

Préparer son projet JS

By [@e-books](#)

Table of Contents

Introduction

1. Les bases

- 1.1 Squelette de projet : backend
- 1.2 Squelette de projet : frontend
- 1.3 Livraison

2. Les tests

3. Faites vos outils

- 3.1 Mon 1er outil npm
- 3.2 Un peu d'interactivité

Préparer son projet JS

par [@k33g_org](#)

J'ai prévu d'écrire pas mal d'articles et de tutos sur différents outils et frameworks, notamment autour de javascript. Que ce soit pour du tuto, de la démo, du POC ou de la réalisation "pour de vrai", le besoin d'industrialiser ses projets javascript, ne serait ce que pour éviter les tâches répétitives, mais aussi pour faire plus "propre" et se faciliter la vie, se fait sentir rapidement. Donc fini les downloads de librairies à la main, l'absence de gestion de version, etc. ... Sans vouloir utiliser toute la batterie d'outils disponibles (on ne peut pas tout connaître, et au bout d'un moment c'est improductif), il y a un minimum syndical que l'on peut mettre en œuvre sans trop d'efforts pour des apports significatifs.

Pourquoi ce tuto?

Je m'aperçois autour de moi (mais même moi) que finalement à part les "purs fronts", les développeurs utilisent très peu les outils d'intégration du monde javascript à leur disposition, voire même ne les connaissent pas.

Contributions

Si vous souhaitez contribuer, faire des remarques, ne pas être d'accord, etc. ... N'hésitez pas à utiliser les fonctionnalités de contribution de **GitHub** et à faire des PR, des issues, etc. ...

Bonne lecture.

Les bases

Les bases : Objectifs

Je voudrais pouvoir disposer d'un modèle de projet (ou squelette) qui me permette de démarrer ma production rapidement. Et que ce modèle de projet soit "duplicable" facilement. Je vais partir d'une webapp avec un front-end en javascript (pour l'exemple j'aurais de l'**Angular** pour préparer mes futurs articles) et un back-end node.js avec du **Express.js**, mais cela peut se décliner avec d'autres frameworks, s'utiliser uniquement avec la partie front, etc. ...

Pré-requis

- Avoir installé node.js
- Avoir installé npm
- Pour le reste nous verrons au fur et à mesure

Créer le squelette de projet

Créez une arborescence de projet de ce type avec (après vous pouvez adapter) un fichier **app.js** qui contiendra votre code applicatif, **index.html** votre page web, **main.js** sera le code "côté serveur".

```
skeleton/
└── public.src/
    ├── js/
    │   └── app.js
    └── index.html
└── main.js
```

Déclarer les librairies "back-end"

A la racine de skeleton, créer un fichier **package.json** :

```
skeleton/
└── public.src/
    ├── js/
    │   └── app.js
    └── index.html
└── main.js
└── package.json
```

Avec le contenu suivant :

```
{
  "name": "skeleton",
  "description" : "skeleton project",
  "version": "0.0.0",
  "dependencies": {
    "express": "4.1.x",
    "body-parser": "1.0.2"
  }
}
```

Nous venons de définir quelles étaient les librairies dont nous avions besoin côté serveur. Maintenant pour les télécharger/installer c'est simple, tapez dans un terminal, la commande :

```
npm install
```

En fonction de votre bande passante, cela prendra plus ou moins de temps, mais **npm** va télécharger les dépendances dans un répertoire `node_modules`, vous devriez donc avoir l'arborescence suivante :

```
skeleton/
└── node_modules/
    ├── express/
    │   └── body-parser/
    ├── public.src/
    │   ├── js/
    │   │   └── app.js
    │   └── index.html
    └── main.js
    └── package.json
```

Vous noterez que vous pouvez définir dans le fichier `package.json` les versions dont vous avez besoin et les figer. Cela peut vous éviter de mauvaises surprises lors d'un changement de version de framework.

