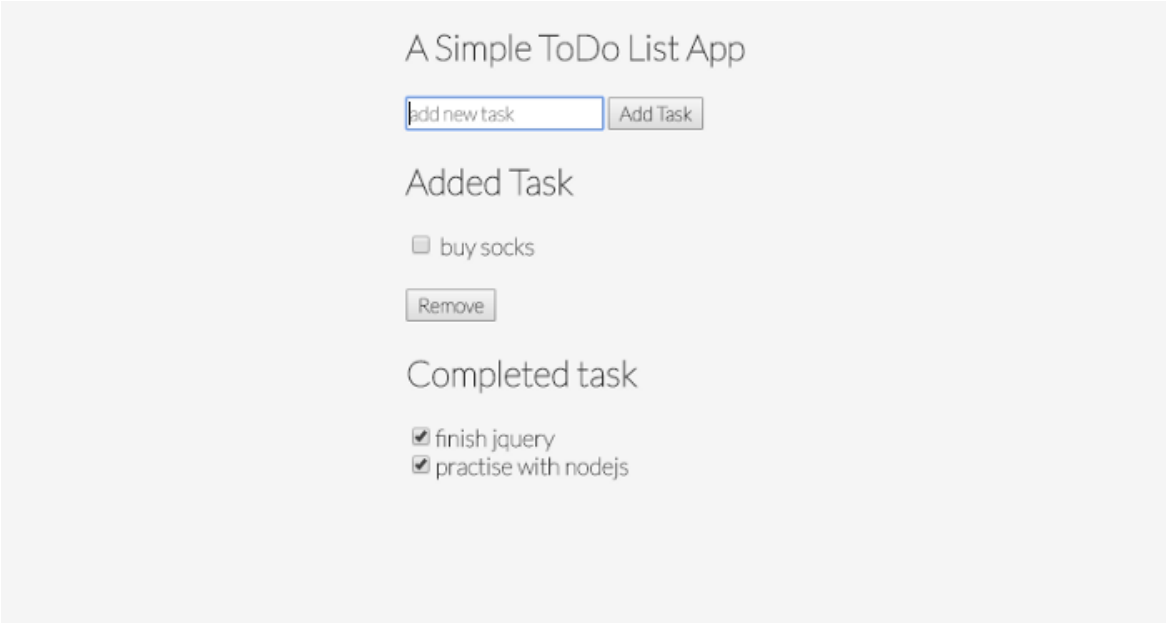
Création d'une application « Todo List » avec Node.js et Express



Cette application est très basique mais présente les caractéristiques suivantes:

- Ajouter et compléter la tâche sur une seule page
- Stocker les nouvelles tâches et les tâches terminées dans un autre tableau
- Utilise le framework express
- Et un style CSS très minimal

Configuration de base :

- Créez un répertoire vide nommé **`todo-app`**
- Ouvrez la console, accédez au répertoire et exécutez **`npm init`**
- Remplissez les informations requises pour initialiser le projet
- Dans le répertoire **`todo-app`**, créez un nouveau fichier nommé **`index.js`**- c'est là que nous allons placer la plupart de notre code.

- Installez express dans la console en lançant:

```
npm install express –save
```

- Une fois installé, nous pouvons ensuite ajouter le code suivant dans le fichier **`index.js`**

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Hello World!');
});
app.listen(3000, function () {
  console.log('Example app listening on port 3000!')
});
```

- Nous pouvons maintenant tester notre serveur en exécutant:

```
noeud index.js
```

Et si vous ouvrez votre navigateur et visitez: **`localhost:3000`** et vous devriez voir Hello World!

Configuration de la vue : pour la configuration de la vue, nous ne répondrons probablement pas avec du texte brut **`Hello World`** lorsque quiconque visite notre route racine, nous renverrions un fichier HTML contenant un texte de titre, des boutons et un formulaire permettant d'ajouter une nouvelle tâche. Et pour cela, nous utiliserons EJS (Embedded JavaScript). EJS est un *langage de template*.

- Installez EJS en lançant:

```
npm install ejs –save
```

- Configurez ensuite le moteur de template avec cette ligne de code (juste en dessous des instructions require) dans le fichier i**ndex.js**:

```
app.set('view engine', 'ejs');
```

- EJS est accessible par défaut dans le répertoire **`views`**. Créez donc un nouveau dossier nommé **`views`** dans votre répertoire.

- Dans ce dossier **`views`**, ajoutez un fichier nommé **`index.ejs`** (Pensez au fichier **`index.ejs`** comme un fichier HTML).

```
<html>
  <head>
    <title> ToDo App </title>
    <link
href="https://fonts.googleapis.com/css?family=Lato:100"
rel="stylesheet">
    <link href="/styles.css" rel="stylesheet">
  </head>
<body>
  <div class="container">
     <h2> A Simple ToDo List App </h2>
<form action ="/addtask" method="POST">
       <input type="text" name="newtask" placeholder="add new
task">        <button> Add Task </button>
    <h2> Added Task </h2>
<button formaction="/removetask" type="submit"> Remove
</button>
</form>
    <h2> Completed task </h2>
</div>
</body>
</html>
```

Maintenant, remplaçons notre code **`app.get`** dans le fichier **`index.js`** par un autre pour envoyer le HTML équivalent au client.

