Portfolio en ligne

Guillaume Pelletier-Auger

29 septembre 2016

Résumé

Ce portfolio d'art numérique est un site statique généré avec Node.js.

1 Génération d'un site Web statique avec Node.js

Je crée un portfolio en ligne pour mon travail en art numérique et je planifie le mettre à jour fréquemment. Le processus de mise à jour doit être aussi simple que possible. Mon but est de généré le contenu HTML en utilisant une série de modules Node.js. Chacun de ces modules sera traité en utilisant le module *fs* (pour *file system*) de Node.js.

Voici ce que j'ai en ce moment :

```
var fs = require("fs");

var header = `<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Guillaume Pelletier-Auger</title>
</head>
`;

var body = `
<body><h1>I'm writing my own HTML files with Node.js. It's pretty neat.</h1>
It's actually like... awesome! I can now think of a website as a data structure.
</body>
</html>
`;

fs.writeFile('index2.html', header + body, function(err) {
    if (err) {
        return console.error(err);
    } else {
        console.log("Data written successfully!");
    }
});
```

Je songeais précédemment à utiliser un générateur de sites Web statiques (*Static Site Engine*) comme Harp.js, mais il m'est clair maintenant que ce sera beaucoup plus simple de tout faire moi-même dans Node.js. Mon système aura ainsi une bien plus grande indépendance fonctionnelle.

2 Un site Web est une structure de données

Je dois penser à mon portfolio comme étant une structure de données. J'ai commencé avec ce fichier JSON :

```
{
    header : "<head></head><body><h1></h1>",
    items : {
        oscillators : {
            title_fr : "Les Oscillateurs",
            title_en : "Oscillators",
            content_fr : `

            `,
            content_en : `
            `,
        },
        joy_and_confusion : {

        },
        dunes : {
        }
    }
}
```

Je dois avoir une fonction qui ressemble à ça :

```
function buildWorks() {
    var content = `` ;
    for (var i = 0 ; i < portfolio.items.length ; i++)  {
        var title = portfolio.items[i].title ;
        content = content + `<img src="${title}.jpg"></a>` ;
    }
}
```

3 Liste de fichiers comme base de données

Mon système sera beaucoup plus efficace si chaque page de mon portfolio correspond à un fichier différent dans ma base de données. Il me serait en effet désagréable de travailler sur les diverses pages de mon portfolio si elles étaient toutes écrites dans le même fichier JavaScript ou JSON. Ça deviendrait rapidement lourd et pas clair.

En utilisant la méthode *readdirSync* du module *fs* de Node.js, j'obtiens la liste complète des fichiers qui sont dans le dossier *Pages*. Les fichiers qui seront dans le dossier *Pages* seront des modules Node.js (d'où l'utilisation de la méthode *require*) qui renvoient un objet JavaScript littéral. L'objet contient les propriétés fr, en et link.

```
var fs = require("fs");
var files = fs.readdirSync('pages');
var page = require('./pages/' + files[0]);
```

Voici à quoi ressemble une page définie comme module Node.js, pour l'instant :

```
exports.fr = {
    title : "Les Dunes",
    description : "Une série d'image générées par des fonctions itératives.",
    content : `<i>Les Dunes</i> est une série d'images générées par une variété de fonctions itératives.
    Elles ont été créées avec p5.js.`
};

exports.en = {
    title : "Dunes",
    description : "A series of images generated by iterated functions.",
    content : `<i>Dunes</i> is a series of images generated by various iterated functions.
    They were created in JavaScript with the p5.js library.`
}

exports.link = null;
```

La fonction qui va générer toutes mes pages doit donc traiter ces données. Elle doit :

1. Vérifier que toutes les données sont présentes.
2. Vérifier si la propriété *link* est *null* ou non.
   (a) Si *link* est *null*, ça veut dire qu'il s'agit d'une page qui sera hébergée directement sur mon site Web. La fonction doit donc créer un fichier HTML en utilisant les données contenues dans l'objet renvoyé par le module.
   (b) Si la propriété n'est pas *null*, ça veut dire que cet élément du portfolio est hébergé sur un site externe. La fonction ne crée donc aucun fichier HTML pour cet élément du portfolio.
3. La fonction doit utiliser chaque élément dans le dossier *Pages* pour générer les fichiers index.html, index-en.html et index-fr.html. Forcément, la fonction doit aussi lire dans un autre

dossier, nommé *Thumbnails*, qui contiendra une image pour chaque élément du portfolio.

