TP4 : Initiation au framework Expres.js

Dans cet exemple, vous aller expérimenter comment créer un serveur Web Express.js.

- Avant de commencer ce tp, il est recommandé de créer un nouveau dossier et l'ouvrir dans Visual Studio Code.
- Une fois le dossier créé, via le terminal de VS Code, initialiser votre projet en lançant l'instruction suivante : npm init
- Ouvrir le dossier et examiner le contenu du fichier «*package.json*», à quoi sert ce fichier ?

1. Créer un serveur web à l'aide du module express

Dans ce qui suit, nous allons créer un serveur Web qui répond sur la route "/" en utilisant le Framework web Express.js.

- Toujours dans le même dossier, installer le module express en utilisant la commande npm install express (Attention ! Vous aurez besoin d'une connexion internet)
- A votre avis pourquoi faut-il privilégier une installation locale du module express?
- Vérifier l'arborescence de votre dossier, que constatez-vous ?
- Quelle la version du module express installée ? depuis quelle dépôt est-elle récupérée ?
- Créer un fichier appelée «app.js».
- Ouvrir le fichier app.js et y copier le contenu suivant:

```
const express = require("express");
const app = express()

app.get('/student', function(req, res) {
    res.send("bonjour");
});
app.listen(3000, () => {
    console.log("Server is Running")
})
```

Tout changement que vous effectuerez dans l'application Express ne sera pas visible tant que le serveur n'aura pas redémarré. Nous allons automatiser ce redémarrage dès qu'une modification sera effectuée grâce à *nodemon*.

Installer nodemon en spécifiant l'instruction suivante *:* npm install *-g* nodemon

Depuis le dossier de l'application, taper la commande suivante pour lancer votre première application web avec express: nodemon app.js

Au niveau de votre navigateur, lancer l'url http://localhost:3000/student

2. Routage statique

Créer un serveur express et attachez-lui trois routes de telle sorte qu'il ait le comportement suivant.

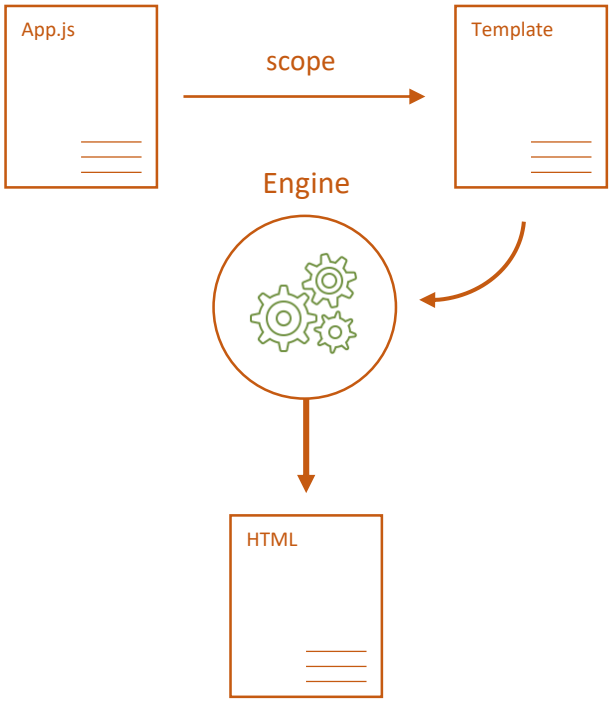
- A la réception de toute sorte de requête, le serveur écrit sur la console «*requête reçue*» suivi de l'heure actuelle (en utilisant Date.now()).
- Si la requête est un GET vers «/student/» le serveur écrit sur la console «*envoie des infos*».
- Si la requête n'est pas GET vers «student », le serveur écrit «abort » sur la console et termine la réponse dans un état d'erreur.
- Vérifier que la visite d'une URL incorrecte renvoie une erreur 404

3. Un routage dynamique

Écrire un router express qui répond à des URI dynamiques de la forme cours/:numeroducours/descr et qui renvoie le message "Vous avez demandez le cours numero" suivi du numero de cours qui apparait dans l'URI.

4. Utilisation des templates

On veut maintenant écrire un serveur express qui renvoie non plus du simple texte ou des pages HTML minimalistes, mais des pages HTML plus complexes. Pour cela on utilisera des templates.



- Installer le système de templates EJS (Embedded Java Script) avec npm.

npm install ejs --save

Et maintenant, nous pouvons demander à notre serveur Express.js d'utiliser un moteur de vue en ajoutant les lignes suivantes au fichier app.js que nous avons créé. Les lignes doivent être ajoutées juste après le const app = express();

```
app.use(express.static("public/img"));
app.use(express.static("public/css"));
app.use(express.static("public/js"));
```

Maintenant, ajoutez ce qui suit à views/index.ejs pour créer le tempalte de vue:

```
<html>
<head>
 <title><%= titre %></title>
 <link rel="stylesheet" href="styles.css">
</head>
<body>
<h1><%= titre %></h1>
<p> Bienvenue à <%= titre %></p>
</body>
</html>
```

Ensuite, nous devons mettre à jour notre route "/" dans routes/index.js pour utiliser le template. Notez que nous passons la valeur du titre au template:

```
app.get("/", (req,res) => {
    const titre = "Express";
    res.render("index", {
    titre: "Express avec EJS",
    });
   });
```

Démarrez l'application, puis accédez à http://localhost:3000 dans votre navigateur et observez que la valeur du titre a été injectée dans le template.

- Ecrire une page Web cours.ejs qui dépend de trois paramètres : un titre, un descriptif de cours et une liste d'enseignants.
- La page contient un header avec le logo de l'université et des liens de navigation
- La page contient également la liste des cours sous forme de tableau avec pour chaque ligne l'id du cours, le l'intitulé du cours, le(s) enseignant(s) et un lien hypertexte nous permettant d'afficher sur une nouvelle page les étudiants inscrits dans ce cours. (écrire un script express (students.js)monté sur un URI dynamique de la forme cours/:numeroducours/students.
- Modifier la page «cours.ejs» pour rendre l'intitulé du cours comme un lien clickable. Le lien renvoie la page description.ejs permettant d'afficher par exemple le syllabus du cours (resumé du cours)
- Tester le résultat de l'appel à différentes pages web avec l'uri suivante : /cours/i/desc.
- Votre application de ressembler à cette interface.