Node.js

J. ROMAGNY

Node.js permet d'utiliser JavaScript **côté serveur** comme on le ferait avec PHP ou Asp.Net. Ses qualités : légèreté, portabilité, rapidité.

« JavaScript Everywhere » (client, serveur, db)

MEAN = **M**ongoDB, **E**xpress, **A**ngularJS, **N**ode.js (MEAN.JS)

1. Conseils

La base d'un projet Node.js c'est un serveur (« app.js » ou « server.js ») + un package (« package.json ») avec les propriétés et dépendances du projet. Après on peut améliorer la structure du projet (avec express, le pattern MVC, vues et vues partielles, moteur de vue, config, routes, services, etc.)

- ✓ Utiliser de préférence **Visual Studio** avec l'extension.

Express

- ✓ **Express generator** peut être utile pour créer un projet de départ

(Installer express-generator en global) Naviguer jusqu'au dossier où créer le projet puis

```
express –e nomprojet
```

NPM

- ✓ « **npm init** » pour **créer le package** (« package.json ») du projet (Naviguer jusqu'au dossier du projet puis « npm init »)
- ✓ **Installer** les **modules avec --save** pour ajouter la dépendance au package (« package.json »)

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

- ✓ **Créer des vues partielles** (header, footer, etc.) pour éviter de répéter le même code entre pages.
- ✓ Ranger les **ressources** (images, feuilles de Styles, etc.) dans un **dossier « public »**

MongoDB

- ✓ Utiliser **Robomongo** pour gérer ses bases de données.

Démarche pour tester le site

1. Naviguer jusqu'au dossier de MongoDB et lancer mongod en indiquant le chemin vers la base de données

```
mongod –dbpath ./data
```

2. Ouvrir une seconde invite de commande, naviguer jusqu'au dossier du projet (ou depuis le menu contextuel de Visual Studio) et lancer le serveur ou Debug

```
node app.js
```

2. Outils de développement

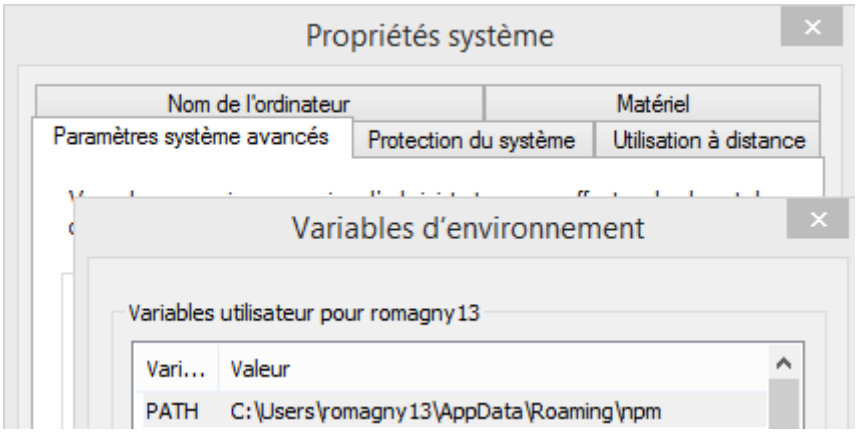
- ✓ Brackets,WebStorm, Sublime Text, Atom, etc.
- ✓ Visual Studio avec NodeJS tools for Visual Studio : ajout d'un onglet de Templates « nodejs », ouverture de la console depuis le menu contextuel du projet et fenêtre « Node.js interactive window »)
- ✓ Visual Studio Code (présentation par John Papa)

3. Installation de Node.js

➔ Télécharger et installer la version désirée (Windows, Linux, Mac).

Lancer une invite de commande et taper une commande comme « node » ou « npm init » pour vérifier s'il n'y a pas d'erreurs.

En cas d'erreur « npm n'est pas reconnu en tant que commande interne » Vérifier la variable d'environnement « PATH »



L'installation se fait dans le répertoire « C:\Program Files\nodejs » pour Windows.

Se **former** avec la « NODESCHOOL » **Tutoriels**

La documentation **API**

4. Projet minimum et bases

a. Création d'un serveur

1. Créer un répertoire pour le projet et créer un serveur (un fichier « server.js » ou « app.js » par exemple)

```
var http = require('http');

var html = "<!DOCTYPE html><html><head><meta charset=\"utf-8\" /><title>Demo
NodeJS</title></head><body><h1>D&eacute;monstration NodeJS</h1></body></html>";

var server = http.createServer(function (req, res) {
    res.writeHead(200, { "Content-Type": "text/html" });
    res.write(html);
    res.end();
});
server.listen(8080);

console.log("listen on port 8080");
```

Chargement du module « http »

Création du serveur

En-tête

« req » pour request,
« res » pour response

Sortie (texte, html, etc.)

Types de contenus :

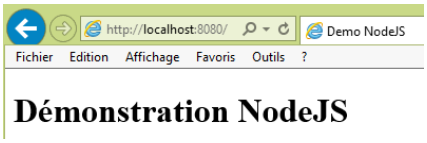
- ✓ Texte : text/plain
- ✓ HTML : text/html
- ✓ CSS : text/css
- ✓ Image : image/jpeg
- ✓ Vidéo : video/mp4
- ✓ Zip : application/zip
- ✓ Etc.

Utiliser le port 3000 ou 8080 pour des tests en local

2. Lancer un serveu
   Ouvrir une invite de commande, naviguer jusqu'au dossier du
   (Commande « cd ») et lancer le serveur

```
node server.js
```

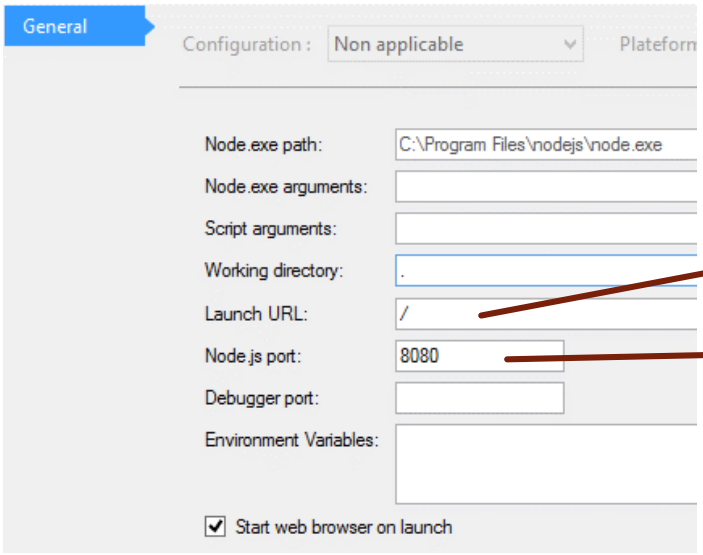
3. Tester la page dans le navigateur. Aller à « http://localhost:80



➔ Pour arrêter un serveur : **CTRL + C**

b. Déboguer avec Visual Studio 2013

Dans les propriétés du projet. Le Debug marche avec IE, Chrome, l'inspecteur de page, etc.



c. URL et paramètres

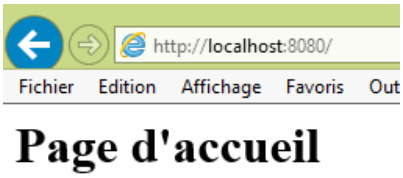
Afficher les pages selon l'url.

```
var http = require('http');
var url = require('url');

var server = http.createServer(function (req, res) {

    var page = url.parse(req.url).pathname;
    console.log(page);

    res.writeHead(200, { "Content-Type": "text/html" });
    if (page == '/') {
        res.write('<h1>Page d\'accueil</h1>');
    }
    else if (page == '/contact') {
        res.write('<h1>Page de contact</h1>');
    }
    else {
        res.write('<h1>Erreur 404</h1><h2>Page non trouv&eacute;e.</h2>');
    }
    res.end();
});
server.listen(8080);
```

Chargement du module « url »

Récupération du chemin de la page demandée (après l'url du site)

Affiche dans la console de la navigation



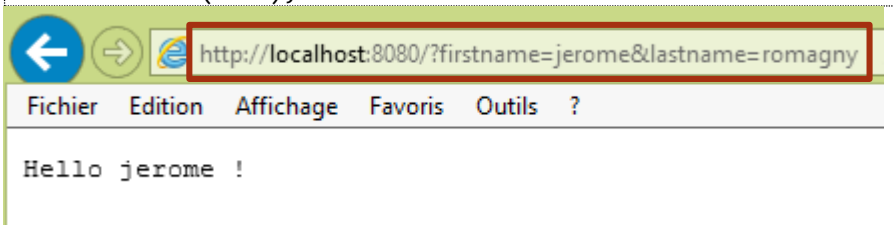
➔ Paramètres d'url (**querystring**)

```
var http = require('http');
var url = require('url');
var querystring = require('querystring');

var server = http.createServer(function (req, res) {
    var params = querystring.parse(url.parse(req.url).query);

    res.writeHead(200, { "Content-Type": "text/plain" });

    if ('firstname' in params && 'lastname' in params) {
        res.write('Hello ' + params['firstname'] + ' !');
    }
    else {
    }
    res.end();
});
server.listen(8080);
```

Besoin du module « querystring » pour récupérer les paramètres dans l'url

Récupération des paramètres passés dans un tableau

Accès aux membres du tableau de paramètres



d. Modules

On utilise « **require » pour charger un module :**

- en passant le nom du module
- avec le chemin vers le module personnalisé pour charger un module.

➔ Modules de base (accessibles sans à avoir à les importer)

Exemple « http »

```
var http = require('http');
```

➔ Modules installés (dans « node-modules ») avec npm (le code ne change pas)

➔ Créer ses modules personnalisés

Les différentes façons de créer un module

On crée un nouveau fichier javascript

```
exports.methodOne = function () {
};
exports.methodTwo = function (app) {
};
```

Méthodes et variables déclarées avec « exports ». On référence les modules utilisés en haut, on pourrait également avoir des variables

```
(function (monmodule) {

    monmodule.methodOne = function () {
    };
    monmodule.methodTwo = function () {
    };

})(module.exports);
```

```
monobjet ={}
monobjet.methodOne = function () {
};
monobjet.methodTwo = function () {
};
module.exports = monobjet ;
```

Export d'objet

Utiliser le module créé

```
var monmodule = require('./monmodule.js');
//ou
var monmodule = require('./monmodule');

monmodule.methodOne() ;
```

Chemin vers le module. On peut indiquer l'extension du fichier (*.js)

Module avec passage de paramètre

```
module.exports = function (app) {
  app.use(…);
}
```

Passage de paramètre et utilisation

Utilisation

```
var monmodule = require('./monmodule')(app);
```

Ou

```
(function (monmodule) {

    monmodule.methodOne = function (app) {

    };

})(module.exports);
```

Utilisation

```
var monmodule = require('./monmodule');
monmodule.methodOne(app) ;
```

e. Evénements

Ecouter et émettre des événements

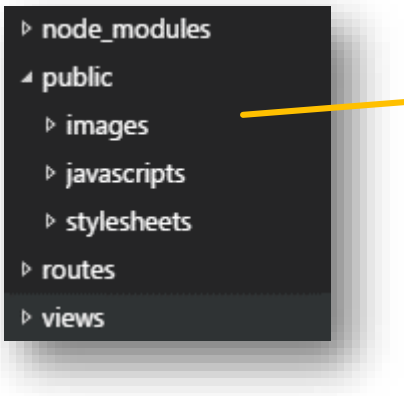
```
var eventEmitter = require('events').EventEmitter;
var myEvent = new eventEmitter();

myEvent.on('myEventEmittted', function (message) {
    console.log(message);
});

myEvent.emit('myEventEmittted', 'Ev&eacute;nement d&eacute;clench&eacute;');
```

Besoin du module « events »

On écoute l'événement, avec fonction

On déclenche l'événement et on passe un paramètre à la fonction appelée

f. Ressources (dossier « public »)



Créer un dossier « public » et y ajouter les Ressources

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

app.set("view engine", "vash");
app.use(express.static(__dirname + "/public"));
```

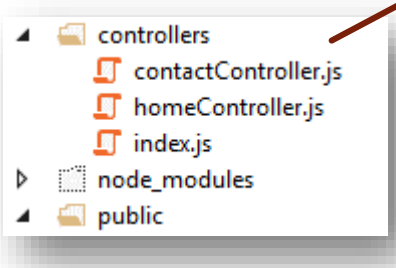
Référencer le répertoire « public » de l'application

Utilisation d'une ressource