Un bout de code serveur pour vérifier que tout fonctionne

Dans le fichier `main.js`, saisissez ceci :

```
var express = require('express')
, http = require('http')
, bodyParser = require('body-parser')
, app = express()
, http_port = 3000;

app.use(express.static(__dirname + '/public.src'));
app.use(bodyParser());

app.get("/hello", function(req, res) {
  res.send("<h1>Hello</h1>");
});

app.listen(http_port);
console.log("Listening on " + http_port);
```

Deux bouts de codes clients pour vérifier que tout fonctionne

Dans le fichier `app.js`, saisissez ceci :

```
document.querySelector("hello").innerHTML = "Hello World!";
```

Dans le fichier `index.html`, saisissez ceci :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>skeleton</title>
</head>
<body>
  <!-- Note de l'auteur
      certains n'aiment pas les tags personnalisés
      à l'approche des WebComponents, je pense qu'il faut commencer
```

```
à penser et apprendre à bosser ensemble autrement  
-->  
<hello></hello>  
<script src="js/app.js"></script>  
  
</body>  
</html>
```

Vérification

Il suffit de lancer :

```
node main.js
```

Puis d'ouvrir dans votre navigateur :

- <http://localhost:3000> qui vous chargera la page `index.html` avec le message "Hello World!"
- <http://localhost:3000/hello> qui va appeler notre route express `hello` et qui nous affichera un "gros" **Hello**.

Rien de transcendant, mais ça a le mérite de nous permettre de vérifier que nous avons bien tout installer correctement.

Remarque avant de passer à la suite

Nul besoin de vous "trimbaler" le répertoire `node_module` avec tout son contenu, vous pourrez grâce à `package.json` tout re-télécharger. A moins de savoir à l'avance que vous allez faire une démo dans un endroit sans connexion web.

Déclarer les librairies "front-end"

Pour faire une jolie webapp et dans tout faire "from scratch", nous allons avoir besoin de framework(s) javascript, de css, etc. ... Pour cela nous avons besoin d'installer **Bower** <http://bower.io/>, c'est un utilitaire (par Twitter) qui permet de télécharger différents projets de frameworks javascript et css. Bower c'est un peu comme **npm** mais plutôt dédié front.

Pour l'installer une fois pour toute :

```
npm install -g bower
```

le paramètre `g` signifie que **Bower** sera installé de façon "globale" donc accessible de n'importe où dans vos répertoires (donc pas dans installé dans le répertoire `node_module`). *Sous Ubuntu je suis obligé de faire sudo npm install -g bower*.

Ensuite créez à la racine de `skeleton` un fichier `bower.json` :

```
{  
  "name": "skeleton",  
  "version": "0.0.0",  
  "dependencies": {  
    "jquery": "2.1.1",  
    "bootstrap": "3.1.1",  
    "angular": "1.2.16",  
    "angular-resource": "1.2.16"  
  }  
}
```

```
}
```

Ensuite je souhaite que **Bower** me télécharge les composants (frameworks) dans le répertoire `public.src/bower_components`, donc créez, toujours à la racine de `skeleton` un fichier `.bowerrc` :

```
{
  "directory": "public.src/bower_components"
}
```

Et puis lancer la commande :

```
bower install
```

Vous obtiendrez ceci :

```
skeleton/
├── node_modules/
│   ├── express/
│   └── body-parser/
├── public.src/
│   ├── bower_components/
│   │   ├── angular/
│   │   │   ├── angular-resource/
│   │   │   ├── bootstrap/
│   │   │   └── jquery/
│   ├── js/
│   │   └── app.js
│   └── index.html
└── main.js
└── package.json
└── bower.json
└── .bowerrc
```

Ensuite nous allons faire un tout petit peu d'Angular pour valider que tout est installé correctement.

Modification de la page `index.html`

Modifions notre page `index.html` pour référencer nos css et frameworks javascript :

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>skeleton</title>
  <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap.min.css">
  <link rel="stylesheet" href="bower_components/bootstrap/dist/css/bootstrap-theme.min.css">
</head>
<body ng-app="skeletonApp">

  <hello-world ng-controller="HelloCtrl" class="container">
    <h1>{{message}}</h1>
  </hello-world>

  <script src="bower_components/angular/angular.min.js"></script>
  <script src="bower_components/angular-resource/angular-resource.min.js"></script>
  <script src="bower_components/jquery/dist/jquery.min.js"></script>
```

```
<script src="bower_components/bootstrap/dist/js/bootstrap.min.js"></script>
<script src="js/app.js"></script>

</body>
</html>
```

