Déjà de retour," puis la variable de session dans le navigateur.

```

1 const express = require('express')
2 const session = require('express-session')
3
4 const app = express()
5
6 app.use(session({
7   secret: 'monmotdepassesecret', // phrase secrète
8   resave: false, // don't save session if unmodified
9   saveUninitialized: false // pas de création de session tant que rien n'est sauvegardé
10 }))
11
12 app.get('/', (req, res) => {
13   if (req.session.user) {

```

```
14     res.setHeader('Content-Type', 'text/plain; charset=UTF-8');
15     res.status(200);
16     res.write('Déjà de retour, '+req.session.user+"?")
17     res.end()
18 } else {
19     req.session.user = "toi"
20     res.setHeader('Content-Type', 'text/plain; charset=UTF-8');
21     res.status(200);
22     res.end("Bienvenue")
23 }
24
25 app.listen(3000)
```