TP 3: Premiers pas avec Express

Objectifs

- Création d'applications Web avec Express.js

Création d'applications Web avec Express.js

Express.js (https://expressjs.com/), ou Express, a été et reste le framework Web le plus populaire pour créer des applications Web dans Node.js. Express était l'un des premiers frameworks Web Node.js et était basé sur le framework Sinatra pour Ruby on Rails. Express.js était un projet de la Fondation OpenJS (https://openjsf.org/projects/), et auparavant de la Fondation Node.js.

Dans cet exemple, nous verrons comment créer un serveur Web Express.js.

Pour commencer, nous allons créer un dossier nommé ***tp3*** et initialiser notre projet en exécutant les commandes suivantes:

$ mkdir tp3

$ cd tp3

$ npm init –yes

Dans cet exemple, nous allons créer un serveur Web qui répond sur la route “**/”** à l'aide d'Express.js. Tout d'abord, commençons par installer le module express:

$ npm install express

Maintenant, nous devons créer quelques répertoires et fichiers pour notre application Web. Dans votre répertoire **tp3**, entrez les commandes suivantes dans votre terminal:

$ touch app.js

$ mkdir routes public

$ touch routes/index.js public/styles.css

Notre fichier **app.js** est l'endroit où nous instancions Express. Ouvrez le fichier **app.js** et importez les dépendances suivantes, définissez le port de notre serveur Express.js et initialisez express:

```
const express = require("express");
const path = require("path");
const index = require("./routes/index");
const PORT = process.env.PORT || 3000;
const app = express();
```

Ensuite, nous allons enregistrer le middleware statique Express.js pour héberger le répertoire public. Nous allons également monter notre route d'index et démarrer notre serveur Express sur notre port spécifié:

```
app.use(express.static(path.join(__dirname, "public")));
app.use("/", index);
app.listen(PORT, () => {
      console.log(`Le serveur écoute sur le port ${PORT}`);
});
```

Nous devons maintenant ajouter notre gestion des routes dans le fichier **index.js** qui se trouve dans le répertoire routes. Ajoutez ce qui suit à **index.js**:

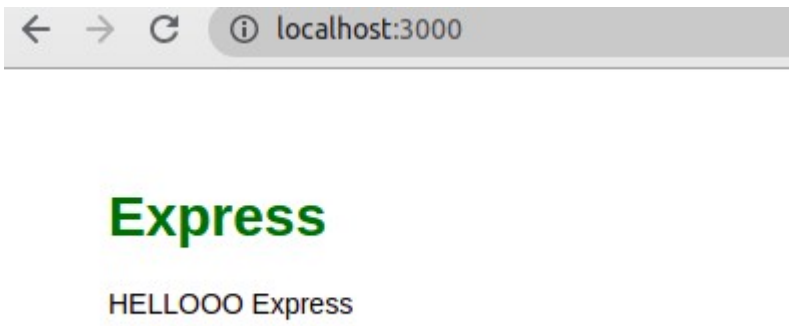
```
const express = require("express");
const router = express.Router();
router.get("/", (req, res) => {
      const title = "Express";
      res.send(`
                  <html>
                  <head>
                  <title> ${title} </title>
                  <link rel="stylesheet" href="styles.css"></head>
                  <body>
                  <h1> ${title} </h1>
                  <p> HELLOOO  ${title} </p>
                  </body>
                  </html>
            `);
});
module.exports = router;
```

Nous pouvons maintenant ajouter du style à notre application en utilisant CSS. Ajoutez ce qui suit à **public/styles.css**:

```
body {
      padding: 50px;
      font: 14px "Lucida Grande", Helvetica, Arial, sans-serif;
}
h1 {
      color:darkgreen;
}
```

Nous pouvons maintenant démarrer notre serveur Express.js en exécutant la commande suivante:

$ node app.js



Nous avons maintenant créé un serveur Web Express.js qui répond avec une page HTML sur la route /.