```
<link href="/css/default.css" rel="stylesheet" />
```

Dans une vue par exemple on référence ensuite la feuille de styles, inutile d'indiquer le répertoire « public »

g. Contrôleurs



Créer un dossier « controllers »

Création de contrôleur

Exemple « homeController » pour la page d'accueil du site

```
(function (homeController) {

    homeController.init = function (app){

        app.get('/', function (req, res) {
            res.render('index');
        });
    }
})(module.exports);
```

Utilisation du contrôleur

```
var express = require('express');
var app = express();
var homeController = require('./homeController.js');

homeController.init(app);

app.listen(8080);
```

Référencement du contrôleur

Appel de la méthode « init » du contrôleur et passage de « app » en paramètre

Créer un index listant les contrôleurs

```
(function (controllers) {

    var homeController = require('./homeController.js');
    var contactController = require('./contactController.js');

    controllers.init = function (app) {
        homeController.init(app);
        contactController.init(app);
    }

})(module.exports);
```

Utilisation

```
var express = require('express');
var app = express();
var controllers = require('./controllers');

app.set("view engine", "vash");

controllers.init(app);

app.listen(8080);
```

On indique le dossier contenant les contrôleurs

Initialisation des contrôleurs en appelant la méthode

5. Tools

Un petit résumé :

- Avec **PHP** (Symfony, etc.) on utilisera plutôt « **composer** »
- Avec **Angular** : on utilisera plutôt « **Bower** » (les librairies seront ajoutées dans un dossier « bower_components »
- Avec **Node** on utilisera « **NPM** » (les librairies seront ajoutées dans un dossier « node_modules »)

Commandes utiles :

« cd » pour naviguer dans les dossiers
Glisser les répertoires dans l'invite de commande
« dir » lister les dossiers et fichiers du répertoire courant
« md » créer un dossier
touche « espace » permet de répéter la dernière commande entrée.
« cls » pour nettoyer la console

a. NPM

Node Package Manager permet d'installer et publier des **modules**.

Créer le « package.json » d'un projet

Naviguer vers le dossier du projet puis

```
npm init
```

Répondre aux questions … puis « package.json » est généré

Installer des modules

Installation globale

```
npm install nommodule –g
```

Installation locale à un projet

Naviguer vers le dossier du projet puis

```
npm install nommodule
```

Installation avec ajout de la dépendance à « package.json »

```
npm install nommodule --save
```

Installer toutes les dépendances définies dans le « package.json » d'un projet

Exemple de package.json

```
{
  "name": "NodejsDemo",
  "version": "0.0.1",
  "description": "NodejsDemo",
  "main": "server.js",
  "author": {
    "name": "romagny13",
    "email": ""
  },
  "dependencies": {
    "body-parser": "^1.12.3",
    "ejs": "^2.3.1",
    "express": "^4.12.3",
    "mongoose": "^4.0.2"
  }
}
```

Dépendances qui seront installées (« ^ » pour version

Naviguer vers le dossier du projet puis

```
npm install
```

Autres commandes utiles

Désinstaller un module (suppression dossier et dépendance dans « package.json »)

```
npm rm nomdule
```

Mettre à jour

```
npm update
```

b. Bower

Bower est un gestionnaire de dépendances. Il permet d'installer facilement des packages sans avoir à se préoccuper des dépendances. Par exemple pour utiliser Bootstrap on a besoin de jQuery, Bower se charge d'ajouter automatiquement jQuery.

Liste des packages disponibles ,Bower specs
(Bower nécessite que NodeJS et GIT soient installés)

➔ Installation de Bower en global (depuis une invite de commande)

```
npm install bower -g
```

➔ Installer Bower localement au projet avec ajout de dépendance à « package.json »

```
npm install bower --save
```

Ajout de dépendance de développement (« devDependencies » dans « package.json »)

```
npm install bower --save-dev
```

➔ Définir le dossier où doivent être installés les fichiers

Créer un fichier à la racine du projet « .bowerrc ». Exemple les fichiers seront installés dans le dossier « public/lib »

```
{
  "directory": "public/lib"
}
```

➔ Créer « bower.json »

```
bower init
```

➔ **Installer** un package

Naviguer vers le dossier du projet dans lequel installer la dépendance ou depuis l'invite de commande ouverte depuis Visual Studio

```
bower install bootstrap --save
```

Remplacer par le package désiré

Ex installation de bootstrap dans le dossier créé « AngularJSDemo » . jQuery sera installé en même temps..

➔ Mettre à jour le package

```
bower update bootstrap
```

c. Git

Installation

http://www.git-scm.com/

Utilisation de « **Git Bash** » pour les commandes

Exemple ajout de variables globales puis

Config

```
git config --global user.name "Jerome Romagny"
git config --global user.email "romagny13@....."
git config --list
```

Définir un dossier

Naviguer jusqu'au répertoire de projet « cd … » puis

```
git init
```

Commit

Valider les changements dans le dossier (ajout/ suppression de fichiers, etc.)

```
git commit -a -m "initial commit"
```

On peut également définir les fichiers à ajouter puis « com

Message décrivant la

```
git add .
git commit -m "inital commit"
```

Commandes utiles :

- « git help »
- « git status » permet de lister les changements du répertoire « fichier ajouté, supprimé » à « commit » ..
- « git log » permet de lister tous les « commit »

Ignorer des dossiers

Créer un fichier « .gitignore » à la racine du projet et indiquer les dossiers à ignorer.
Exemple ignorer les modifications du dossier « node-modules », …

```
.idea/
node_modules/
public/lib
```

Github

Cloner un projet github (copié dans le dossier défini auparavant)

```
git clone http://github.com/...git
```

Remote

```
git remote add origin https ://github.com/…git
git push
```

(Demande les identifiants à chaque fois avec « https », on peut également utiliser « ssh »)
…Puis seulement « git push » les fois suivantes.

Menu contextuel Windows

Sous Windows on a accès à quelques commandes depuis le menu contextuel

d. GruntJS 

Permet d'accélérer le développement en ayant pas à relancer à chaque fois le serveur. On peut ainsi éditer une vue, la feuille de style, etc. et rafraîchir la page sans avoir à tout relancer.

Installer **Grunt** en global avec npm

```
npm install grunt-cli -g
```

Installer **Nodemon** en local au projet

```
npm install grunt-nodemon --save-dev
```

Créer un fichier de configuration pour Grunt « gruntfile.js »

```
module.exports = function (grunt){
    grunt.initConfig({
        nodemon: {
            all: {
                script: 'server.js',
                options: {
                    watchedExtensions: ['js']
                }
            }
        }
    });
    grunt.loadNpmTasks('grunt-nodemon');
    grunt.registerTask('default', ['nodemon']);
}
```

Pour lancer le serveur il suffit désormais de taper la commande « grunt » à la place de « node server.js »

```
grunt
```

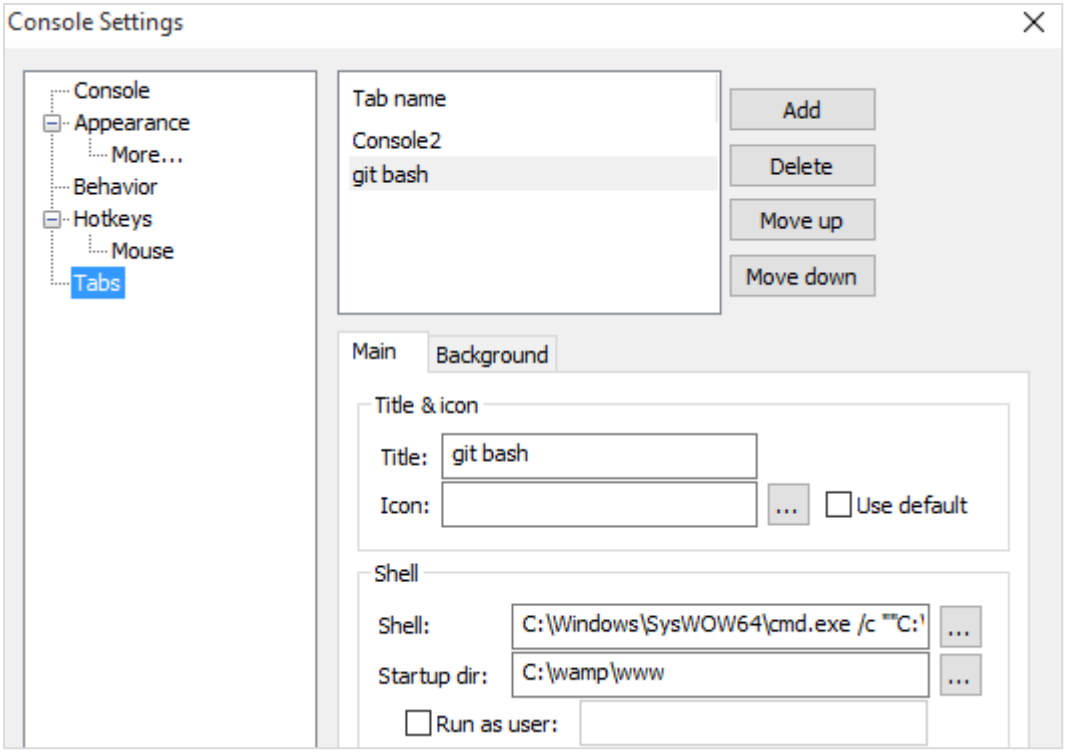
```
c:\users\donjr\documents\visual studio 2013\Projects\NodejsWebAppDemo\NodejsWebA
ppDemo>grunt
Running "nodemon:all" (nodemon) task
[nodemon] v1.3.7
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node server.js`
```

Documentation

e. Console 2 avec gitbash

Il est possible d'utiliser console 2 combiné avec gitbash (installée avec Git) pour avoir une console moins austère.

Lancer console 2 puis menu « Edit » … « Settings »… onglet « Tabs »… « Add »



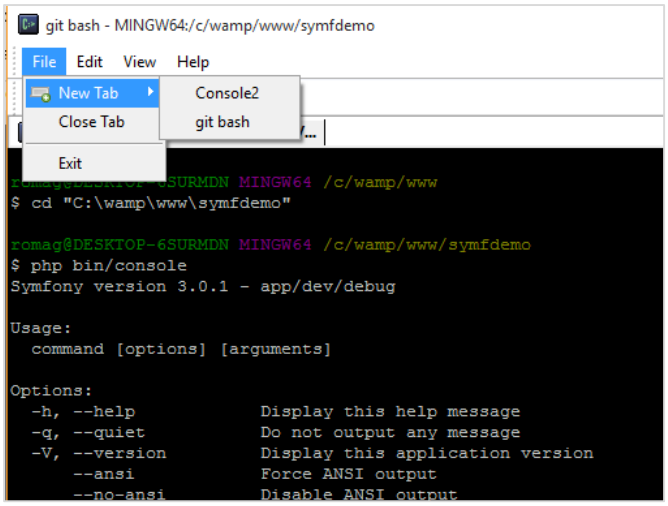
Title : « git bash » par exemple

Shell : C:\Windows\SysWOW64\cmd.exe /c ""C:\Program Files\Git\bin\sh.exe" --login -i"

Startup uri : ce que l'on veut

Le chemin vers git peut varier

On peut désormais ouvrir une nouvelle tab git bash



6. Express

Framework pour NodeJS permettant :

- de **gérer les routes**
- les **vues** (moteurs de vues)
- Fournit également des middlewares (compression, cookie-parser, cookie session, server-static, etc) 4 paramètres (« err » pour les erreurs, « req » pour la requête, « res » pour la réponse à renvoyer, « next » le callback)

Installation avec npm

```
npm install express
```

a. Créer un projet de départ avec « Express Generator »

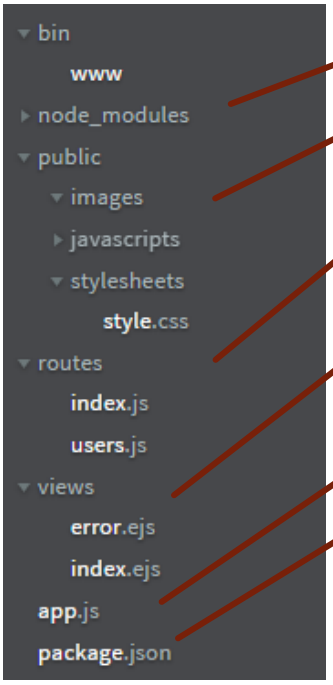
1. Installation globale d' « express generator »