Ma boucle pour traiter chacun des fichiers dans le dossier *pages* ressemblera donc à ça :

```
//We get a reference to the Node.js filesystem module.
var fs = require("fs");

//We let the files variable be an array containing
//the full names of all the files inside the "pages" folder.
var files = fs.readdirSync('pages');

//We loop through the list of files.
//If the currently processed file doesn't have an external link, we run the generatePage() function.
for (var i = 0; i < files.length; i++) {
    var page = require('./pages/' + files[i]);
    if (!files[i].link){
        generatePage(files[i]);
    }
}

//We call the generateMosaic() function twice, sending it the list of files and language option.
generateMosaic(files, "fr");
generateMosaic(files, "en");
```

La fonction *generateMosaic()* ressemblerait à ça :

```
function generateMosaic(files, language) {
    //We start writing the HTML content.
    var mosaic = `<div class="mosaic">`;

    //We loop through all the files.
    for (var i = 0; i < files.length; i++) {
        //We let "page" be the file that is currently processed.
        var page = require('./pages/' + files[i]);

        if (language == "fr") {
            var pageLang = page.fr;
        } else if (language == "en") {
            var pageLang = page.en;
        }

        //I reformat the filename using a regular expression.
        //This is done because the filenames in the "pages" folder are .js,
        //and I need .html files for the output.

        var r = /([^ :\\/]*?)(?:\.([^  :\\/.]*))?$/;

        var formattedName = files[i].match(r)[1];

        //We form the full link, considering also the language argument.
        //If page.link exists, we pick it. Otherwise, we build the link.
        var link = (page.link || "./" + language + "/" + formattedName + ".html");
```

```
        //Next, we need to build the divs containing the title and description of the portfolio item.
        var title = pageLang.title;
        var description = pageLang.description;

        var itemDiv = `<div class = "portfolio-item"><div class = "thumbnail">
        <a href="${link}"><img src="../images/${formattedName}.jpg"></a></div>
        <div class = "portfolio-description">
        <h2>${title}</h2><p>${description}</p></div>`;

        mosaic = mosaic + itemDiv;
    }
    //Closing the div of the mosaic
    mosaic = mosaic + `</div>`;
}
```

Je n'aime pas vraiment cette immense fonction un peu désordonnée. Peut-être devrais-je apprendre des éléments de programmation fonctionnelle pour améliorer mon code.

Voici la fonction *generatePage()* :

```
function generatePage(file) {
    var header = generateHeader(file);

    var footer = ``;

}
```

4 Création des liens de navigation

La création des liens de navigation sera clairement la partie la plus complexe de mon système.

5 Questions ouvertes

Comment ferais-je pour trier les divers éléments du portfolio ? C'est une chose fondamentale. Pour l'instant, mon système ne fera que créer chaque élément du portfolio en ordre alphabétique. Il me faut absolument un système pour trier ces éléments à ma volonté.

Devrais-je avoir un fichier qui contient plusieurs pages de mon portfolio et optionnellement des liens vers des fichiers externes qui définiraient des pages de portfolio plus complexes ? Je songe au fait que de nombreux items de mon portfolio pourrait être hébergés sur d'autres serveurs, ce qui rendrait l'acte de créer un fichier JavaScript pour définir chacun de ces items un peu redondant.

Une idée de réponse à la première question : je pourrais utiliser des nombres au début de mes noms de fichiers, comme *0001-dunes.js* et ensuite mon programme utiliserait ces nombres pour classer les items du portfolio mais ignorerait ces nombres lors de la capture du nom de fichier. *0001-dunes.js*

deviendrait *dunes.html*.

Ce n'est pas une bonne solution. Le mieux ce serait d'avoir une liste quelque part qui contiendrait les items que je veux afficher dans l'ordre. Ce système me permettrait d'ajouter aisément un item en début de liste sans modifier les items suivants. Il me permettrait aussi d'ignorer des items à volonté. Je pourrais créer un système qui prendrait cette liste et chercherait chacun des items tout d'abord dans un fichier commun comme *portfolio.json* et ensuite dans le dossier *pages*.