Modification de `app.js`

Remplacer le code de `app.js` par celui-ci:

```
var skeletonApp = angular.module("skeletonApp", []);
var HelloCtrl = skeletonApp.controller("HelloCtrl", function($scope) {
  $scope.message = "Hello World!";
});
```

Vous pouvez relancer `node main.js` et ouvrir <http://localhost:3000> et vous obtiendrez un splendide **Hello World!**.

Tout fonctionne, vous pouvez commencer à coder!

Mais, mais, mais ...

"Livrer" votre application

Le Front

Grunt

Je ne sais pas si vous avez remarqué, (si ce n'est pas le cas allez vérifier), mais **Bower** ne se contente pas de télécharger simplement la librairie javascript ou la feuille de styles css, il ramène "tout le monde", c'est à dire l'ensemble des codes sources, documentation, tests, ... Et nous n'avons pas besoin de tout ça pour que notre application fonctionne en production. Donc vous avez la solution "à la main" ou la solution "industrialisée" avec **Grunt**, un lanceur de tâches <http://gruntjs.com/>.

Commençons par installer **Grunt** de manière global (accessible partout comme pour Bower).

```
npm install -g grunt
```

Sous Ubuntu, je suis obligé de pré fixé par un `sudo`.

Ensuite dans le fichier `package.json` ajoutez une nouvelle dépendance, le plugin grunt "grunt-contrib-copy" <https://github.com/gruntjs/grunt-contrib-copy> qui sert à copier des fichiers d'une destination vers une autre.

```
{
  "name": "skeleton",
  "description" : "skeleton project",
  "version": "0.0.0",
  "dependencies": {
    "express": "4.1.x",
    "body-parser": "1.0.2",
    "grunt-contrib-copy": "0.5.0"
  }
}
```

et lancez un :

```
npm update
```

un `npm install` aurait aussi fonctionné

Vous obtenez donc 2 nouveaux répertoires dans `node_modules` :

```
skeleton/
└── node_modules/
    ├── express/
    ├── body-parser/
    ├── grunt/
    └── grunt-contrib-copy/
```

Gruntfile.js

Ensuite créez un fichier `Gruntfile.js` à la racine de `skeleton` avec le contenu suivant :

```
module.exports = function(grunt) {

  grunt.initConfig({
    copy: {
      main: {
        files:[
          {expand: true, cwd: "public/src/bower_components/bootstrap/dist", src: ['fonts/**'], dest: 'public/js/vendors/bootstrap'},
          , {expand: true, cwd: "public/src/bower_components/bootstrap/dist/", src: ['js/bootstrap.min.js'], dest: 'public/js/vendors/bootstrap'}
          , {expand: true, cwd: "public/src/bower_components/bootstrap/dist/", src: ['css/bootstrap.min.css'], dest: 'public/js/vendors/bootstrap'}
          , {expand: true, cwd: "public/src/bower_components/bootstrap/dist/", src: ['css/bootstrap-theme.min.css'], dest: 'public/js/vendors/bootstrap'}
          , {expand: true, cwd: "public/src", src: ['js/**'], dest: 'public'}
          , {expand: true, cwd: "public/src/bower_components/jquery/dist/", src: ['jquery.min.js'], dest: 'public/js/vendors'}
          , {expand: true, cwd: "public/src/bower_components/angular/", src: ['angular.min.js'], dest: 'public/js/vendors'}
          , {expand: true, cwd: "public/src/bower_components/angular-resource/", src: ['angular-resource.min.js'], dest: 'public/js/vendors'}
        ]
      },
      index : {
        expand: true, cwd: "public/src", src: ['index.html'], dest: 'public',
        options: {
          process: function (content, path) {
            return content
              .replace(/bower_components/g, "js/vendors")
              .replace(/bootstrap\dist/g, "bootstrap")
              .replace(/jquery\dist//g, "")
              .replace(/angular\//g, "")
              .replace(/angular-resource\//g, "")
            }
        }
      }
    }
  });
}
```

```
        grunt.loadNpmTasks('grunt-contrib-copy');
    }
```