```
npm install –g express-generator
```

2. Naviguer vers le dossier ou générer le projet. Par exemple « documents » (cd C:\Users\romagny13\Documents\)
3. Génération du projet

```
express –e generator-demo
```

Nom du projet



Modules installés avec npm

« public » : images, scripts (application Angular,etc.) ,feuilles

Utilise le router de express pour définir les différentes routes qui redirigent vers une vue

Vues (et vues partielles) + moteur de vue ejs

« Main » crée le serveur

Package contenant les informations du projet (auteur,etc.) ainsi que les dépendances

On navigue vers le dossier du projet et on installe les dépendances (remplacer « generator-demo » par le répertoire du

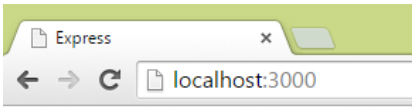
4. Installation des dépendances du projet

```
cd generator-demo && npm install
```

5. Debug

```
node ./bin/www
```

On peut se rendre à la page « http://localhost:3000/ »



Express

Welcome to Express

b. Routes

Avec « app »

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

app.set("view engine", "ejs");

app.get('/', function (req, res) {
   res.render('index');
});

app.get('/customers', function (req, res) {
  res.render('customers');
});

app.get('/customers/:id', function (req, res) {
   res.setHeader('Content-Type', 'text/html');
   res.end("<h2>Id pass&eacute ; : " + req.params.id + "</h2>");
});

app.get('*', function (req, res) {
   res.setHeader('Content-Type', 'text/html');
   res.end("<h1>Page introuvable</h1>");
});

app.listen(3000);
```

Route dynamique avec paramètre

Page 404

Changer le répertoire des vues

```
app.set('views', __dirname + 'views');
```

Avec « router »

On crée 2 fichiers dans un dossier « routes », « index.js » pour les routes commençant avec « / », un autre pour les routes commençant par « /customers »

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

router.get('/', function (req, res) {
   res.render('index');
});

router.get('*', function (req, res) {
   res.setHeader('Content-Type', 'text/html');
   res.end("<h1>Page introuvable</h1>");
});

module.exports = router;
```

« customers.js »

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

router.get('/', function (req, res) {
  res.render('customers');
});

router.get('/:id', function (req, res) {
   res.setHeader('Content-Type', 'text/html');
   res.end("<h2>Id pass&eacute ; : " + req.params.id + "</h2>");
});

module.exports = router;
```

« customers » sera ajouté automatiquement au chemin

Exemple

Utilisation (« app.js »)

```
var express = require('express');
var app = express();
var routes = require('./routes/index');
var customers = require('./routes/customers');

app.set("view engine", "ejs");

app.use('/', routes);
app.use('/customers', customers);

app.listen(3000);
```

Routes et contrôleurs

Exemple dans « customers.js » dans dossier « routes »

```
var customerController = require('../controllers/customerController.js');
router.get('/', customerController.get);
router.post('/', customerController.post);
```

On fait appel aux méthodes d'un contrôleur

… Contrôleur (« customerController.js » dans dossier « controllers »)

```
var customers = require("../customers.json");

(function (customerController) {

    customerController.get = function (req, res) {
        res.render('customers', { customers : customers });
    }
    customerController.post = function (req, res) {
        // etc.
    }

})(module.exports);
```

c. Moteurs de vues

Express utilise plusieurs moteurs de vue (ejs, vash, jade, hbs, swig, etc.)

Les vues sont rangées dans un **dossier** « **Views** » par convention, ce qui permet de n'avoir à définir que la vue (sans le chemin). On peut également définir des vues partielles qui seront inclues dans la page.

✓ ejs

Installation

```
npm install ejs --save
```

➔ Afficher une valeur. On utilise **<%= … %>**

```
<h1><%= title %></h1>
```

➔ Bloc de code entre <% … %>

```
<ul>
    <% customers.forEach(function(customer){ %>
      <li><%= customer.name %></li>
    <% }) %>
  </ul>
```

Passage de paramètre à la vue

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

app.set("view engine", "ejs");

app.get('/customer/:id', function (req, res) {
    res.render('customer', { id: req.params.id });
});

app.listen(8080);
```

On change le moteur de vue pour « ejs »

Paramètre passé

Vue trouvée automatiquement dans le répertoire « views », on peut ne pas indiquer l'extension (*.ejs)

✓ Vash

Installation

```
npm install vash --save
```

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

app.set("view engine", "vash");
```

Peu de changements dans le code à part le moteur de vue

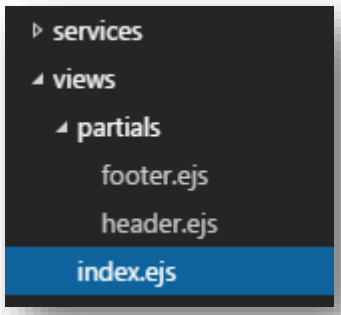
Vue *.vash

Utilisation de @model dans le template

```
<h1>L'identifiant du contact est @model.id !</h1>
```

Documentation

Vues partielles



La page assemble les différentes vues pour obtenir le code complet. On pourrait également avoir une vue partielle pour lister l'ensemble des scripts, une autre pour le « head » par exemple.

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <meta charset="utf-8" />
   <title></title>
        <link href="/css/default.css" rel="stylesheet" />
</head>
```

```
<body>
<%= include partials/header.ejs %>

<section>
        content goes here
</section>

<%= include partials/footer.ejs %>
</body>
</html>
```

Utilisation de deux vues partielles (moteur de vues ejs)

Exemple de vue partielle

Ne contient que la partie à insérer dans la page

```
<footer>
   <div>Copyright &copy; J. ROMAGNY</div>
</footer>
```

d. Variable partagée

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

app.locals.pageTitle="Node Express Demo";
```

(Pas besoin de passer la variable)

Dans la vue (ejs)

```
<h1><%= pageTitle %></h1>
```

e. json

On charge simplement un fichier json

```
app.locals.customers = require("./customers.json");
```

```
[
    {"name":"M. PILLON","email":"mpillo@gmail.com"},
    {"name":"M. BELLIN","email":"mbellin123@htomail.com"}
]
```

Dans une vue (ejs)

```
<ul>
      <% customers.forEach(function(customer){ %>
        <li><%= customers.name %></li>
      <% }) %>
   </ul>
```

7. Bases de données

a. SQL Server

Plusieurs Drivers existent.

… Avec **Tedious**. Installer Tedious (documentation)

```
npm install tedious
```

➔ **Note** : il faut lancer **SQL Browser**

```
var Connection = require('tedious').Connection;
var Request = require('tedious').Request;

exports.getAll = function (req, res) {
   var config = {
     server: 'DonJR',
     userName: 'sa',
     password: 'password123456' ,
     database: 'peopledb',
     options : {
                instanceName: 'SqlExpress',
                rowCollectionOnRequestCompletion: 'true'
            }
  }

  var connection = new Connection(config);
  connection.on('connect', function (err) {
    if (err) {
       console.log("Erreur");
     }
     else {
        var request = new Request("use peopledb; select * from Person", function
(err, rowCount, columns) {
        var result = [];
            columns.forEach(function (column) {
                var person = new Person(column[1].value, column[2].value);
                    result.push(person);
                });

                 res.render('index.ejs', { people: result });
         });
         connection.execSql(request);
      }
    });

};
```

Nom du serveur, indiquer le nom de l'instance (instanceName) dans

« rowCompletionOnRequestCompletion » permet de récupérer le résultat de la reauête en callback

Exécution de la

Autre exemple avec **paramètres**

```
var TYPES = require('tedious').TYPES;

var sql = 'use peopledb; insert into Person(name,twitter) values(@name,@twitter);';
var request = new Request(sql, function (err) {
     console.log("Erreur");
});

request.addParameter('name', TYPES.VarChar, 'M. Bellin');
request.addParameter('twitter', TYPES.VarChar, '@mbellin');

connection.execSql(request);
```

Paramètres

Exécution de la

b. MongoDB

MongoDB est une base NoSQL

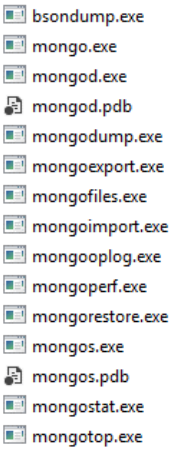
Autres bases NoSQL : RavenDB, Cassandra, Neo4j, Redis, CouchDB

1. Installation de MongoDB

1. Télécharger le zip de MongoDB (ou le msi et choisir le dossier du projet pourl'installation)

name	odified	size	md5	sig	sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-3.0.3-rc1-signed.msi	30 15:46:55	82337280	md5		sha1	sha256
win32/mongodb-win32-x86_64-2008plus-ssl-3.0.3-rc1.zip	-30 15:46:53	80017994	md5	sig	sha1	sha256

2. Créer un dossier « mongodb » dans le projet et y copier l'ensemble de l'archive



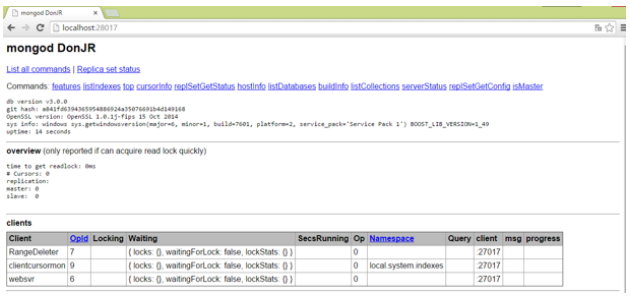
3. Créer un dossier « data » dans « mongodb »
4. Ouvrir une invite de commande, naviguer jusqu'au dossier « mongodb » du projet et indiquer le dossier dans lequel la base est installée

```
mongod –dbpath ./data
```

(Un message « waiting for connections on port 27017 » apparait si tout se passe bien)

Pour avoir accès aux bases depuis « http://localhost:28017 »

```
mongod –dbpath ./data --rest
```



2. Créer une base avec une invite de commandes

Une **base de données NoSQL** a :

- des **collections**
  - contenant des **documents** (**JSON**), **schemaless** (le format peut différer, par exemple un document peut avoir un nom et un autre avoir un nom et un email)

Lancer l'interpréteur de commandes « mongo.exe ». Ouvrir une seconde invite de commande et naviguer jusqu'au dossier « mongodb » du projet

(Note on devra lancer seulement « mongod » , l'interperteur de commandes ne sert que pour insérer des documents à une base de données)

```
mongo
```

Créer ou utiliser une base

```
use peopledb
```

Insertion d'un document dans une collection

```
db.people.insert({name:"J. Romagny", twitter:"@romagny13"})
```

Collection

Document

Un **identifiant unique** est ajouté à chaque document (_id)

```
> db.people.find()
{ "_id" : ObjectId("5542be0d27f284581549abc2"), "name" : "J. ROMAGNY", "twitter"
 : "@romagny13" }
```

Obtenir un document

Utilisation de la méthode « find »

➔ **Retourner tous** les documents de la collection

```
db.people.find()
```

➔ **Filtrer** les résultats

- $gt : plus grand
- $lt : plus petit
- $gte : plus grand ou égal
- $lte: plus petit ou égal
- $or : ou
- $and : et
- $in, $all, $exist, $type ,$regex

```
db.people.find({name : "J. Romagny"})
db.people.find({age : {$gt :18}})
```

Trier

Avec la méthode « sort »

```
db.people.find().sort({name :1})
```

Tri de la collection de personnes sur le nom en ordre croissant

Modification de document

```
db.people.update({name : "J. Romagny"}, {$set :{twitter : "@jerome_romagny13"}})
```

Méthode « update »

Filtre des documents recherchés

Modification de la propriété « twitter »