Le framework Express.js résume les API de protocole Web de base Node.js sous-jacentes fournies par les modules de base **http** et **https**. Express.js fournit une interface pour le routage et l'ajout de middleware.

La ligne **const app = express()**; est l'endroit où nous créons notre serveur Express.js, où app représente le serveur. La fonction **app.use()** est utilisée pour enregistrer le middleware. Dans le contexte d'Express.js, middleware désigne les fonctions qui s'exécutent pendant le cycle de vie d'une requête. Les fonctions middleware d'Express.js ont accès aux objets de requête (req) et de réponse (res).

Le middleware peut exécuter du code, modifier ou opérer sur les objets de requête et de réponse, mettre fin au cycle requête-réponse ou appeler un autre middleware. Le dernier middleware doit mettre fin au cycle requête-réponse, sinon la requête se bloquera.

La première fois que nous appelons **app.use()** dans l'exemple, nous lui passons la méthode **express.static**. La méthode **express.static** renvoie une fonction middleware qui tente de localiser le chemin fourni. Le middleware créera un flux d'écriture à partir du fichier spécifié et le diffusera vers l'objet de demande. Dans l'exemple, nous utilisons la fonction **express.static** pour servir le répertoire public.

Dans le cas d'utilisation suivant de **app.use()**, nous passons la chaîne **/** comme argument et **index** où / est le point de montage du middleware et index est le routeur Express.js que nous avons défini dans **routes/index.js**.

Un point de montage (**mount point**) est utilisé pour restreindre les demandes qui correspondent au point de montage, plutôt que de s'appliquer à toutes les demandes entrantes. L'ordre des middlewares Express.js est important, car ils s'exécutent successivement (l'un après l'autre). Si nous devions utiliser **app.use()** pour enregistrer deux middlewares sur le même point de montage, le premier aurait la priorité.

routes/index.js est l'endroit où nous définissons notre gestion des routes à l'aide de l'utilitaire **Router** d'Express.js. Dans **routes/index.js**, nous créons un objet routeur nommé **router**. L'objet routeur fournit des méthodes qui correspondent aux verbes du protocole HTTP.

Dans l'exemple, nous utilisons uniquement la méthode **router.get()**, mais des méthodes existent également pour les autres verbes HTTP tels que PUT, POST, PATCH et DELETE.

Nous passons **router.get()** deux arguments: le premier est la route à enregistrer (/) et le second est la fonction de gestionnaire de route. La fonction de gestion de route appelle **res.send()** pour renvoyer le contenu HTML. La méthode **send()** est automatiquement ajoutée à l'objet de réponse par

Express.js. La méthode **res.send()** est similaire à la méthode **res.end()**, mais avec des fonctionnalités supplémentaires telles que la détection de type de contenu.

Nous utilisons **module.exports = router;** pour exporter l'instance du routeur. L'instance de routeur est également classée et traitée comme un middleware. Il est possible de fournir un troisième argument, à côté, de la méthode **router.get()**. L'argument suivant est une fonction de rappel qui représente le middleware ou la tâche à suivre. Cependant, dans notre scénario, l'argument suivant était inutile car le cycle requête-réponse s'est terminé lorsque nous avons appelé **res.send()**.

Ajout de vues avec Express.js

Express.js est souvent utilisé pour générer et servir des pages Web HTML. Pour y parvenir, il est courant d'implémenter une couche de vue, qui se charge de la génération du contenu. En règle générale, le contenu est créé dynamiquement à l'aide de templates. Il existe une variété de moteurs de modèles qui gèrent l'injection et l'analyse des modèles.

Voyons comment nous pouvons ajouter une couche de vue à l'aide du moteur de template JavaScript intégré (EJS ==> Embedded JavaScript) avec le serveur Web Express.js que nous avons créé dans cet exemple.

