1 OBJECTIFS

Cette activité doit permettre :

- de pratiquer le framework Express.JS
- de comprendre le fonctionnement des routes
- d'utiliser les middlewares les plus connus

Cette activité est inspirée du tutoriel sur OpenClassroom :

	https://openclassrooms.com/fr/courses/1056721-des-applications-ultra-rapides-avec-node-js/1057616-tp-la-todo-list Par Mathieu Nebra

2 DÉROULEMENT

Node.JS doit être actif et les commandes npm connues.

2.1 APPLICATION TODO LIST

L'application doit permettre de voir l'ensemble des tâches, d'ajouter des tâches et supprimer une tâche particulière. Nous allons créerons les chemins suivants :

action	vue
Liste des tâches	/todo
Ajouter une tâche	/todo/ajouter
Supprimer une tâche	/todo/supprimer/:id

L'application utilisera un cookie pour mémoriser les informations de l'utilisateur, nous utiliserons le middleware 'cookie-session'.

L'application doit permettre de récupérer facilement les données d'un formulaire, c'est le middleware 'body-parser' qui s'en chargera.

3 CRÉATION DU PROJET

Les étapes classiques de création de notre projet Node.JS sont :

1. Créer un dossier ToDoList pour l'application
2. Dans ce répertoire, lancer la commande **npm init** pour initialiser le projet, avec **todolist.js** comme fichier principal
3. lancer la commande **npm install express ejs** pour installer Express et la dépendance ejs (vues)
4. lancer la commande **npm install -g nodemon** pour installer l'utilitaire nodemon (une application qui relance automatiquement le serveur, lorsqu'un fichier du serveur est modifié). L'option -g permet de rendre l'installation générale (et non locale au projet).
5. lancer la commande **npm install cookie-session body-parser** pour installer les middlewares concernés
6. Créer un répertoire **views** qui contiendra les fichiers EJS d'affichage

Le module cookie-session permet de gérer facilement un cookie de session (rappel, NodeJS s'exécute côté serveur) tandis que le module body-parser permet d'analyser facilement le contenu d'une requête HTML (notamment le contenu d'une requête de type POST).

Vous trouverez une aide dans la documentation d'Express :

http://expressjs.com/en/4x/api.html (la version 5 est en beta au 08 octobre 2022)

https://www.geeksforgeeks.org/express-js-app-listen-function/

3.1 CODE SOURCE TODOLIST.JS

Le fichier principal doit contenir l'emplacement des dépendances et le routage des trois vues.

```
var express = require('express')
var session = require('cookie-session')
var bodyParser = require('body-parser')
var urlencodedParser = bodyParser.urlencoded({ extended: false })

var app = express()

// création de la session
app.use(session({secret: 'todotopsecret'}))
.use(function(req, res, next){
    // vérification s'il existe déjà une liste (middleware 'maison')
    if (typeof(req.session.todolist) == 'undefined') {
        req.session.todolist = []
    }
    next()
})
.get('/todo', function(req, res) {
    // On affiche la todolist et le formulaire
    res.render('todo.ejs', {todolist: req.session.todolist})
})
.post('/todo/ajouter/', urlencodedParser, function(req, res) {
    // ajouter un élément à la liste (si le champs n'est pas vide)
    if (req.body.newtodo != '') {
        req.session.todolist.push(req.body.newtodo)
    }
    res.redirect('/todo')
})
.get('/todo/supprimer/:id', function(req, res) {
    // supprimer l'élément 'id' de la liste
    if (req.params.id != '') {
        req.session.todolist.splice(req.params.id, 1)
    }
    res.redirect('/todo')
})
.use(function(req, res, next){
    // redirection vers l'affichage de la liste si la vue n'existe pas
    res.redirect('/todo')
})

.listen(3000, function () { console.log("Server is up on port 3000")})
```

lancer le script avec la commande **nodemon todolist.js**

L'erreur suivante doit s'afficher :

```
Error: Failed to lookup view "todo.ejs" in views directory "……\ToDoList\views"
```

Si vous obtenez une erreur de droit d'exécution de nodemon, recherchez la solution dans les annexes du support de cours (voir Set-ExecutionPolicy Unrestricted)