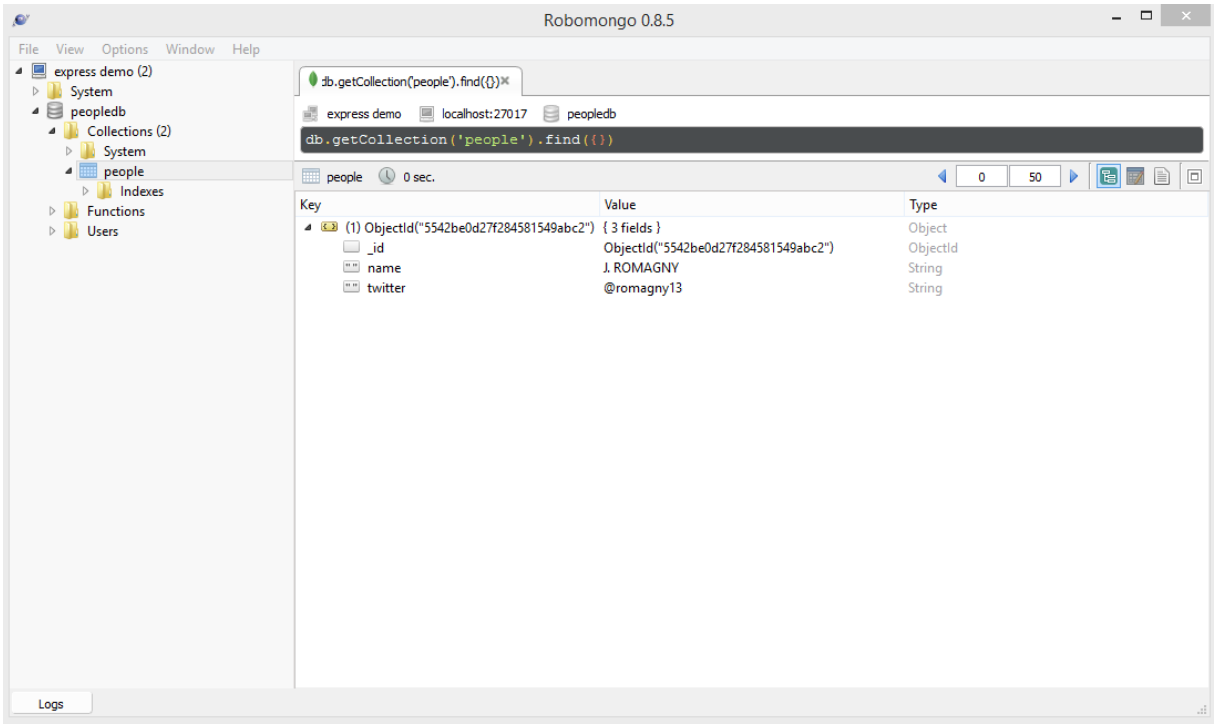
Supprimer un document

```
db.people.remove ({name : "J. Romagny"})
```

$unset permet de supprimer une propriété du document

3. Gérer ses bases avec Robomongo

Robomongo permet de gérer facilement ses bases de données MongoDB



Autres outils d'administration

c. Mongoose

Documentation

1. Installation de Mongoose

```
npm install mongoose
```

2. Schéma/modèle, CRUD et vues

L'exemple décrit correspond un peu à ce que l'on ferait avec Asp.Net Mvc (une application web avec des pages/vues, contrôleurs, modèle). Sauf qu'ici on utilise une base de données NoSQL et NodeJS.

✓ **Le schéma**

Dans un dossier « models », exemple « person.js ».

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var personSchema = new Schema({
    name: String,
    twitter: String
});

module.exports = mongoose.model('Person', personSchema, 'People');
```

Configuration du schéma

Modèle « Person » respectant le schéma. En 3ème paramètre on peut indiquer le nom de la collection « visée »

Shema Types possibles :

- String
- Number
- Date
- Boolean
- Buffer
- ObjectId (Ex : userId: mongoose.Schema.Types.ObjectId)
- Mixed
- Array

Routes

*Un **contrôleur** est utilisé pour exécuter les actions selon les routes demandées.*
*On se **connecte** à la **base de données** « peopledb » avec **Mongoose**.*

*On utilise « **body-parser** » pour pouvoir récupérer les valeurs des **éléments des formulaires** soumis (documentation). Installer body-parser :*

```
npm install body-parser
```

```
var mongoose = require('mongoose');
var express = require('express');
var bodyParser = require('body-parser');
var peopleController = require('./controllers/peopleController.js');

var app = express();
mongoose.connect('mongodb://localhost:27017/peopledb');
app.use(bodyParser.urlencoded({ extended: true }));
app.set("view engine", "ejs");

// ACCUEIL
app.get('/', function (req, res) {
    res.render('home');
});

// LISTE
app.get('/people', function (req, res) {
    peopleController.getAll(req, res);
});

// NEW
app.get('/person/new', function (req, res) {
    peopleController.create(req, res);
});
// soumission de formulaire
app.post('/person/new', function (req, res) {
    peopleController.doCreate(req, res);
});

// UPDATE
app.get('/person/edit/:id', function (req, res) {
    peopleController.edit(req, res);
});
// soumission de formulaire
app.post('/person/edit', function (req, res) {
    peopleController.doEdit(req, res);
});

// DELETE
app.get('/person/delete/:id', function (req, res) {
    peopleController.delete(req, res);
});

app.listen(8080);
```

Connexion à la base de données « peopledb » avec mongoose

Configuration de body-parser

Configuration du moteur de template

Affichage du formulaire pour que l'utilisateur entre les informations

Soumission du formulaire (« post »)

Paramètre d'url

CRUD avec Mongoose

✓ Le contrôleur « peopleController.js »

```
(function (peopleController) {
    var Person = require('../models/person.js');

    exports.getAll = function (req, res) {
        var query = Person.find();
        query.exec(function (err, people) {
            res.render('people.ejs', { people: people });
        });
    };
    exports.create = function (req, res) {
        var newPerson = new Person({ name: '', twitter: '' });
        res.render('person.new.ejs', { person: newPerson });
    };
    exports.doCreate = function (req, res) {
        var entry = new Person({ name: req.body.name, twitter: req.body.twitter });
        entry.save();
        res.redirect(301, '/people');
    };

    exports.edit = function (req, res) {
        var query = Person.findOne({ _id: req.params.id });
        query.exec(function (err, result) {
            res.render('person.edit.ejs', { person: result });
        });
    };
    exports.doEdit = function (req, res) {
        Person.update({ _id: req.body.id }, { $set: { name: req.body.name, twitter:
req.body.twitter } }, function (err) {
            if (!err) {
                res.redirect(301, '/people');
            }
        });
    };
    exports.delete = function (req, res) {
        Person.remove({ _id: req.params.id }, function (err) {
            if (!err) {
                res.redirect(301, '/people');
            }
        });
    };
})(module.exports);
```

Utilisation de la méthode « find » du modèle pour récupérer tous les documents de la collection

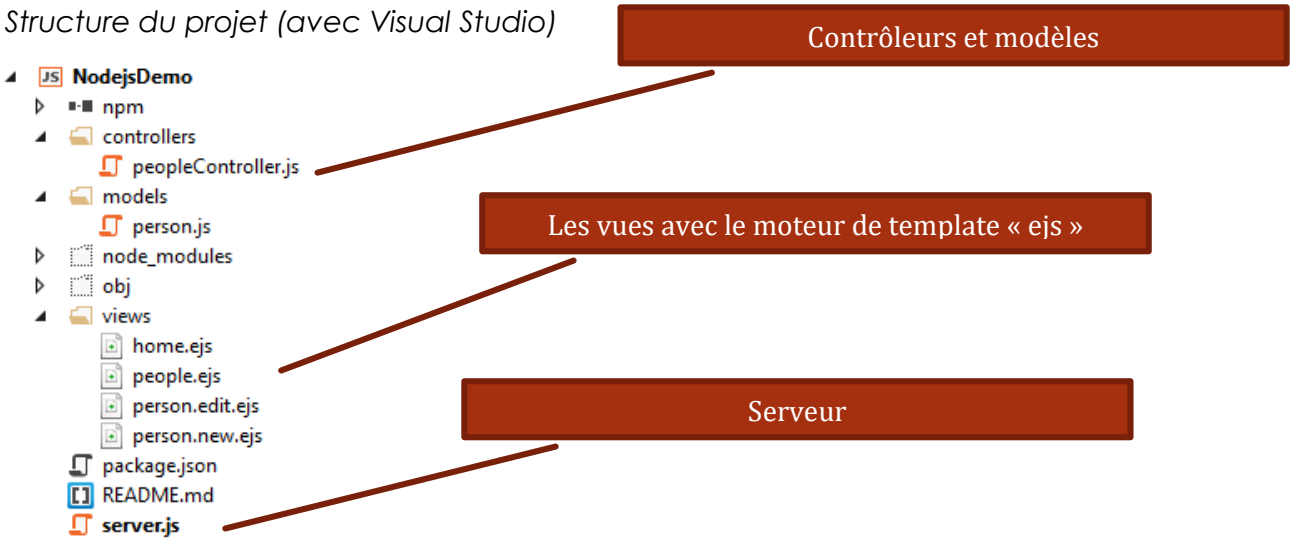
Ajout d'un nouveau document. Récupération des valeurs de formulaire grâce au module « body-parser »

Utilisation de la méthode « update » du modèle pour modifier un élément, le premier paramètre représente la condition de sélection (ici sélection de l'élément par l'id).

$set sert à modifier les éléments

Suppression du document avec la méthode « remove » du modèle auquel on passe une condition de sélection

Structure du projet (avec Visual Studio)



Une requête de sélection complexe

« where » permettant de filtrer, « sort » pour trier », « limit » pour le nombre de résultats, etc.

```
var query = Person.find();

query.sort({ createdOn: 'desc' })
.limit(12)
.exec(function (err, results) {
    res.render('index', { title: 'R&eacute;sultats', notes: results });
});
```

Les 2 façons de créer un nouveau document

- ✓ Avec la méthode « **create** » du modèle

Fonction « callback »

```
Person.create({ name: req.body.name, twitter: req.body.twitter }, function (err) {
    if (!err) {
        res.redirect(301, '/people');
    }
});
```

- ✓ Avec la méthode « **save** »

```
var entry = new Person({ name: req.body.name, twitter: req.body.twitter });
entry.save();
```

Vues

(Avec « **ejs** »)

Page de liste avec liens édition, suppression, et

Boucle sur le tableau passé

```
<body>
<a href="./person/new">Ajouter une personne</a>
<table>
<tr><th>Nom</th><th>Twitter</th></tr>
<% for(var i=0; i<people.length; i++) {%>
   <tr>
          <td><%= people[i].name %></td>
          <td><%= people[i].twitter %></td>
          <td><a href="./person/edit/<%= people[i]._id %>">Editer</a>
           <td><a href="./person/delete/<%= people[i]._id %>">Supprimer</a>
   </tr>
<% } %>
</table>
</body>
```

Formulaire d'ajout

Méthode « post » et URL appelée

```
<body>
    <form action="/person/new" method="post">
              <input type="text" name="name" placeholder="Entrez le nom" />
              <input type="text" name="twitter" placeholder="Entrez le twitter" />
              <input type="submit" value="Valider" />
    </form>
</body>
```

Formulaire d'édition

Un champ caché pour l'id

```
<body>
    <form action="/person/edit" method="post">
           <input type="hidden"  name="id"  value="<%= person._id %>">
              <input type="text" name="name" placeholder="Entrez le nom" value="<%=
person.name %>"/>
              <input type="text" name="twitter" placeholder="Entrez le twitter"
value="<%= person.twitter %>"/>
              <input type="submit" value="Valider" />
    </form>
</body>
```

« Binding » des éléments passés

3. Procédure de Test

Naviguer jusqu'au dossier du projet..puis au dossier contenant mongodb

```
mongod –dbpath ./data
```

Lancer le serveur (Naviguer jusqu'au dossier du projet)

```
node server.js
```

4. Un service WEB avec Node et Mongoose

Ce qui suit serait plutôt l'équivalent d'un service « Asp.Net Web API » que n'importe quelle application « cliente » pourrait consommer (par exemple une application construite avec AngularJS).