Installez le module EJS depuis npm:

$ npm install ejs --save

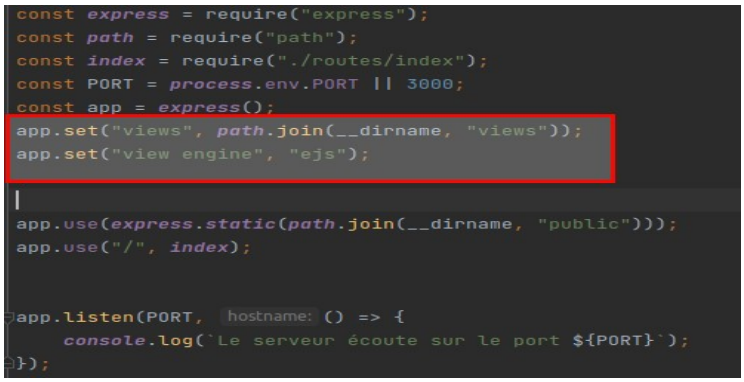
Nous pouvons maintenant créer un répertoire nommé views pour contenir nos templates en entrant les commandes suivantes:

$ mkdir views

$ touch views/index.ejs

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.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");
```



app.set() peut être utilisé pour modifier les paramètres utilisés en interne par Express. La première commande **app.set()** définit le namespace “views” sur notre dossier de vues. Par défaut, Express recherche les vues à cet emplacement; cependant, nous l'avons spécifié pour être explicite.

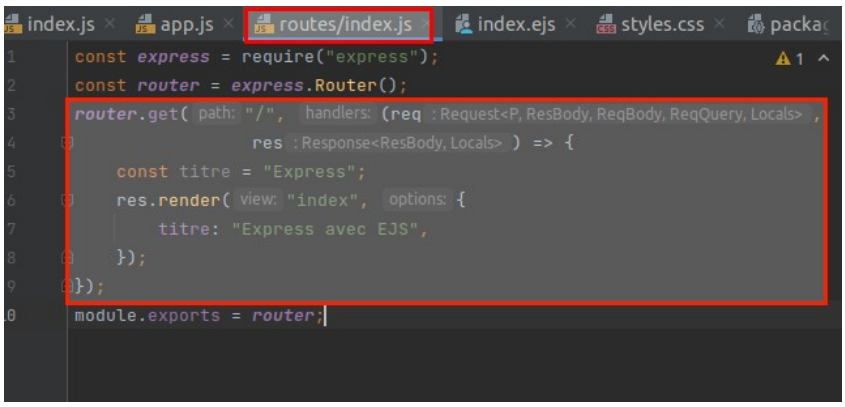
La deuxième commande **app.set()** définit le moteur de vue, et dans notre cas, nous le définissons pour utiliser le moteur de vue EJS. Notez que nous n'avons pas besoin d'importer le module ejs, car Express s'en charge pour nous.:

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

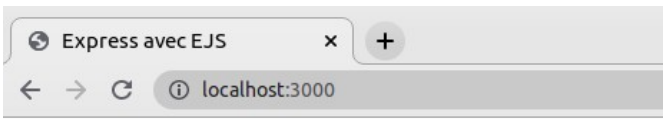
```
<html>
<head>
      <title><%= titre %></title>
      <link rel="stylesheet" href="styles.css">
</head>
<body>
<h1><%= titre %></h1>
<p> Bienvenue à <%= titre %></p>
</body>
</html>
```

Ensuite, nous devons mettre à jour notre route “/” dans r**outes/index.js** pour utiliser le template. Notez que nous passons la valeur du titre au template:

```
router.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 et rendu:



Créer un middleware personnalisé avec Express.js

Pour montrer comment créer un middleware, nous allons créer un middleware qui enregistre la méthode HTTP et l'URL de la requête reçue. Créons un fichier nommé **logger.js** dans le répertoire **middleware**:

$ mkdir middleware

$ touch middleware/logger.js

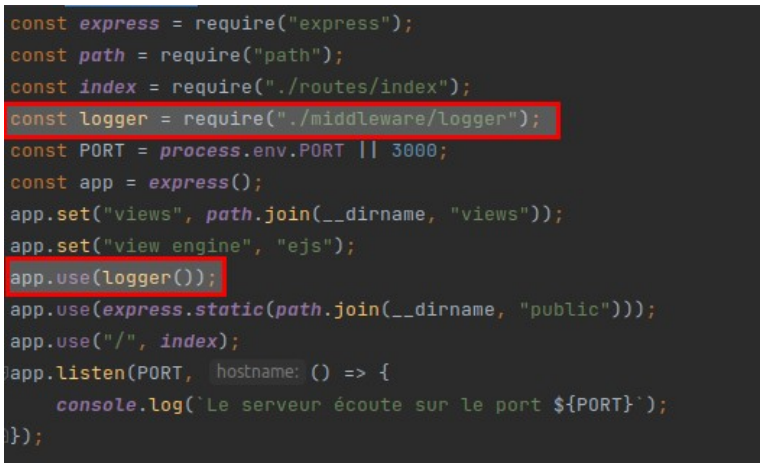
Ajoutez ce qui suit à logger.js pour créer le middleware:

```
module.exports = logger;
function logger() {
	return (req, res, next) => {
		console.log("Requête reçue:", req.method, req.url);
		next();
	};
}
```

Maintenant, de retour dans **app.js,** nous pouvons demander à notre application d'utiliser le middleware “logger”. Pour ce faire, vous devez importer le middleware, puis le transmettre à **app.use()**.

Voici à quoi devrait ressembler **app.js:**

```
const express = require("express");
const path = require("path");
const index = require("./routes/index");
const logger = require("./middleware/logger");
const PORT = process.env.PORT || 3000;
const app = express();
app.set("views", path.join(__dirname, "views"));
app.set("view engine", "ejs");
app.use(logger());
app.use(express.static(path.join(__dirname, "public")));
app.use("/", index);
app.listen(PORT, () => {
	console.log(`Le serveur écoute sur le port ${PORT}`);
});
```



Maintenant, lorsque nous exécutons **app.js** et naviguons vers http://localhost:3000, nous verrons la sortie de terminal suivante, indiquant que notre middleware a été invoqué:

```
Le serveur écoute sur le port 3000
Requête reçue: GET /
Requête reçue: GET /styles.css
Requête reçue: GET /
Requête reçue: GET /styles.css
Requête reçue: GET /
Requête reçue: GET /styles.css
```

Gestion des requêtes POST et des paramètres de routage

Cet exemple vous expliquera comment gérer une demande HTTP POST pour la soumission de formulaire et analyser les données du formulaire. L'exemple utilisera le module **body-parser**. Le module **body-parser** est un middleware qui analyse le corps de la requête entrante.

Installez le module **body-parser** en exécutant la commande suivante:

$ npm install body-parser

Dans le fichier **app.js**, nous devons importer le middleware **body-parser** et demander à notre application Express.js de l'utiliser. En haut de **app.js,** importez le module comme suit:

```
const bodyParser = require("body-parser");
```

Puis ajoutez ce qui suit:

```
app.use(
    bodyParser.urlencoded({
        extended: false,
    })
);
```

L'option **{ extended: false }** indique à **body-parser** d'utiliser la bibliothèque **querystring** pour l'analyse d'URL. Omettre ce paramètre ou le définir sur **true** demandera à **body-parser** d'utiliser la bibliothèque **qs** à la place. La principale différence est que **qs** prend en charge les objets imbriqués.