Dans ce fichier, je demande à **Grunt** de copier les fichiers qui sont dans `public.src/bower_components` dans le répertoire `public/js/vendors` avec la tâche `main`, je déplace aussi tous les fichiers présents dans `public.src/js` vers `public/js`, ensuite avec la tâche `index`, je demande à **Grunt** de déplacer `index.html` dans `public` mais aussi d'en modifier le contenu car j'ai changé les chemins des ressources en les déplaçant.

Ensuite j'enregistre ma tâche : `grunt.loadNpmTasks('grunt-contrib-copy')`

Et maintenant, dès que je veux livrer, je n'ai plus qu'à lancer la commande:

```
grunt copy
```

Et vous obtiendrez l'arborescence suivante:

```
skeleton/
├── node_modules/
│   ├── express/
│   └── body-parser/
└── public/
    ├── js/
    │   ├── vendors/
    │   │   ├── angular.min.js
    │   │   ├── angular-resource.min.js
    │   │   ├── bootstrap/
    │   │   │   ├── css/
    │   │   │   ├── fonts/
    │   │   │   └── js/
    │   │   └── jquery.min.js
    │   └── app.js
    └── index.html
└── public.src/
    ├── bower_components/
    │   ├── angular/
    │   ├── angular-resource/
    │   ├── bootstrap/
    │   ├── jquery/
    │   └── js/
    └── index.html
└── main.js
└── package.json
└── bower.json
└── .bowerrc
```

Le Back

Si vous vous souvenez, dans le code serveur (`main.js`), on définit l'emplacement des fichiers statiques de la façon suivante:

```
app.use(express.static(__dirname + '/public.src'));
```

Et l'on voudrait pouvoir facilement tester la version de développement `/public.src` et la version de "prod" `/public`. Pour cela modifier le code de `main.js` de la façon suivante :

Remplacez:

```
app.use(express.static(__dirname + '/public.src'));
```

Par:

```
if (process.env.NODE_ENV == "dev") {  
  app.use(express.static(__dirname + '/public.src'));  
} else {  
  app.use(express.static(__dirname + '/public'));  
}
```

Et maintenant si vous voulez tester le front "développement", vous lancez node de cette manière :

```
NODE_ENV=dev node main.js
```

Et pour la "production" on reste sur :

```
node main.js
```

Voilà, c'est terminé pour cette partie. Alors c'est une base. Cela peut être fait différemment, s'améliorer, etc. ... J'attends vos propositions :)

Vous pourrez trouver les code source ici <https://github.com/web-stacks/js-quick-start>.

Stay tuned for the next episode ...

Les tests

Les tests

W.I.P.

Faites vos outils

Faites vos outils avec npm et node

Lorsque je code, j'ai souvent des tâches répétitives (quel que soit le langage), ie:

- écrire des modèles et des collections Backbone
- écrire des modèles Play
- écrire des routes pour Express.js
- ...

Nul doute, que vous avez énormément d'autres exemples en tête. L'outil du moment pour régler ce type de problème est [Yeoman](#) (oui il y en a d'autres, mais Yeoman c'est "hype" ;)) et ensuite, ou vous avez de la chance, ou il faut écrire le plugin dont vous avez besoin. Dans certains cas, surtout si vous n'avez pas besoin de quelque chose de complexe, vous pouvez rapidement écrire un utilitaire avec uniquement node et npm.

Spécifications (par exemple)

Je passe mon temps à écrire des modèles et des collections Backbone qui ressemblent à ceci :

```
/*--- Human Model ---*/
var HumanModel = Backbone.Model.extend({
  defaults : function () {
    return {};
  },
  urlRoot : "humans"
});

/*--- Humans Collection ---*/
var HumansCollection = Backbone.Collection.extend({
  url : "humans",
  model: HumanModel
});
```

Et j'aimerais aller plus vite, juste taper `bb Human` et obtenir mon fichier généré.