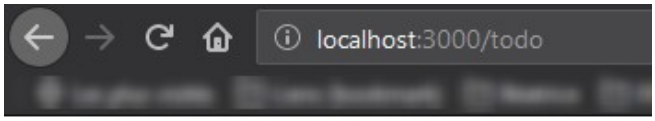
3.2 CODE SOURCE TODO.EJS

Le code pour la vue

/views/todo.ejs

```
<!DOCTYPE html>

<html>
    <head>
        <title>ToDo List</title>
        <style>
            a {text-decoration: none; color: black;}
        </style>
    </head>

    <body>
        <h1>Liste des tâches</h1>

        <ul>
        <% todolist.forEach(function(todo, index) { %>
            <li><a href="/todo/supprimer/<%= index %>">x</a> <%= todo %></li>
        <% }); %>
        </ul>

        <form action="/todo/ajouter/" method="post">
            <p>
                <label for="newtodo">Que dois-je faire ?</label>
                <input type="text" name="newtodo" id="newtodo" autofocus />
                <input type="submit" />
            </p>
        </form>
    </body>
</html>
```

En rafraîchissant la vue dans le navigateur, l'interface doit désormais fonctionner.



Liste des tâches

- ✗ laver la vaisselle
- ✗ faire un cours SLAM5 sur Express.JS
- ✗ Mettre un 2/20 à N°8

Que dois-je faire ? Envoyer

L'affichage du code source de la page montre que la création d'une tâche affecte un lien sur le symbole ✗. De même le formulaire pointe sur la vue /todo/ajouter du serveur.

```
<!DOCTYPE html>
<html>
    <head>
        <title>ToDo List</title>
        <style>
            a {text-decoration: none; color: black;}
        </style>
    </head>

    <body>
        <h1>Liste des tâches</h1>

        <ul>
            <li><a href="/todo/supprimer/0">x</a> laver la vaisselle</li>
            <li><a href="/todo/supprimer/1">x</a> faire un cours SLAM5 sur
Express.JS</li>
            <li><a href="/todo/supprimer/2">x</a> Mettre un 2/20 à N°8</li>
        </ul>

        <form action="/todo/ajouter/" method="post">
            <p>
                <label for="newtodo">Que dois-je faire ?</label>
                <input type="text" name="newtodo" id="newtodo" autofocus />
                <input type="submit" />
            </p>
        </form>
    </body>
</html>
```

4 REMARQUES

4.1 FONCTION RENDER()

Cette instruction exécute le rendu d'une page HTML+EJS et l'envoie dans la variable res. Il n'est plus possible d'exécuter de code après cette fonction (la page est envoyée au client).

La syntaxe est la suivante :

```
res.render(view, [localdata], [callback])
```

- view : le nom du fichier EJS
- localdata : un objet (JSON) qui sera accessible dans la page EJS
- callback : une fonction de retour

4.2 FONCTIONS GET() ET POST()

Ces deux fonctions permettent d'exécuter une fonction lorsque la requête reçue par le serveur est de type GET ou POST. Par défaut, lorsqu'un navigateur demande l'affichage d'une page web, il effectue une requête GET. La requête POST impose l'usage d'un formulaire HTML utilisant cette méthode.

La syntaxe est la suivante :

```
app.get(chemin, fonction)
```

- chemin : le chemin depuis la racine du site web (commence donc impérativement par un `/` : par exemple `/todo`).
- Fonction : une fonction anonyme (ou une fonction fléchée) de traitement qui reçoit deux arguments : **req** pour la requête en provenance du navigateur, et **res** qui contiendra la réponse du serveur. Un troisième argument optionnel est **next** qui est la fonction suivante à exécuter.

4.3 FONCTION USE()

Cette fonction créée un middleware, c'est-à-dire une fonction qui s'exécute de manière transparence (elle ne termine normalement pas une requête). Elle peut modifier la requête ou servir pour du débogage.

4.4 ORDRE

Il faut noter que le routage doit se faire du chemin le plus détaillé (comportant le plus de sous-répertoires) au plus généralisé (par exemple, le chemin `/`).

5 Extension

Dans cette exploration, vous avez simplement utilisé Express.JS dans un cadre entièrement défini.

Maintenant, essayez d'ajouter les éléments suivants à votre liste :

5.1 Champs 'Priorité'

Comment ajouter une priorité au moment de l'enregistrement de la tâche ?

5.2 Édition d'une tâche

Comment ajouter un bouton qui permettrait d'aller sur une vue, permettant d'éditer/modifier une tâche sélectionnée ?