```
app.get ('/', fonction (req, res) {
   res.render ('index');
});
```

À ce stade, une fois que vous avez exécuté **`node index.js`** et entré **`localhost:3000`** dans votre navigateur, vous devriez voir le fichier **`index.ejs`** affiché.

Ajouter une configuration de tâche : Nous n'avons qu'une seule route. Cependant, pour que notre application fonctionne, nous avons également besoin d'une route POST. Si vous consultez notre fichier **`index.ejs`**, vous pouvez voir que notre formulaire envoie une requête POST à la route`/addtask`:

```
<form action = "/ addtask" method = "POST">
```

Maintenant que nous savons où notre formulaire est affiché, nous pouvons définir la route. Une requête POST ressemble à une demande GET, mais avec une légère modification :

```
app.post('/addtask', function (req, res) {
   res.render('index')
});
```

Mais au lieu de simplement répondre avec le même template html, nous devons également accéder au nom du **newtask** saisi par l'utilisateur. Pour cela, nous devons utiliser un *middleware Express*.

Nous allons utiliser le middleware **body-parser** qui nous permettra d'utiliser les paires clé-valeur stockées dans objet **req-body**. Par conséquent, nous pourrons accéder au nom du **newtask** saisi par l'utilisateur côté client et le sauvegarder dans un tableau du côté serveur. Pour utiliser **body-parser**, il faut d'abord l'installer:

npm install body-parser --save

Pour utiliser notre middleware, il faut ajouter les lignes de code suivantes dans notre **index.js**

```
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: true }));
```

Enfin, nous pouvons maintenant mettre à jour notre requête POST pour stocker la valeur de **newtask** dans un tableau tout en ajoutant une boucle à notre fichier **index.ejs** pour afficher une liste de toutes les **newtask** ajoutées par l'utilisateur.
Le fichier **index.js**:

```
//the task array with initial placeholders for added task
var task = ["buy socks", "practise with nodejs"];
//post route for adding new task
app.post('/addtask', function (req, res) {
    var newTask = req.body.newtask;
//add the new task from the post route into the array
    task.push(newTask);
//after adding to the array go back to the root route
    res.redirect("/");
});
//render the ejs and display added task, task(index.ejs) =
task(array)
```

```
app.get("/", function(req, res) {
    res.render("index", { task: task});
});
```

Le fichier **index.ejs**:

```
<h2> Added Task </h2>
   <% for( var i = 0; i < task.length; i++){ %>
<li><input type="checkbox" name="check" value="<%= task[i] %>"
/> <%= task[i] %> </li>
<% } %>
```

Test de l'application :

Exécutez **node index.js** puis ouvrez votre navigateur et entrez: **localhost:3000,** tapez une nouvelle tâche dans le champ de texte et appuyez sur Entrée, votre tâche s'affichera sous le **Added task heading**

Supprimer la configuration de la tâche :

Après avoir ajouté une nouvelle tâche, nous devons pouvoir supprimer la tâche de la section des tâches ajoutées une fois celle-ci terminée et la déplacer dans le tableau complet des tâches côté serveur. Nous pouvons y parvenir en **cochant** la tâche exécutée et en utilisant le bouton **remove button** dans notre fichier EJS.

```
<button formaction="/removetask" type="submit"> Remove </button>
```

Le fichier **index.js**:

```
//the completed task array with initial placeholders for
removed task
var complete = ["finish jquery"];
app.post("/removetask", function(req, res) {
     var completeTask = req.body.check;
//check for the "typeof" the different completed task, then
add into the complete task
if (typeof completeTask === "string") {
     complete.push(completeTask);
//check if the completed task already exist in the task when
checked, then remove using the array splice method
  task.splice(task.indexOf(completeTask), 1);
  } else if (typeof completeTask === "object") {
    for (var i = 0; i < completeTask.length; i++) {
complete.push(completeTask[i]);
    task.splice(task.indexOf(completeTask[i]), 1);
```

```
}
}
   res.redirect("/");
});
```

Le fichier **`index.ejs`**:

```
<h2> Completed task </h2>
    <% for(var i = 0; i < complete.length; i++){ %>
      <li><input type="checkbox" checked><%= complete[i] %>
</li>
<% } %>
```

Terminer l'application :

Pour améliorer l'apparence de l'application, vous avez les styles CSS de base ou bien vous pouvez le personnaliser à votre guise.

Remarque: Pour le CSS, vous devrez ajouter un nouveau dossier au projet appelé `public` et créer dans ce dossier un fichier `css` nommé `styles.css`.

Cependant, Express n'autorisera pas l'accès à ce fichier par défaut. Nous devons donc l'exposer avec la ligne de code suivante dans notre fichier **`index.js`***:*

```
app.use (express.static ("public"));
```

Et avec cela, vous pouvez ajouter vos styles CSS.