Déroulement - Express JS

Présentation initiale

Démarrage

Présentation

Express JS est un framework (cadre de travail) Node.js de taille limitée et proposant de nombreuses fonctions pour simplifier la réalisation d'un site web. Express JS facilite l'organisation de l'application grâce à plusieurs notions

- guide dans la gestion de la requête et la réponse
- pile de middlewares
- routages
- templates

Installation

Après avoir créé un dossier spécifique, nous lançons les commandes suivantes dans ce dossier :

Création d'un package JSON

```
$ npm init
```

Il s'agit de créer un fichier JSON contenant une description de l'application Express JS créée (nom, version, fichier de départ …). L'ensemble des spécifications sont données dans la documentation npm.

Installation d'Express JS proprement dit

```
$ npm install express --save
```

L'option --save permet d'ajouter Express dans la liste des dépendances dans le fichier JSON préalablement créé

Premier affichage

Après avoir écris ce code :

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('Bonjour à tous !');
});

var server = app.listen(8080, function () {
  var addresseHote = server.address().address;
  var portEcoute = server.address().port;

  console.log(' L\'application est disponible à l\'adresse http://%s:%s', addresseHote, portEcoute);
});
```

exécutons le :

```
$ node premiereEssai.js
```

Principe des routes

Si nous souhaitons, prendre en compte un changement dans l'url, nous modifions notre fichier en y ajoutant les lignes suivantes :

```
  …
  res.send('Bonjour à tous !');
});

app.get('/fin', function (req, res) {
  res.send('Au-revoir !');
```

```
});

var server = app.listen(8080, function () {
…
```

Après avoir relancé le serveur web, nous pouvons modifier l'affichage si nous entrons la valeur fin dans la barre d'adresse après localhost.

Fichier statique

L'application web permet de générer des fichiers html en fonction des requêtes qui lui sont envoyées; mais également de fournir des fichiers statiques stockés sur le serveur tels les images, les CSS ou le JavaScript par exemple.

Express permet de simplifier l'écriture des URL pour ces fichiers en les stockants tous dans un dossier parent.

Il faut utiliser, pour ce faire, une fonction middleware (voir plus bas pour cette notion) en écrivant :

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

Vous pouvez désormais stocker tous vos fichiers dans le dossier public, situé à la racine de votre application, et écrire les URL comme suit:

```
http://localhost:8080/images/kitten.jpg
http://localhost:8080/css/style.css
http://localhost:8080/js/app.js
http://localhost:8080/images/bg.png
http://localhost:8080/hello.html
```

Vous pouvez créer de multiples dossiers statiques :

```
app.use(express.static('public'));
app.use(express.static('files'));
```

Express cherchera successivement vos fichiers dans les différents dossiers statiques indiqués.

Les templates : Pug

Principe de base

Pug est un langage de template. Cela permet de séparer le code qui génère l'affichage du reste du code JavaScript. Il faut obligatoirement utiliser la syntaxe du langage. Nous insérons le code suivant dans le fichier JavaScript

```
var express = require('express');
var app = express();

//définit le dossier de base pour les fichiers statiques (css, images, js, …)
app.use('/static', express.static('mestemplates'));

//permet d'utiliser Pug
app.set('view engine', 'pug');

//redéfinir le dossier recevant les fichiers .pug (par défaut : views)
app.set('views','mestemplates');

//chemin de base de mon application :
app.get('/',function (req,res) {
  //permet le rendu du fichier index.pug qui donne un html envoyé ensuite au client
  res.render('index', {title: 'Titre de la page', message : 'Du texte dans la page'});
});
```

Nous créons un fichier index.pug dans le dossier mestemplates.

```
doctype html
head
  title= title
body
  h1= message
```

Inclusion d'un fichier dans un autre

Dans nos pages web, une partie de la structure est commune (en-tête, menu …). Il est donc préférable de séparer ces parties communes du reste. Prenons, tout d'abord, un fichier index.pug

```
doctype html
html
  include ./includes/head.pug
  body
    h1= titreH1
    p Un texte court
  include ./includes/foot.pug
```

Nous utilisons ensuite les fichiers head.pug et foot.pug situés dans le dossier includes lui-même situé dans le dossier mestemplates.

- Le fichier head :

```
head
  title= titrePage
  script(src='script.js')
  link(rel="stylesheet", href="style.css")
```

- Le fichier foot :

```
p
  a(href='mailto:moi@maboitemail.com') Me contacter
```

L'ensemble nous donne un fichier html comple qui est envoyé au client.

Utilisation d'un layout

Une autre technique de séparation du code consiste à utiliser un Layout. En premier lieu, prenons le fichier layout.pug.

```
doctype html
html
  head
  block title
    title Default title
  link(rel="stylesheet", href="style.css")
  script(src="script.js")
  body
    block content
```

Si nous utilisons le fichier index.pug ci-dessous, nous obtenons une page complète.