```
var mongoose = require('mongoose');
var bodyParser = require('body-parser');
var express = require('express');
var Person = require('./models/person.js');
// config
var app = express();
mongoose.connect('mongodb://localhost:27017/peopledb');
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(function (req, res, next) {
    // url du site ou *
    res.setHeader('Access-Control-Allow-Origin', '*');
    res.setHeader('Access-Control-Allow-Methods', 'GET, POST, OPTIONS, PUT, PATCH,
DELETE');
    res.setHeader('Access-Control-Allow-Headers', 'X-Requested-With,content-type');
    // sécurité
    //res.setHeader('Access-Control-Allow-Credentials', true);
    next();
});

// GET ALL
app.get('/api/people', function (req, res) {
    Person.find().exec(function (err, results) {
        res.set("Content-Type", "application/json");
        res.send(200, results);
    });
});

// GET ONE
app.get('/api/person/:id', function (req, res) {
    Person.findOne({ _id: req.params.id }).exec(function (err, result) {
        res.set("Content-Type", "application/json");
        res.send(200, result);
    });
});
// ADD
app.post('/api/person/new', function (r
    console.log(req.body);
    var newPerson = new Person({ name: req.body.name, twitter: req.body.twitter });
    newPerson.save();

    res.set("Content-Type", "application/json");
    res.send(201, newPerson);
});


// UPDATE
app.put('/api/person/edit', function (req, res) {
    console.log(req.body);
    Person.update({ _id: req.body._id }, { $set: { name: req.body.name, twitter:
req.body.twitter } }, function (err) {
        res.set("Content-Type", "application/json");
        res.send(200, req.body);
    });
});
// DELETE
```

Même modèle que dans l'exemple précédent

On configure bodyParser afin de pouvoir parser du **json** (content-type :application/json) et des éléments de **formulaires** reçus (content-type : application/x-www-form-urlencoded)

Configuration des accès. Nécessaire sous Chrome

On reçoit un paramètre id dans l'URL

http://localhost:8080/person/5501db57fac223298d5d0ab4

On reçoit un élément à ajouter au format **JSON**

{ "name":"J. Romagny","twitter":"@romagny13" }

On reçoit un élément à modifier au format **JSON avec en plus l'id**

{ "_id":"5504cfd6d3aa2cc8162e8c56","name":"J. Romagny","twitter":"@romagny13"}

```
app.delete('/api/person/delete/:id', function (req, res) {
    console.log(req.params.id);
    Person.remove({ _id: req.params.id }, function (err) {
        res.set("Content-Type", "application/json");
        res.send(204);
    });
});

app.listen(8080);
```

*Exemple de **service** avec **AngularJS** (avec **$http**), application « cliente » consommant le service WEB.*

```
(function () {
    'use strict';

    angular.module('app').service('peopleService', peopleService);

    function peopleService($http) {
        var baseUrl = "http://localhost:8080";
        var getPeople = function () {
            return $http({
                method: 'GET',
                url: baseUrl + "/api/people",
                params: { 'update-time': new Date().getTime() }
            });
        }
        var getPerson = function (id) {
            return $http.get(baseUrl + "/api/person/" + id);
        }
        var addPerson = function (person) {
            return $http.post(baseUrl + "/api/person/new", person);
        }
        var updatePerson = function (person) {
            return $http.put(baseUrl + "/api/person/edit", person);
        }
        var deletePerson = function (id) {
            return $http.delete(baseUrl + "/api/person/delete/" + id);
        }
        return {
            getPeople: getPeople,
            getPerson: getPerson,
            addPerson: addPerson,
            updatePerson: updatePerson,
            deletePerson: deletePerson
        }
    }
})();
```

Documentation $http

8. Sécurité

a. « Simple »

Installation avec npm de :

- express
- mongoose (MongoDB, schéma et validation)
- ejs (moteur de vue)
- body-parser (pour récupérer les valeurs de formulaires soumis)
- client-sessions (variables de session)

« app.js »

```
var express = require('express'),
    app = express(),
    bodyParser = require('body-parser'),
    session = require('client-sessions'),
    User = require('./Models/user'),
    hasher = require('./auth/hasher.js');

var db = mongoose.connect('mongodb://localhost/simpleauth', function (err) {
    if (err) {
        console.error(err);
    }
});
app.set("view engine", "ejs");
app.use(session({
    cookieName: 'session',
    secret: 'secret'
}));
app.use(express.static(__dirname + "/public"));
app.use(bodyParser.urlencoded({ extended: true }));

/* ROUTES */
app.get('/', function (req, res) {
    if (req.session && req.session.user) {
        res.render('index', { 'user' : req.session.user });
    }
    else {
        res.render('index');
    }
});

app.get('/register', function (req, res) {
    res.render('register');
});
app.post('/register', function (req, res) {

    if (req.body.email == "" || req.body.password == "" || req.body.confirmpassword == "") {
        res.render('register', { email: req.body.email, message: "Vous devez renseigner tous les
champs." });
    }
    else if (req.body.password != req.body.confirmpassword) {
        res.render('register', { email: req.body.email, message: "Les mots de passe ne correspondent
pas." });
    }
    else {
```

Connexion à la base MongoDB

Configuration Moteur de vue, session, …

Les routes pourraient être dans des fichiers séparés et le code dans des

Affiche la vue « index » à laquelle on passe l'uilisateur s'il est connecté

Affiche la vue pour se

Vérification des données du

```
        var salt = hasher.createSalt();
        var passwordHash = hasher.computeHash(req.body.password, salt);
        var newUser = new User({
            email : req.body.email,
            password : passwordHash,
            salt: salt
        });
        newUser.save(function (err) {
            if (err) {
                var message = getErrorMessage(err);
                res.render('register', { 'email': req.body.email, 'message': message });
            }
            else {
                loginUser(req,res,newUser);
            }
        });
    }
});
app.get('/login', function (req, res) {
    res.render('login');
});
app.post('/login', function (req, res) {
    if (req.body.email == "" || req.body.password == "") {
        res.render('register', { email: req.body.email, message: "Vous devez renseigner tous les
champs." });
    }
    else {
        User.findOne({ 'email' : req.body.email }, function (err, user) {
            if (err) {
                var message = getErrorMessage(err);
                res.render('login', { message: message });
            }
            else if (!user) {
                res.render('login', { email: req.body.email, message: 'Pas d\'Utilisateur enregistré
avec cet email.' });
            }
            else {
                var salt = user.salt;
                var testHash = hasher.computeHash(req.body.password, salt);
                if (user.password == testHash) {
                    loginUser(req,res,user);
                }
                else {
                    res.render('login', { email: req.body.email, mes
identifiants invalides.' });
                }
            }
        });
    }
});
app.get('/logoff', function (req, res) {
    req.session.destroy();
    res.redirect('/');
});
var loginUser = function (user){
    user.password = undefined;
    user.salt = undefined;
    req.session.user = user;
    var redirect_to = req.session.redirect_to ? req.session.redirect_to : '/';
    delete req.session.redirect_to;
    res.redirect(redirect_to);
}
```

Inscription de l'utilisateur (ajout à la base MongoDB)

Si pas d'erreur durant l'inscription, on enregistre l'utilisateur dans une variable de session et redirection

Vérification des données du

On recherche l'utilisateur par l'email dans la base

Si un utilisateur a été trouvé pour cet email on vérifie que l'email saisi correspond bien

Déconnexion (destruction de la session et redirection)

Suppression des données sensibles avant ajout de l'utilisateur à une variable de session

```
var getErrorMessage = function (err) {
    var message = "";
    if (err.errors && err.errors.email && err.errors.email.message) {
        message += err.errors.email.message + '\n';
    }
    if (err.errors && err.errors.password && err.errors.password.message) {
        message += err.errors.password.message + '\n';
    }
    if (err.message == '') {
        message = err.message;
    }
    return message;
}

var port = process.env.PORT || 3000;
app.listen(port);
console.log("Listen on " + port + " ...");
```

Modèle

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

var userSchema = new Schema({
    email: { type: String, trim: true, required: 'Le champ email est requis', match: [/.+\@.+\..+/,
'L\'adresse email est invalide.'], unique: true },
    password: { type : String, required : 'Le champ mot de passe est requis.' },
    salt: String,
    created: { type: Date, default: Date.now }
});

userSchema.path('email').validate(function (value, next) {
    var User = mongoose.model('User');
    User.findOne({ 'email' : value }, function (err, user) {
        if (err) {
            next(false);
        }
        next(!user);
    });
}, 'Cet email est déja enregistré.');


module.exports = mongoose.model('User', userSchema, 'users');
```

L'email est requis, doit être valide et pas déjà utilisé dans la base MongoDB

Hasher

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

(function (hasher) {

    hasher.createSalt = function () {
        return crypto.randomBytes(128).toString('base64');
    };

    hasher.computeHash = function (password, salt) {
        var hmac = crypto.createHmac("sha1", salt);
        return hmac.update(password).digest("hex");
    };

})(module.exports);
```

Module inclus dans node (pas besoin de l'installer)

Création d'une « clé » (salt) et retour du mot de passe

Vues

Formulaire d'enregistrement de « register.ejs »

```
<h2>Nouveau? Inscrivez-vous!</h2>
<form action="/register" method="post">
   <p><input type="email" name="email" placeholder="Email" value="<% if(locals.email){ %> <%= email  %>
<% } %>"/></p>
   <p><input type="password" name="password" placeholder="Mot de passe" /></p>
   <p><input type="password" name="confirmpassword" placeholder="Confirmer le mot de passe" /></p>
   <span class="text-error"><% if(locals.message){ %> <%= message %> <% } %></span>
   <p><input type="submit" value="S'inscrire" /></p>
</form>
```

Formulaire de connexion de « login.ejs »

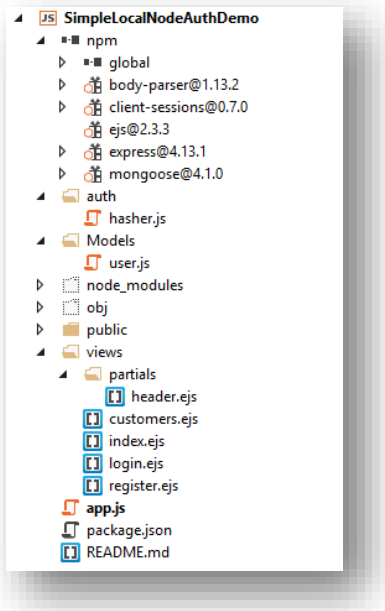
```
<h2>Bienvenue! Connectez-vous!</h2>
<form action="/login" method="post">
    <p><input type="email" name="email" placeholder="Email" value="<% if(locals.email){ %> <%= email
%> <% } %>"/></p>
     <p><input type="password" name="password" placeholder="Mot de passe" /></p>
     <span class="text-error"><% if(locals.message){ %> <%= message %> <% } %></span>
     <p><input type="submit" value="Se connecter" /></p>
</form>
```

Vue partielle « header.ejs » affichant un menu selon que l'utilisateur est connecté ou non

```
<ul class="loginMenu">
        <% if(locals.user){ %>
        <li><span>Bonjour <%= user.email %> </span></li>
        <li><a href="/logoff">Se déconnecter</a></li>

        <% } else { %>
        <li><a href="/register">S'inscrire</a></li>
        <li><a href="/login">Se connecter</a></li>
        <% } %>
</ul>
```

Utilisateur connecté,

Organisation du projet



Autorisation

Pour accèder à la page « clients », il faut que l'utilisateur soit authentifié

```
app.get('/customers', ensureAuthenticated, function (req, res) {
    var customers = require("./customers.json");
    res.render('customers', { 'user' : req.session.user, customers : customers });
});
function ensureAuthenticated(req, res, next) {

    req.session.redirect_to = req.url; // redirect url

    if (req.session && req.session.user) {
        return next();
    }
    else {
        res.redirect('/login');
    }
}
```

b. Avec « Passport »

« Passport » est un midddleware d'authentification pour node.js (Documentation)

Installation

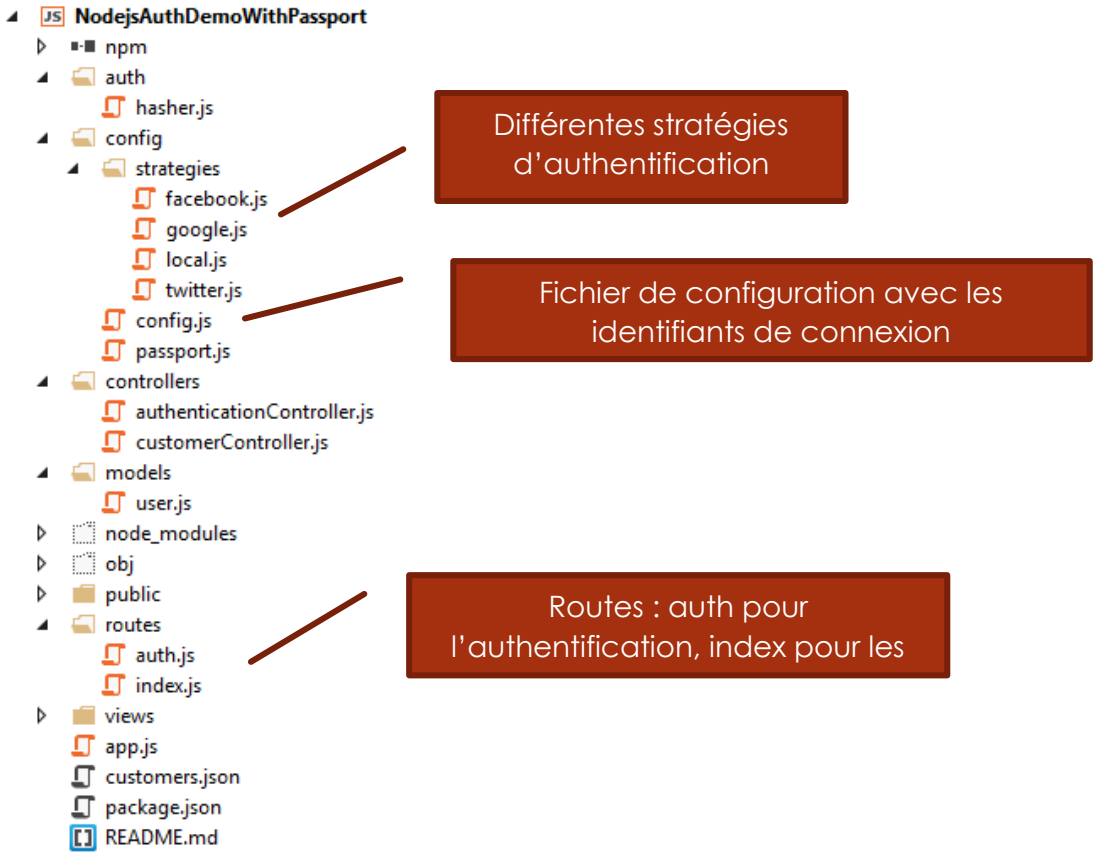
```
npm install passport --save
```

Puis