Juste fais-le ...

package.json

Commencez par créer un répertoire `bbtools`. Ensuite dans ce répertoire, créez un fichier `package.json` avec le contenu suivant :

```
{
  "name": "bbtools",
  "version": "0.0.0",
  "bin": { "bb": "bb.js" },
  "dependencies": {
    "underscore": "1.6.0"
  }
}
```

L'objectif étant de disposer d'une commande `bb` qui exécutera le fichier `bb.js`. Notez `"underscore": "1.6.0"`, nous allons nous servir des capacités de "templating" de la librairie Underscore

Notre template

Créez (dans le même répertoire) un fichier `bb.tpl` avec le contenu suivant :

```
/*--<%= modelName %> Model ---*/
var <%= modelName %>Model = Backbone.Model.extend({
  defaults : function () {
    return {}
  },
  urlRoot : "<%= modelName.toLowerCase() %>s"
});

/*--<%= modelName %>s Collection ---*/
var <%= modelName %>sCollection = Backbone.Collection.extend({
  url : "<%= modelName.toLowerCase() %>s",
  model: <%= modelName %>Model
});
```

Le code applicatif

Créez (dans le même répertoire) un fichier `bb.js` avec le contenu suivant :

```
#!/usr/bin/env node

var fs = require('fs');
var _ = require('underscore');

require.extensions['.tpl'] = function (module, filename) {
  module.exports = fs.readFileSync(filename, 'utf8');
};

var tpl = _.template(require("./bb.tpl"));
var modelName = process.argv[2]

fs.writeFileSync(
  process.cwd() + "/" + modelName + ".js"
, tpl({modelName :modelName})
);
```

Explications

- `#!/usr/bin/env node` : rendra le fichier exécutable
- `_.template(require("./bb.tpl"))` va lire notre fichier `bb.tpl` et le transforme en objet template pour Underscore
- `process.cwd()` permet de connaître le répertoire d'où l'on appelle la commande `bb`
- `process.argv[2]` permet d'obtenir le 1er argument passé à la commande `bb` (comme par exemple `bb Human`)
- `tpl({modelName :modelName })` : on "explique" au template que l'on passe le contenu de `modelName` à `modelName` définie dans le template

Enregistrez votre outil

Pour que votre nouvelle commande soit disponible "partout" (pouvoir l'appeler de n'importe quel répertoire), tapez la commande suivante (on est toujours dans notre répertoire de développement) :

```
npm link
```

Si ce n'est pas déjà fait, cela va télécharger les dépendances nécessaires (underscrore dans notre cas), puis créer un "lien symbolique" vers votre fichier `bb.js`

Maintenant, de n'importe où, dans une console (terminal), vous pouvez taper `bb Animal` et cela vous générera votre fichier `Animal.js` avec votre modèle et votre collection.

Voilà, c'est très simple et utile. Si vous sentez que votre outil devient une usine à gaz, passez quand même à Yeoman ;).

Un peu d'interactivité dans vos outils node

Je vous expliquais il y a peu comment construire ses propres outils en ligne de commande avec node et npm : <http://k33g.github.io/2014/05/09/NPM-NODE-CLI.html>. Nous allons voir aujourd'hui comment ajouté un peu d'interactivité et de couleur à tout ça. Donc, il faut repartir du même projet (donc lire l'article).

2 nouveaux modules : prompt et colors.js

Lorsque je parle d'interactivité, je veux dire, que mon "programme" va "me poser" des questions auxquelles je vais devoir répondre. Pour cela je vais utiliser **prompt** (<https://github.com/flatiron/prompt>), et pour les couleurs, le bien nommé **colors.js** (<https://github.com/Marak/colors.js>). Modifiez donc votre fichier `package.json` de la manière suivante :

```
{
  "name": "bbtools",
  "version": "0.0.0",
  "bin": { "bb": "bb.js" },
  "dependencies": {
    "underscore": "1.6.0",
    "prompt": "0.2.12",
    "colors": "0.6.2"
  }
}
```

Puis faites moi un petit `npm install` pour télécharger les 2 nouvelles dépendances.

