Introduction au développement d'applications Web côté serveur avec Node.js et Express

Nicolas Le Sommer
Nicolas.Le-Sommer@univ-ubs.fr
Université Bretagne Sud, IUT de Vannes, Département Informatique

Présentation de Node.js

- Node.js est un environnement de développement reposant sur le support d'exécution V8 JavaScript Engine
  - V8 implante la dernière norme ECMAScript
- Node.js implante un modèle basé événement et un modèle d'entrée/sortie non bloquant
- Node.js offre une API pour
  - la programmation réseau (HTTP, HTTPS, UDP/Datagram, SSL)
  - gestion de fichiers (FileSystem, Stream)
  - ...
- https://nodejs.org

Exemples

- Serveur HTTP

```
var http = require('http');
http.createServer(function(req,res){
    res.writeHead(200, {'Content-Type': 'text/plain'});
    res.end('Hello World\n');
}).listen(1337, '127.0.0.1');
console.log('Server running at http://127.0.0.1:1337/');
```

- Affichage du contenu d'un répertoire

```
const fs = require('fs');

fs.readdir('.', (error, files) => {
  console.log(files);
});
```

- Connexion/déconnexion à une base de données

```
let db = new sqlite3.Database(':memory:', (err) => {
  if (err) {
    return console.error(err.message);
  }
  console.log('Connected to the in-memory
               SQlite database.');
});
```

```
db.close((err) => {
  if (err) {
    return console.error(err.message);
  }
  console.log('Close the database connection.');
});
```

Quelques outils

- *npm* (Node Package Module) : gestionnaire de modules de Node.js
- *yarn*: outils de gestion de paquetages et de gestion de dépendances entre paquetages
- *Grunt* : gestionnaire de tâches (équivalent au Makefile)
- Socket.io : API Socket en Javascript
- Cadres de conception pour applications Web
  - Express, Meteor, Sails, NuxtJS
- Outils de comptabilité Navigateur/Node.js
  - Browserify
- Outils d'optimisation et de réduction du code (minification)
  - Parceljs, Webpack.js, Uglify.js, Closure

Introduction à npm

- Quelques commandes à connaître
  - ***npm init*** : création d'un nouveau module (création du fichier package.json)
  - ***npm install*** : installation d'un nouveau module localement
  - ***npm install -g*** : installation d'un nouveau module de manière globale (nécessite les droits root)
  - ***npm update*** : mise-à-jour des modules
  - ***npm list*** : liste les modules utilisés

Introduction à npm

- Déclaration des exports dans le code

```
var obj1 = new Object1();
module.exports = obj1;
```

```
var obj1 = new Object1();
var obj2 = new Object2();
module.exports = {o1: obj1, o2: obj2};
```

- Déclaration des dépendances dans le code

```
var http = require('http');
var obj1 = require('mon_module').o1;
var obj2 = require('mon_module').o2;
```

Fichier package.json

- Déclaration des dépendances et des scripts dans le fichier ***package.json***

```
{
  "name": "webapp.js",
  "version": "1.0.0",
  "description": "Implementation of my web application",
  "main": "node_app.js",
  "scripts": {
    "serve_webapp": "webpack-dev-server --open",
    "clean": "\rm -rf build node_modules",
    "watch": "webpack --watch"
  },
  "author": "NLS",
  "keywords": [],
  "license": "ISC",
  "dependencies": {
    "engine.io": "^3.2.0"
  },
  "devDependencies": {
    "webpack": "^4.6.0",
    "webpack-cli": "^2.1.2",
    "webpack-dev-server": "^3.1.4"
  }
}
```

Les modules dans Node.js

- Les modules introduits dans ES6 sont disponibles à partir de la version 10 de Node.js
- Les fichiers doivent être suffixés par .mjs (et non pas .js).
  - Pas de partage possible entre de modules entre les navigateurs et Node.js sans renommage des fichiers (i.e. .js → .mjs)

Express.js : présentation

- Express est un intergiciel dédié au développement d'applications Web
  - http://expressjs.com/
- Il repose sur Node.js
- Il fournit notamment
  - un mécanisme de routage
    - Aiguille les requêtes vers les bons « contrôleurs » chargés de traiter les requêtes
  - un serveur HTTP
  - des moteurs de modèle de rendu
    - Jade, Pug, EJS
  - une gestion des sessions

Express.js : le routage

- Routage de base

```
app.METHOD(PATH, HANDLER)
```

- app : est une instance d'Express
- METHOD : une méthode HTTP
- PATH : chemin sur le serveur
- HANDLER : fonction de traitement

```
app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.post('/', function (req, res) {
  res.send('Got a POST request');
});
```

- Gestionnaire de routage

```
app.route('/book')
  .get(function(req, res) {
    res.send('Get a random book');
  })
  .post(function(req, res) {
    res.send('Add a book');
  })
  .put(function(req, res) {
    res.send('Update the book');
  });
```

Express.js : le routage

- La classe ***express.Router*** permet de créer des gestionnaires de route modulaires
- Une instance ***Router*** est un intergiciel et un système de routage complet
  - Il peut être considéré comme une mini-application, ou un sous ensemble de l'application.

```
var express = require('express');
var router = express.Router();

// middleware that is specific to this router
router.use(function timeLog(req, res, next) {
  console.log('Time: ', Date.now());
  next();
});
// define the home page route
router.get('/', function(req, res) {
  res.send('Birds home page');
});
// define the about route
router.get('/about', function(req, res) {
  res.send('About birds');
});
```

Méthodes de réponse

Méthode	Description
res.download()	Invite à télécharger un fichier.
res.end()	Met fin au processus de réponse.
res.json()	Envoie une réponse JSON.
res.redirect()	Redirige une demande.
res.render()	Génère un modèle de vue.
res.send()	Envoie une réponse de divers types.
res.sendFile()	Envoie une réponse sous forme de flux d'octets.
res.sendStatus()	Définit le code de statut de réponse et envoie sa représentation sous forme de chaîne comme corps de réponse.

Express et les sessions

- Pour gérer les sessions, il faut installer le module « express-session »
- La gestion de session repose sur un cookie
- Une session peut être créée, sauvegardée, restaurée, détruite

```
var express = require('express');
var session = require('express-session') ;
var app = express() ;
app.use(session({
  secret: 'mysecret',
  cookie: { secure: true }
})) ;
app.get('/', function(req, res, next) {
  if (req.session.my_variable === undefined) {
       req.session.my_variable = req.session.id ; // the session ID
  }
}) ;
app.get('/quit', function(req, res, next) {
  if (req.session.id !== undefined) {
       req.session.destroy() ;
  }
}) ;
```

Express.js : fichier de configuration app.js

- Fichier *app.js* de l'application Web développée avec Express (fichier représentant l'application)

```
var createError = require('http-errors');
var express = require('express');
var path = require('path');
var cookieParser = require('cookie-parser');
var logger = require('morgan');

var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(logger('dev'));
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));
```

```
app.use('/', indexRouter);
app.use('/users', usersRouter);

// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error');
});

module.exports = app;
```