```
npm install passport-local --save
```

Et **passport-facebook**, **passport-twitter** et **passport-google-auth** selon les besoins…
(voir les stratégies disponibles)

Authentification (« local », « facebook », « twitter », « google », etc.)



« app »

```
var express = require('express'),
    app = express(),
    bodyParser = require('body-parser'),
    session = require('client-sessions'),
    routes = require('./routes/index'),
    auth = require('./routes/auth'),
    mongoose = require('mongoose'),
    config = require('./config/config');

var db = mongoose.connect(config.db, function (err) {
    if (err) {
        console.error(err);
    }
});
app.set("view engine", "ejs");
app.use(session({ cookieName: 'session', secret: 'cat reset' }));
app.use(express.static(__dirname + "/public"));
app.use(bodyParser.urlencoded({ extended: true }));

var passportConfig = require('./config/passport')(app);

app.use('/', routes);
app.use('/auth', auth);

var port = process.env.PORT || 3000;
app.listen(port);
console.log("Listen on " + port + " ...");
```

« Config »

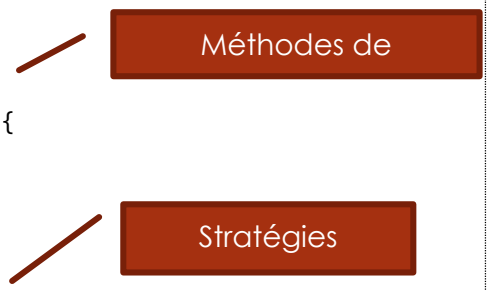
```
module.exports = {
    db: 'mongodb://localhost/peopledb',
    facebook: {
        clientID: '<your clientID>',
        clientSecret: '<your client secret>',
        callbackURL: '/auth/facebook/callback'
    },
    twitter: {
        clientID: '<your clientID>',
        clientSecret: '<your client secret>',
        callbackURL: '/auth/twitter/callback'
    },
    google: {
        clientID: '<your clientID>',
        clientSecret: '<your client secret>',
        callbackURL: 'http://localhost:3000/auth/google/callback'
    }
};
```

Le modèle (« mongoose »

```
var mongoose = require('mongoose'),
    Schema = mongoose.Schema;

var userSchema = new Schema({
    firstName: { type: String, trim: true, default: '' },
    lastName: { type: String, trim: true, default: '' },
    displayName: { type: String, trim: true },
    username: { type: String , trim: true },
    email: { type: String, trim: true, default: '', required: 'Le champ email est
requis.', match: [/.+\@.+\..+/, 'L\'adresse email est invalide.'], unique: true },
    photo : { type: String },
    created: { type: Date, default: Date.now },
    // local
    password: { type: String },
    salt: { type: String },
    // OAuth
    facebook: { type : Object },
    twitter: { type : Object },
    google: { type : Object }
});

module.exports = mongoose.model('User', userSchema, 'users');
```

« passport »

```
var passport = require('passport');

module.exports = function (app) {
    app.use(passport.initialize());
    app.use(passport.session());
    passport.serializeUser(function (user, next) {
        next(null, user);
    });
    passport.deserializeUser(function (user, next) {
        next(null, user);
    });
    require('./strategies/local-strategy')();
    require('./strategies/facebook-strategy')();
    require('./strategies/twitter-strategy')();
    require('./strategies/google-strategy')();
}
```



Routes (« auth.js »)

```
var express = require('express'),
    router = express.Router(),
    passport = require('passport'),
    authenticationController = require('../controllers/authenticationController');

// local
router.route('/register')
      .get(authenticationController.showRegister)
      .post(authenticationController.register);
router.route('/login')
      .get(authenticationController.showLogin)
      .post(authenticationController.login);

// facebook
router.route('/facebook').get(passport.authenticate('facebook', { scope: ['email'] }));
router.route('/facebook/callback').get(authenticationController.oauthCallback('facebook'));

// twitter
router.route('/twitter').get(passport.authenticate('twitter'));
router.route('/twitter/callback').get(authenticationController.oauthCallback('twitter'));

// google
router.route('/google').get(passport.authenticate('google'));
router.route('/google/callback').get(authenticationController.oauthCallback('google'));

// log off
router.get('/logoff', function (req, res) {
    req.logout();
    req.session.destroy();
    res.redirect('/');
});

module.exports = router;
```

« authenticationController »

```
var passport = require('passport'),
    hasher = require('../auth/hasher'),
    User = require('../models/user');

(function (authenticationController) {

    authenticationController.showRegister = function (req, res) {
        res.render('register', { email: "", message: "" });
    }

    authenticationController.register = function (req, res) {

        var body = req.body;
        if (body.email == "" || body.password == "" || body.confirmpassword == "") {
            res.render('register', { email: req.body.email, message: "Vous devez renseigner tous les
champs." });
        }
        else if (body.password != body.confirmpassword) {
            res.render('register', { email: req.body.email, message: "Les mots de passe ne correspondent
pas." });
        }
        else {
            User.findOne({ 'email': req.body.email }, function (err, findUser) {
                if (findUser) {
                    res.render('register', { email: req.body.email, message: "Un utilisateur avec cet
email existe déjà." });
                }
                else {
                    var salt = hasher.createSalt();
                    var passwordHash = hasher.computeHash(req.body.password, salt);
                    var newUser = new User({
                        username : req.body.email,
                        email : req.body.email,
                        password : passwordHash,
                        salt: salt
                    });
                    newUser.save(function (err) {
                        if (err) {
                            res.render('register', { email: req.body.email, message: "Un erreur est
survenue. Impossible d'enregistrer cet utilisateur." });
                        }
                        else {
                            loginUser(req, res, newUser);
                        }
                    });
                }
            });
        }
    }

    // login
    authenticationController.showLogin = function (req, res) {
        res.render('login', { message: "" });
    }

    authenticationController.login = function (req, res) {
        var body = req.body;
        if (body.email == "" || body.password == "") {
            res.render('login', { email: req.body.email, message: "Vous devez renseigner tous les
champs." });
```

On vérifie que les champs sont renseignés correctement

Ajout d'un nouvel utilisateur

```
        }
        else {
            passport.authenticate('local', function (err, user) {
                if (err) {
                    res.render('login', { email: req.body.email, message: "Utilisateur introuvable ou
identifiants invalides." });
                }
                else {
                    loginUser(req, res, user);
                }

            })(req, res);
        }
    }
    // OAuth ...
    authenticationController.oauthCallback = function (strategy) {
        return function (req, res, next) {
            passport.authenticate(strategy, function (err, user) {
                if (err) {
                    var message = getErrorMessage(err);
                    res.render('login', { message: message });
                }
                else {
                    loginUser(req, res, user);
                }
            })(req, res, next);
        };
    };

    var getErrorMessage = function (err) {
        var message = "";
        if (err.errors && err.errors.email && err.errors.email.message) {
            message = err.errors.email.message + '\n';
        }
        else if (err.message) {
            message = err.message;
        }
        return message;
    }

    var loginUser = function (req, res, user) {

        user.password = undefined;
        user.salt = undefined;

        req.logIn(user, function (err) {
            if (err) {
                res.render('login', { message: err.message });
            } else {

                var redirect_to = req.session.redirect_to ? req.session.redirect_to : '/';
                delete req.session.redirect_to;
                res.redirect(redirect_to);
            }
        });
    }

})(module.exports);
```

Suppression des données sensibles afin de passer

« Local »

« local-strategy »

```
var passport = require('passport'),
    LocalStrategy = require('passport-local').Strategy,
    hasher = require('../../auth/hasher'),
    mongoose = require('mongoose'),
    User = mongoose.model('User');

module.exports = function () {

    passport.use(new LocalStrategy({
        usernameField: 'email',
        passwordField: 'password'
    },
    function (email, password, next) {
        // find
        User.findOne({ email: email }, function (err, user) {

            if (err || !user) {
                return next({ message : "Utilisateur introuvable ou identifiants invalides." });
            }
            else {

                if (user.salt == undefined) {
                    return next({ message: "Utilisateur introuvable ou identifiants invalides." });
                }

                var testHash = hasher.computeHash(password, user.salt);
                if (testHash === user.password) {
                    return next(null, user);
                }
                else {
                    return next({ message: " Utilisateur introuvable ou identifiants invalides." });
                }
            }
        });
    }));
}
```

On cherche si un utilisateur est inscrit avec cet email et on compare les mots de

Facebook

Créer une application avec Facebook Developpers et la rendre « publique » (onglet « Status & Review »)

« facebook-strategy »

```
var passport = require('passport'),
    FacebookStrategy = require('passport-facebook').Strategy,
    mongoose = require('mongoose'),
    User = mongoose.model('User'),
    config = require('../config');

module.exports = function () {

    passport.use(new FacebookStrategy({
        clientID: config.facebook.clientID,
        clientSecret: config.facebook.clientSecret,
        callbackURL: config.facebook.callbackURL,
        profileFields: ['id', 'displayName', 'link', 'photos', 'email']
    },
    function (accessToken, refreshToken, profile, next) {

        if (profile.emails == undefined) {
            return next({ message : "Erreur, aucun email associé à ce compte facebook." });
        }

        User.findOne({ 'facebook.id': profile.id }, function (err, user) {

            if (user) {
                next(null, user);
            }
            else {
                User.findOne({ 'email' : profile.emails[0].value }, function (err, findUser) {
                    if (findUser) {
                        return next({ message : "Un utilisateur avec cet email est déjà enregistré."
});
                    }
                });

                var newUser = new User();
                newUser.firstName = profile.name.givenName;
                newUser.lastName = profile.name.familyName;
                newUser.displayName = profile.displayName;
                newUser.email = profile.emails[0].value;
                newUser.username = profile.username;
                newUser.photo = profile.photos[0].value;
                newUser.facebook = {};
                newUser.facebook.id = profile.id;
                newUser.facebook.token = accessToken;

                newUser.save(function (err) {
                    if (err) {
                        next(err);
                    }
                    else {
                        next(null, newUser);
                    }
                });
            }
        });
    }));
}
```

Identifiants de l'application

Couplé au scope défini, permet de récupérer des

Gère le cas où aucun email n'est renvoyé

On cherche dans la base MongoDB si un utilisateur avec cet identifiant Facebook est déjà enregistré, et le renvoie si c'est le cas.

Twitter

Créer une application avec Twitter Developpers et récupérer les identifiants.

« twitter-strategy »

```
var passport = require('passport'),
    TwitterStrategy = require('passport-twitter').Strategy,
    mongoose = require('mongoose'),
    User = mongoose.model('User'),
    config = require('../config');

module.exports = function () {

    passport.use(new TwitterStrategy({
        consumerKey: config.twitter.clientID,
        consumerSecret: config.twitter.clientSecret,
        callbackURL: config.twitter.callbackURL
    },
    function (accessToken, tokenSecret, profile, next) {