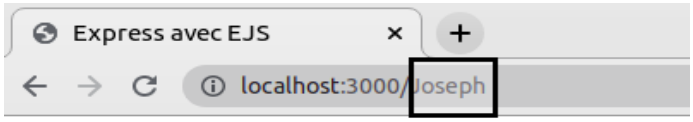
Mettez à jour le fichier **views/index.ejs** comme suit:

```
<html>
<head>
      <title><%= titre %></title>
      <link rel="stylesheet" href="styles.css">
</head>
<body>
<h1><%= titre %></h1>
<p> Bienvenue à <%= titre %><strong style="color: red"><%= name %></strong></p>
<form method=POST action=data>
      Name: <input name=name><input type=submit>
</form>
</body>
</html>
```

Mettez à jour le fichier **routes/index.js** comme suit:

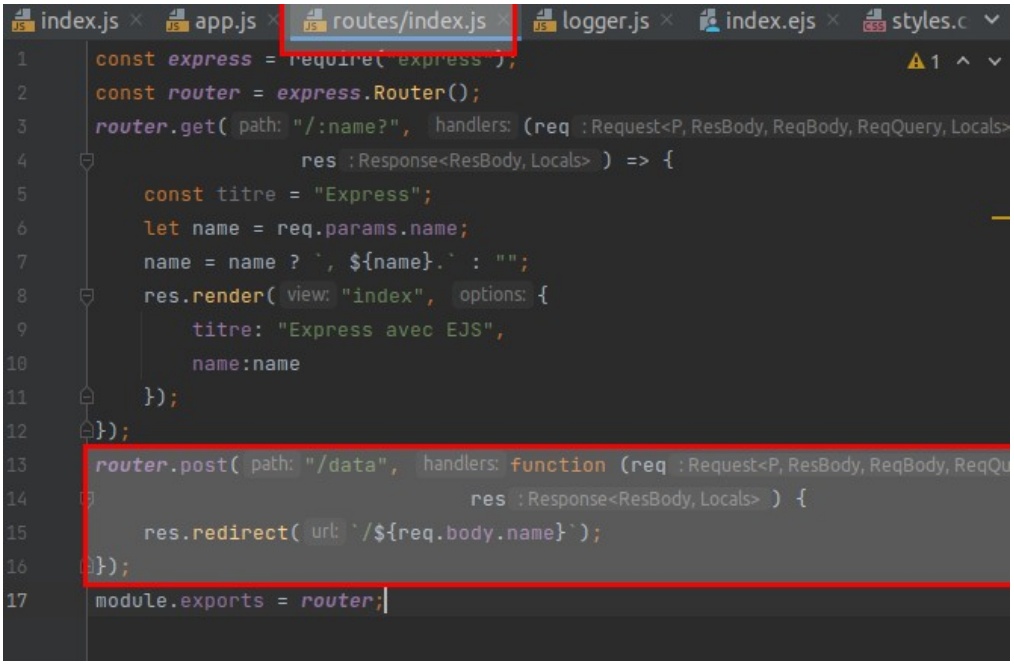
```
const express = require("express");
const router = express.Router();
router.get("/:name?", (req,
                       res) => {
      const titre = "Express";
      let name = req.params.name;
      name = name ? `, ${name}.` : "";
      res.render("index", {
            titre: "Express avec EJS",
            name:name
      });
});
module.exports = router;
```

nous avons modifié notre route HTTP GET dans **routes/index.js.** La route gérera également un paramètre facultatif "**name**" (le **?** indique que le paramètre est facultatif).



Sous l'enregistrement de la route **get()**, nous pouvons maintenant ajouter un gestionnaire de route HTTP POST pour la route **/data**. Cet itinéraire sera appelé sur soumission de notre formulaire. Le gestionnaire de route redirige vers notre route **/name?**:

```
router.post("/data", function (req,
                                res) {
    res.redirect(`/${req.body.name}`);
});
```



Création des “partials” EJS

créer un nouveau sous-répertoire “partials”:

$ mkdir views/partials

Dans ce répertoire, créez un nouveau fichier **head.ejs** ($ touch views/partials/head.ejs) et ajoutez les lignes de code suivantes:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/twitter-bootstrap/4.5.2/css/
bootstrap.min.css">
<link rel="stylesheet" href="styles.css">
<style>
    body { padding-top:50px; }
</style>
```

Ensuite, créez un nouveau fichier **header.ejs** ($ touch views/partials/header.ejs) et ouvrez-le avec votre éditeur de code. Ajoutez les lignes de code suivantes:

```
<nav class="navbar navbar-expand-lg navbar-light bg-light">
      <a class="navbar-brand" href="/">EJS DEMO</a>
      <ul class="navbar-nav mr-auto">
            <li class="nav-item">
                  <a class="nav-link" href="/">Accueil</a>
            </li>
            <li class="nav-item">
                  <a class="nav-link" href="/login">Login</a>
            </li>
      </ul>
</nav>
```

Ensuite, créez un nouveau fichier **footer.ejs** ($ touch views/partials/footer.ejs) et ouvrez-le avec votre éditeur de code. Ajoutez les lignes de code suivantes:

```
<p class="text-center text-muted">&copy; Copyright 2021 IUT-BM</p>
```

Vous avez trois “partials” définis. Vous pouvez maintenant les inclure dans vos vues. Utilisez **<%- include('RELATIVE/PATH/TO/FILE') %>** pour incorporer un EJS “partial” dans un autre fichier.

Le trait d'union **<%-** au lieu de seulement **<%** pour indiquer à EJS de rendre le HTML.

Le chemin d'accès au “partial” est relatif au fichier courant.

Mettez à jour le fichier **index.ejs** comme suit:

```
<html>
<head>
      <title><%= titre %></title>
      <%- include('partials/head'); %>
</head>
<body>
<header>
      <%- include('partials/header'); %>
</header>
<div class="container">
<h1><%= titre %></h1>
<p> Bienvenue à <%= titre %><strong style="color: red"><%= name %></strong></p>
<form method=POST action=data>
      Name: <input name=name><input type=submit>
</form>
</div>