Modification de `bb.js`

Maintenant, plutôt que de passer le nom du modèle en argument, je souhaite que ce soit l'outil qui demande le nom du modèle, je voudrais aussi pouvoir saisir les valeurs par défaut du modèle. Nous allons donc modifier le code du fichier `bb.js` de la façon suivante :

Ajout des référence à **prompt** et **colors**:

```
#!/usr/bin/env node

var fs = require('fs')
, _ = require('underscore')
, prompt = require('prompt')
, colors = require('colors');
```

Ici on ne change rien:

```

require.extensions['.tpl'] = function (module, filename) {
  module.exports = fs.readFileSync(filename, 'utf8');
};

var tpl = _.template(require("./bb.tpl"));

```

Ajout d'un "schema" de saisie pour prompt:

```

var schema = {
  properties: {
    model_name: {
      description: 'Enter model name'.green,
      pattern: /^[a-zA-Z]+$/,
      message: 'Model name must be only letters'.inverse.red,
      required: true
    },
    default_values: {
      description: 'Default values'.blue,
      default: ""
    }
  }
};

```

Vous notez que vous avez "la main" sur ce que vous pouvez ou n pouvez pas saisir. Ensuite, notez aussi ceci : 'Enter model name'.green ou 'Model name must be only letters'.inverse.red et encore 'Default values'.blue , c'est **colors.js** qui permet en augmentant les strings de définir les couleurs d'affichage des textes dans votre console ou terminal.

Utilisation du "schema" de saisie:

Il suffit de passer à `prompt.get()` votre `schema` ainsi que la fonction à exécuter avec comme paramètres les erreurs (`err`) et les données saisies (`result`). `result` est un objet qui a pour propriétés les propriétés de `schema.properties` donc `model_name` et `default_values`

```

prompt.get(schema, function (err, result) {

  if(err) {
    console.log("Bye!".rainbow);
    process.exit(1);
  } //Ctrl+c to exit

  console.log(JSON.stringify(result).cyan);

  fs.writeFileSync(
    process.cwd() + "/" + result.model_name + ".js"
    , tpl({modelName: result.model_name, defaultValues: result.default_values})
  );

});

```

Modification du template `bb.tpl`

Nous modifions le template pour prendre en compte la nouvelle valeur `defaultValues: result.default_values` :

```

/*--- <%= modelName %> Model ---*/

```

```

var <%= modelName %>Model = Backbone.Model.extend({
  defaults : function (){
    return {<%= defaultValues %>}
  },
  urlRoot : "<%= modelName.toLowerCase() %>s"
});

/*--- <%= modelName %>s Collection ---*/
var <%= modelName %>sCollection = Backbone.Collection.extend({
  url : "<%= modelName.toLowerCase() %>s",
  model: <%= modelName %>Model
});

```

Utilisation

Dans un terminal, tapez `bb` et répondez (trompez vous aussi) :

```

2. node
k33g_org@k33g-orgs-MacBook-Pro:~$ bb
prompt: Enter model name: Hum4n
error: Invalid input for Enter model name
error: Model name must be only letters
prompt: Enter model name: 

```

```

2. bash
Last login: Tue May 13 16:27:08 on ttys012
k33g_org@k33g-orgs-MacBook-Pro:~$ bb
prompt: Enter model name: Human
prompt: Default values: firstName:"John", lastName:"Doe"
{"model_name":"Human","default_values":"firstName:\"John\"", lastName:\"Doe\""}
k33g_org@k33g-orgs-MacBook-Pro:~$ 

```

Avec le code généré :

```

/*--- Human Model ---*/
var HumanModel = Backbone.Model.extend({
  defaults : function (){
    return {firstName:"John", lastName:"Doe"}
  },
  urlRoot : "humans"
});

```

```
/*--- Humans Collection ---*/
var HumansCollection = Backbone.Collection.extend({
  url : "humans",
  model: HumanModel
});
```

C'est tout simple, un peu plus "user friendly", et bientôt vous pourrez refaire **Yeoman** ;).