```
extends ./layout.pug

block title
  title= titrePage

block content
  h1= titreH1
  p Un texte court
```

Chaque fichier .pug appelé fera appel au layout.

Commencement

Exercice 1 : Premier serveur

Intitulé

Création d'un premier serveur :

- require du module
- exécution d'express.
- methode get avec fonction de retour
- écoute du serveur

Correction

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
  res.send('Bonjour');
});

app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Les routes et les fichiers statiques

Exercice 2 : Gestion des routes - début

Intitulé

- Même base que dans l'exercice 1
- Mise en place des routes pour distinguer deux actions

Correction

```
var express = require('express');
var app = express();

app.get('/', function(req, res) {
  res.send('Bonjour');
});

app.get('/fin', function(req, res) {
  res.send('Au revoir');
});

app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Exercice 3 : Utilisation des fichiers statiques 1

Intitulé

- Reprise de l'exercice 1
- Utilisation de la méthode app.use pour afficher une image

Correction

```
var express = require('express');
var app = express();

app.use("/public", express.static(__dirname + '/public'));

app.get('/', function(req, res) {
  res.send('Bonjour');
});

app.get('/image', function(req, res) {
  res.send('<img src="public/apache.png">');
});

app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Exercice 4 : Utilisation des fichiers statiques 2

Intitulé

- utilisation de deux dossiers
- chaque dossier permet de servir une image en fonction de la route donnée

Correction

```
var express = require('express');
var app = express();

app.use("/public", express.static(__dirname + '/public'));
app.use("/public2", express.static(__dirname + '/public2'));

app.get('/', function(req, res) {
  res.send('Bonjour');
});

app.get('/image', function(req, res) {
  res.send('<img src="public/apache.png">');
});

app.get('/image2', function(req, res) {
  res.send('<img src="public2/512x512bb-85.png">');
});

app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Exercice 5 : Utilisation de fichiers statiques 3

Intitulé

- Utilisation de res.sendFile
- Envoi du fichier html et de l'image

Correction

```
var express = require('express');
var app = express();

app.use("/public", express.static(__dirname + '/public'));

app.get('/', function(req, res) {
  res.sendFile('page.html',{root:'public'});
});

app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Exercice 6 : Routes et fichiers statiques

Intitulé

- Réalisation d'un mini site
- Trois pages sont fournis en fonction de la route indiquée

Correction

```
var express = require('express');
var app = express();

app.use("/public", express.static(__dirname + '/public'));

app.get('/', function(req, res) {
  res.sendFile('page.html',{root:'public'});
});

app.get('/page', function(req, res) {
```

```
  res.sendFile('page2.html',{root:'public'});
});


app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Les templates

Exercice 7 : Template Pug - première utilisation

Intitulé

- Définition d'un dossier pour les fichiers statiques
- Utilisation du module Pug
- Utilisation de la méthode res.render.

Correction

template

```
doctype html
head
  title Titre
body
  h1 Un titre de paragraphe
```

fichier js

```
var express = require('express');
var app = express();

//définit le dossier de base pour les fichiers statiques (css, images, js, …)
app.use("/static", express.static(__dirname + '/static'));

//permet d'utiliser Pug
app.set('view engine', 'pug');

//redéfinir le dossier recevant les fichiers .pug (par défaut : views)
app.set('views','static');

//chemin de base de mon application :
app.get('/',function (req,res) {
  //permet le rendu du fichier index.pug qui donne un html envoyé ensuite au client
  res.render('index');
});

app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Exercice 8 : Template Pug - utilisation de variables

Intitulé

- Reprise de l'exercice précédent
- Utilisation d'un objet passés par la méthode res.render

Correction

template

```
doctype html
head
  title= title
body
  h1= message
```

```
  p Il fait beau
```

fichier js

```
var express = require('express');
var app = express();

//définit le dossier de base pour les fichiers statiques (css, images, js, …)
app.use("/static", express.static(__dirname + '/static'));

//permet d'utiliser Pug
app.set('view engine', 'pug');

//redéfinir le dossier recevant les fichiers .pug (par défaut : views)
app.set('views','static');

//chemin de base de mon application :
app.get('/',function (req,res) {
  //permet le rendu du fichier index.pug qui donne un html envoyé ensuite au client
  res.render('index2', {title: 'Titre de la page', message : 'Du texte dans la page'});
});

app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Exercice 9 : Template Pug - mini-site

Intitulé

- Mise sous forme de template Pug des trois documents HTML précédents.

Correction

Exercice 10 : Template Pug - inclusion

Intitulé

- Utilisation de l'inclusion.
- L'en-tête et le menu sont mis à part.

Correction

Exercice 11 : Template Pug - layout

Intitulé

- Utilisation du layout.
- L'en-tête et le menu sont mis à part.

Correction

Amélioration

Exercice 12 : Gestion des routes

Intitulé

- Utilisation des références dans l'url
- Mise en place de req.params et req.query

Correction

```
var express = require('express');
var app = express();


app.get('/test/:info1/:info2', function(req, res) {
  console.log(req.params);
  res.send('Bonjour');
});

app.get('/search', function(req, res) {
  console.log(req.query);
  res.send('Bonjour');
});


app.listen(808,function(){
  console.log('Ecoute sur le port 808');
});
```

Exercice 13 : Gestion des Posts - body parser

Intitulé

- Reprise du code de l'exercice précédent.
- Création d'une page avec un formulaire.
- Traitement du formulaire pour afficher les valeurs envoyées dans une autre page.

Correction

```

```

Exercice 14 : Gestion des erreurs

Intitulé

- Reprise du code de l'exercice précédent.
- Ajout de la gestion des pages 404.

Correction

```

```

Exercice 15 : Les sessions

Intitulé

- Création d'un fichier js qui affiche le contenu d'un template Pug.
- Créez dans la variable de session un compteur avec req.session.
- À chaque connection, il faut incrémenter le compteur.

Correction

```

```

Exercice 16 : Gestion de la base de donnée

Application pratique

Exercice 17 : Blog

Exercice 18 : MVC

Lien avec la base de données

Intitulé

Correction

```
//Gestion de la base de données
// Création du client
var MongoClient = require('mongodb').MongoClient;
// Connexion à la database
MongoClient.connect(format('mongodb://localhost:27017/testdb'), function(err, db) {
  var collection = db.collection('user');
  // Ajout d'un utilisateur
  collection.insert({
    prenom : 'Kevin',
    nom : 'Durand'
  });
});
```