<footer>
      <%- include('partials/footer'); %>
</footer>
</body>
</html>
```

Transmission de données aux vues et aux “partials”

Ouvrez **routes/index.js** et mettez à jour **router.get()** comme suit:

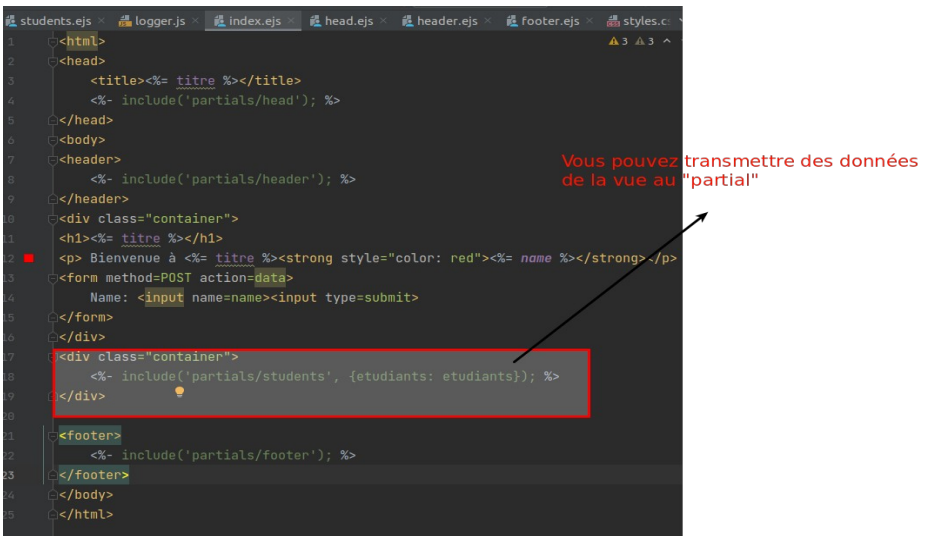
```
router.get("/:name?", (req,
                        res) => {
    const titre = "Express";
    let name = req.params.name;
    name = name ? ` , ${name}.` : "";
    let etudiants = [
        { nom: 'Gates', groupe: "A1"},
        { nom: 'Zuckerberg', groupe: "A2"},
        { nom: 'Pichai', groupe: "B1"},
        { nom: 'Musk', groupe: "B2"}
    ];
    res.render("index", {
        titre: "Express avec EJS",
        name:name,
        etudiants: etudiants
    });
});
```

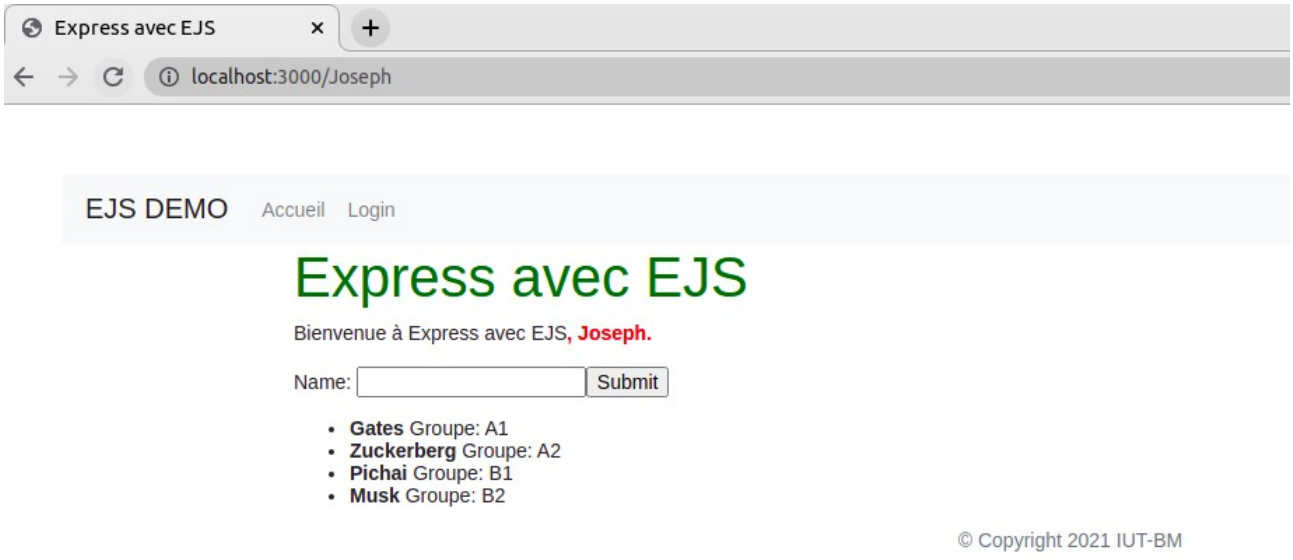
Créer un fichier **partials/students.ejs** ($ touch views/partials/students .ejs) et ajoutez ce qui suit:

```
<% if(typeof etudiants != 'undefined'){ %>
<ul>
    <% etudiants.forEach(function(etudiants) { %>
        <li>
            <strong><%= etudiants.nom %></strong>
            Groupe: <%= etudiants.groupe %>
        </li>
    <% }); %>
</ul>
<% }  %>
```

Maintenant, nous voulons passer la table "**etudiants**" de la vue d'index au “partial” **students.ejs:**

```
<div class="container">
    <%- include('partials/students', {etudiants: etudiants}); %>
</div>
```





À faire: Mise à jour du site “Geeks IUT-BM” avec Express

Téléchargez et extraire le dossier **tp2_ex1_start.zip** depuis https://cours-info.iut-bm.univ-fcomte.fr/index.php/menu-cours-s3/menu-mmi3ihm/2229-s3-programmation-web-cote-serveur-2021.

Nous voulons toujours que l'application ait des pages d'accueil, de cours et d'inscription. Vous devez convertir les routes pour utiliser les mots-clés et la syntaxe trouvés dans Express.js. Vous devez vous assurer que vous servez les assets statiques à partir du répertoire **public** et que toutes les configurations **package.json** nécessaires sont configurées pour lancer l'application localement.

Assurez-vous d'utiliser les **templates EJS**, les **layouts** (express-ejs-layouts) et les “**partials**”.