        User.findOne({ 'twitter.id': profile.id }, function (err, user) {

            if (user) {
                next(null, user);
            }
            else {
                // add
                var newUser = new User();
                newUser.displayName = profile.displayName;
                newUser.username = profile.username;
                newUser.photo = profile.photos[0].value;
                newUser.twitter = {};
                newUser.twitter.id = profile.id;
                newUser.twitter.token = accessToken;

                newUser.save(function (err) {
                    if (err) {
                        next(err);
                    }
                    else {
                        next(null, newUser);
                    }
                });
            }
        });
    }));
}
```

Twitter est un cas particulier. En effet on ne récupère pas d'email. Il faudra peut-être créer une vue demandant à l'utilisateur d'entrer un

Google

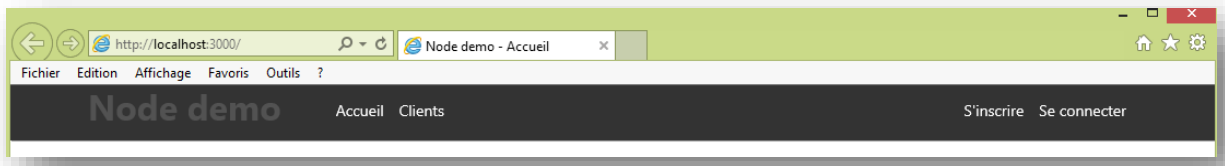
Créer un projet avec la console Google Developers. Créer un « identifiant client » (onglet « identifiants », url de redirection « http://localhost:3000/auth/google/callback » par exemple) … et activer Google+ API (onglet « API »)

« google-strategy »

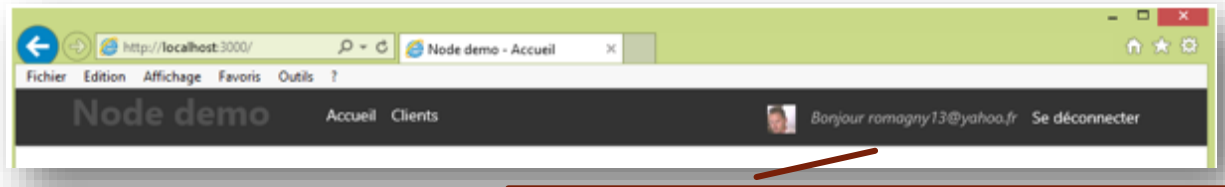
```
var passport = require('passport'),
    GoogleStrategy = require('passport-google-auth').Strategy,
    mongoose = require('mongoose'),
    User = mongoose.model('User'),
    config = require('../config');

module.exports = function () {

    passport.use(new GoogleStrategy({
        clientId: config.google.clientID,
        clientSecret: config.google.clientSecret,
        callbackURL: config.google.callbackURL
    },
    function (accessToken, refreshToken, profile, next) {

        User.findOne({ 'google.id': profile.id }, function (err, user) {

            if (user) {
                next(null, user);
            }
            else {
                User.findOne({ 'email' : profile.emails[0].value }, function (err, findUser) {
                    if (findUser) {
                        return next({ message : "Un utilisateur avec cet email est déjà enregistré."
});
                    }
                });

                var newUser = new User();
                newUser.firstName = profile.name.givenName;
                newUser.lastName = profile.name.familyName;
                newUser.displayName = profile.displayName;
                newUser.email = profile.emails[0].value;
                newUser.username = profile.username;
                newUser.photo = profile.image.url;
                newUser.google = {};
                newUser.google.id = profile.id;
                newUser.google.token = accessToken;

                newUser.save(function (err) {
                    if (err) {
                        next(err);
                    }
                    else {
                        next(null, newUser);
                    }
                });
            }
        });
    }));
}
```

Message personnalisé pour l'utilisateur connecté

Menu



Menu « connecté »



Message personnalisé avec la photo du profil

```
<header>
  <div class="headerContent">
    <h1>Node demo</h1>
    <nav>
      <ul>
        <li><a href="/">Accueil</a></li>
        <li><a href="/customers">Clients</a></li>
      </ul>
    </nav>
    <ul class="loginMenu">

      <% if(locals.user){ %>
        <% if(locals.user.photo){ %>
          <li><img src="<%= user.photo  %>"></img> </li>
        <% } %>
        <li><span>Bonjour <%= user.email %> </span></li>
        <li><a href="/auth/logoff">Se déconnecter</a></li>

      <% } else { %>

        <li><a href="/auth/register">S'inscrire</a></li>
        <li><a href="/auth/login">Se connecter</a></li>

      <% } %>

    </ul>
  </div>
</header>
```

Si l'utilisateur est défini

Autorisation

Routes (« index.js »)

```
var express = require('express'),
    router = express.Router(),
    customerController = require('../controllers/customerController.js');

// index
router.get('/', function (req, res) {
    if (req.user) {
        res.render('index', { 'user' : req.user });
    } else {
        res.render('index');
    }
});

// customers
router.get('/customers', ensureAuthenticated, customerController.getAll);

function ensureAuthenticated(req, res, next) {
    req.session.redirect_to = req.url; // redirect url

    if (req.isAuthenticated()) {
        return next();
    }
    else {
        res.redirect('/auth/login')
    }
}

module.exports = router;
```

On passe l'utilisateur connecté à la

Pour accéder à la page des clients, l'utilisateur doit être authentifié, si ce n'est pas le cas on le redirige vers la page de

1

2

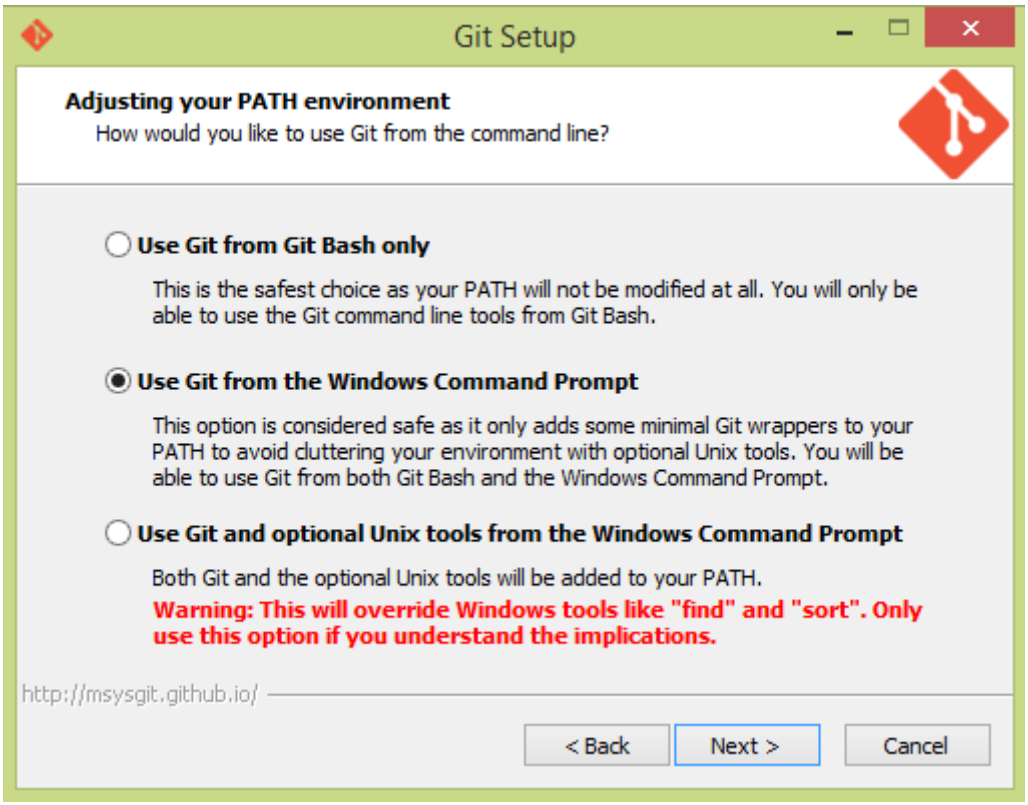
9. **MEAN**.JS (**M**ongoDB, **E**xpress, **A**ngularJS, **N**ode.js)

MEAN.JS

Documentation

Installation

- Node.js
- MongoDB
- Git avec l'option « Use Git from the Windows Command Prompt »



Puis installation en global de …

- **Bower**

```
npm install -g bower
```

- **Grunt**

```
npm install -g grunt -cli
```

- **Yeoman** (Documentation)

```
npm install -g yo
```

- **Generator meanjs** (Generators)

```
npm install -g generator-meanjs
```

Création d'un **projet** (naviguer jusqu'au dossier où créer le projet avec la commande « cd … ») puis

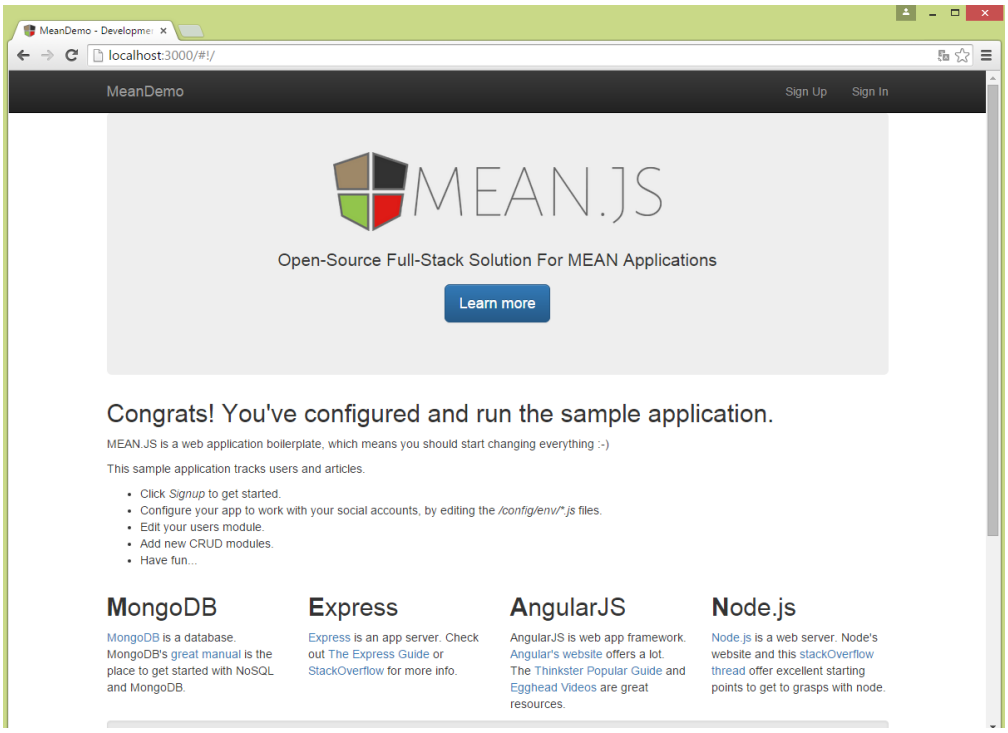
```
yo meanjs
```

Plusieurs questions sont alors posées :

- Nom de l'application donner le nom désiré (dans l'exemple « MeanDemo »)
- Laisser les options par défaut et se contenter de valider avec entrée jusqu'au nom de la compagnie / auteur : donner le nom de l'auteur désiré
- On peut ensuite choisir de générer un exemple avec le code (répondre « Y »)
- Choix des modules AngularJS à inclure
- … enfin valider, le code est généré

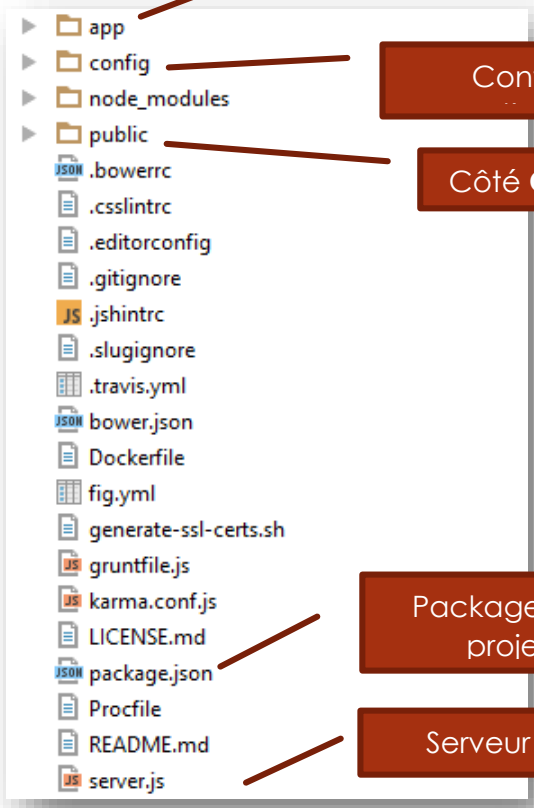
```
You're using the official MEAN.JS generator.
? What would you like to call your application? MeanDemo
? How would you describe your application? (Full-Stack JavaScript with MongoDB,
? How would you describe your application? Full-Stack JavaScript with MongoDB, E
xpress, AngularJS, and Node.js
? How would you describe your application in comma seperated key words? (MongoDB
? How would you describe your application in comma seperated key words? MongoDB,
 Express, AngularJS, Node.js
? What is your company/author name? J. ROMAGNY
? Would you like to generate the article example CRUD module? Yes
? Which AngularJS modules would you like to include? (Press <space> to select)
>(*) ngCookies
 (*) ngAnimate
 (*) ngTouch
 (*) ngSanitize
```

Si tout se passe bien le code est généré dans le dossier. On peut ensuite lancer MongoDB, et le serveur (avec « grunt » ou plus simplement « node server.js ») et accéder au site.



Le projet généré

« **app** » : Côté **Serveur** (MongoDB, Express,



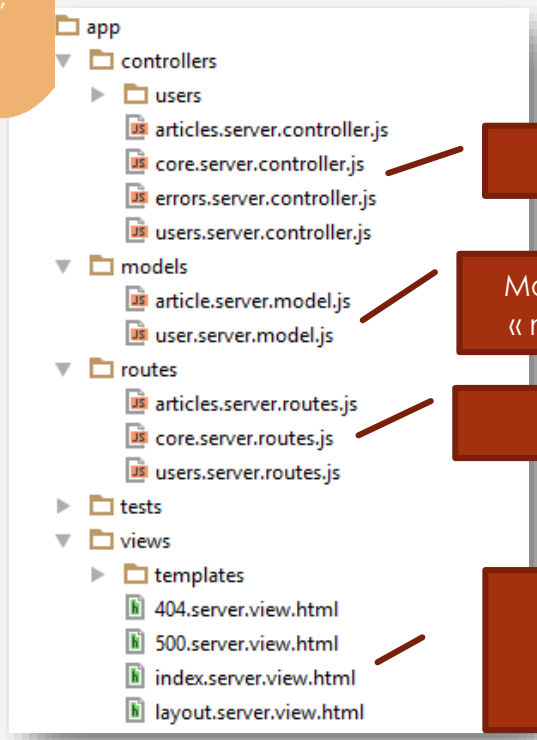
Configuration de

Côté **Client** (AngularJS)

Package avec informations du projet et dépendances

Serveur

Côté serveur (MongoDB, Express,



Contrôleurs

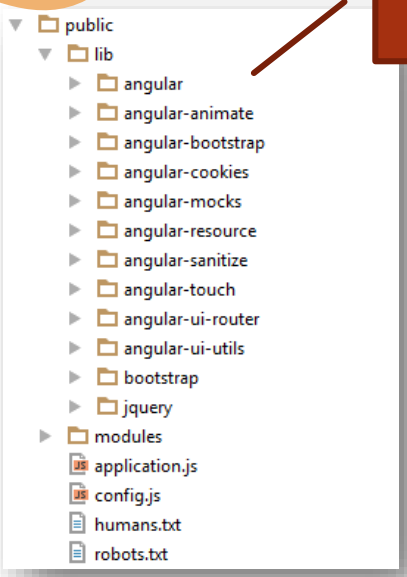
Modèles avec « mongoose »

Routes

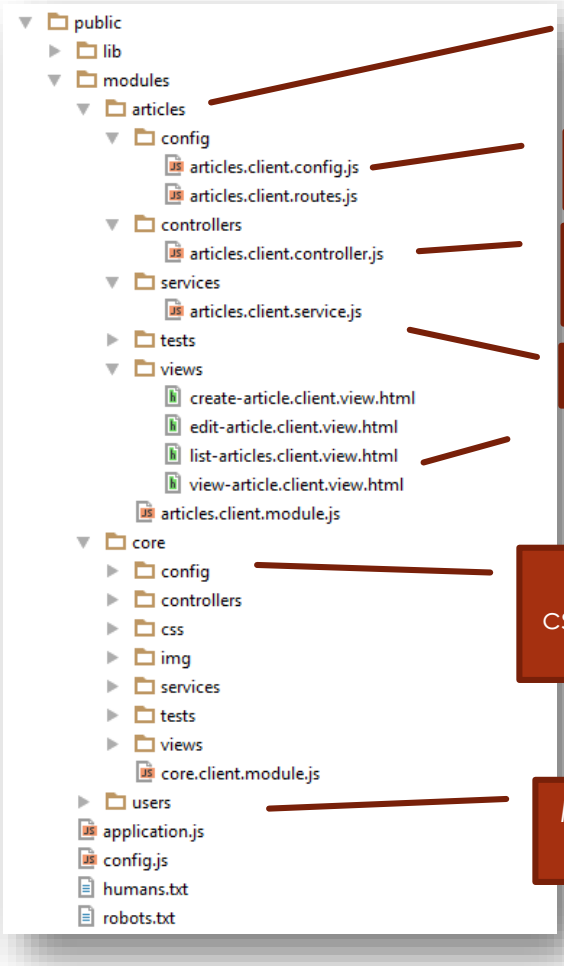
Vues (index : vue par défaut, layout : « master page »)

Moteur de template utilisé : Swig

Côté client (AngularJS)



« lib » contient les fichiers pour AngularJS, jQuery et



Module AngularJS « articles » exemple généré

« config » : routes et menus

Contrôleurs AngularJS avec injection ($scope, etc.)

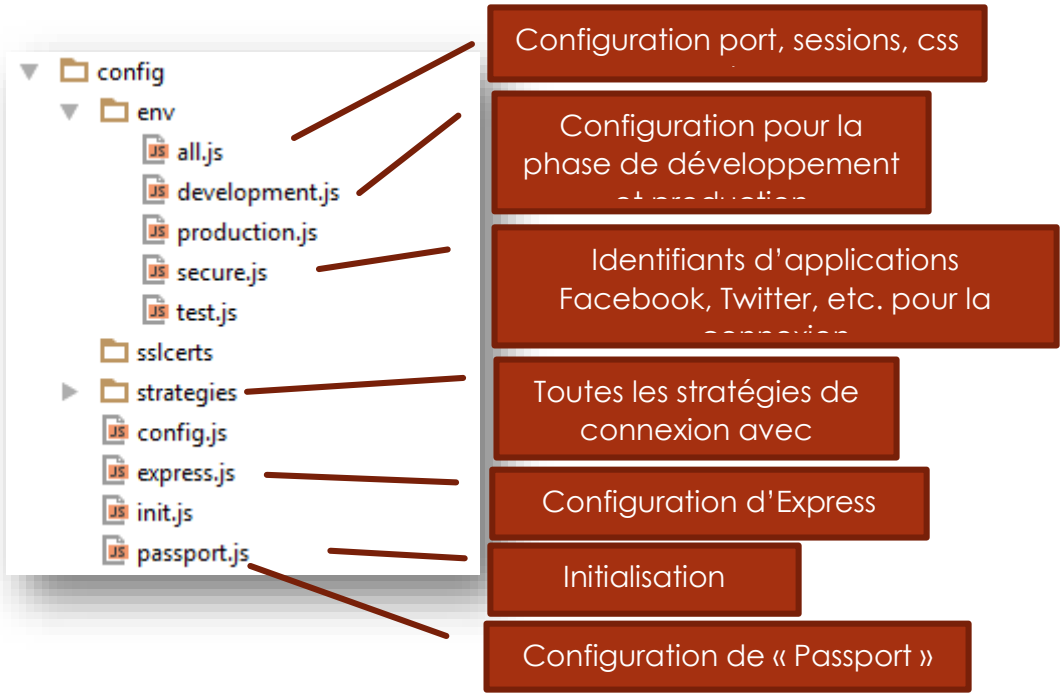
« factories »

Vues (partielles) avec contrôleurs AngularJS et binding (attributs data…)

Module « core » , css, images et fichiers partagés

Module « users » pour l'authentification

Configuration de l'application



Génération de code

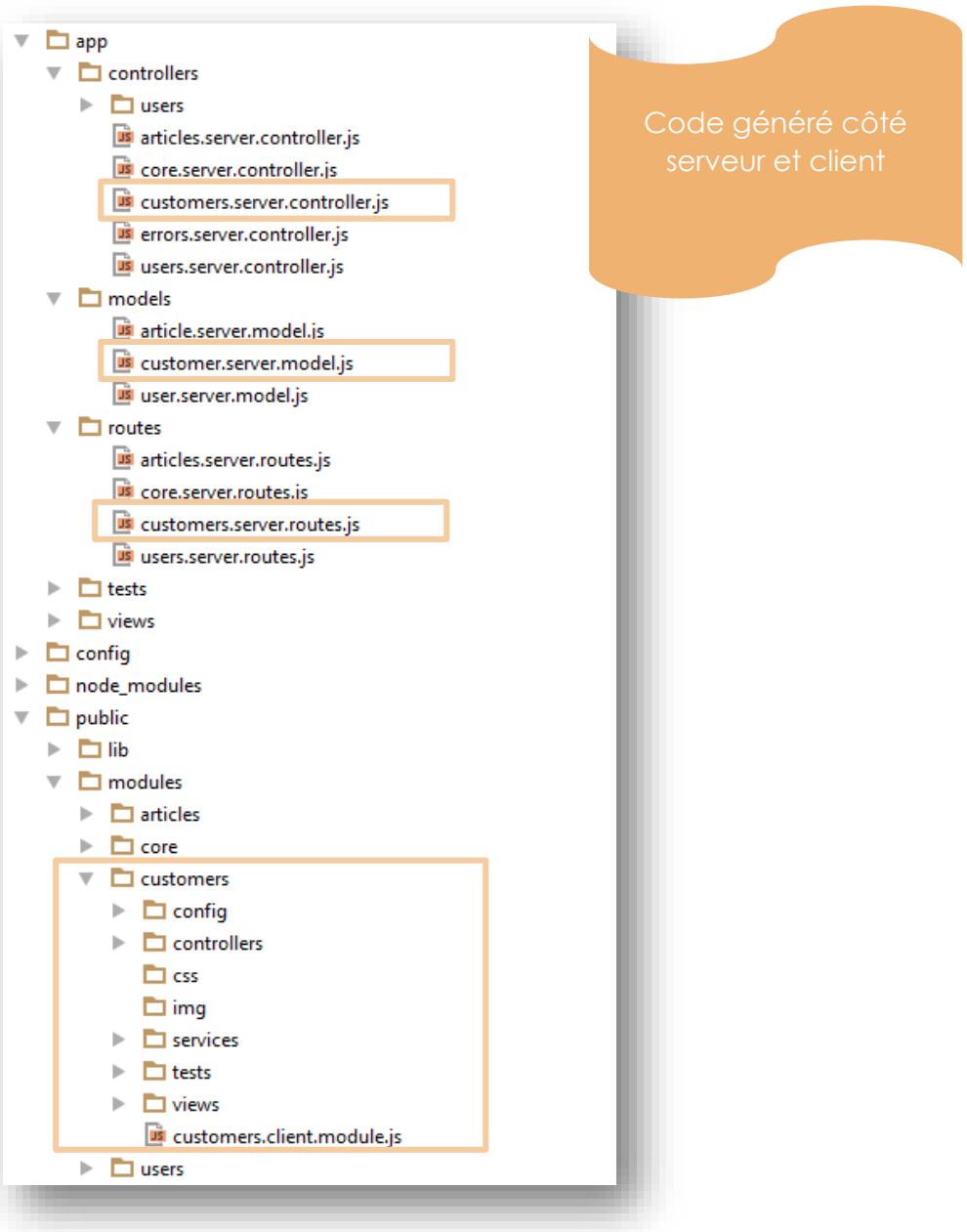
Documentation

« crud-Module »

Depuis une invite de commande, naviguer jusqu'au dossier du projet MEAN.JS (« cd… ») puis …

```
yo meanjs:crud-module <module-name>
```

Exemple création d'un module nommé « customers ». Après avoir répondu aux questions, génération du code



« Express »

Contrôleur

```
yo meanjs:express-controller <controller-name>
```

Modèle

```
yo meanjs:express-model <model-name>
```

Route

```
yo meanjs:express-route <route-name>
```

« Angular »

Module

```
yo meanjs:angular-module <module-name>
```

Contrôleur

```
yo meanjs:angular-controller <controller-name>
```

Route

```
yo meanjs:angular-route <route-name>
```

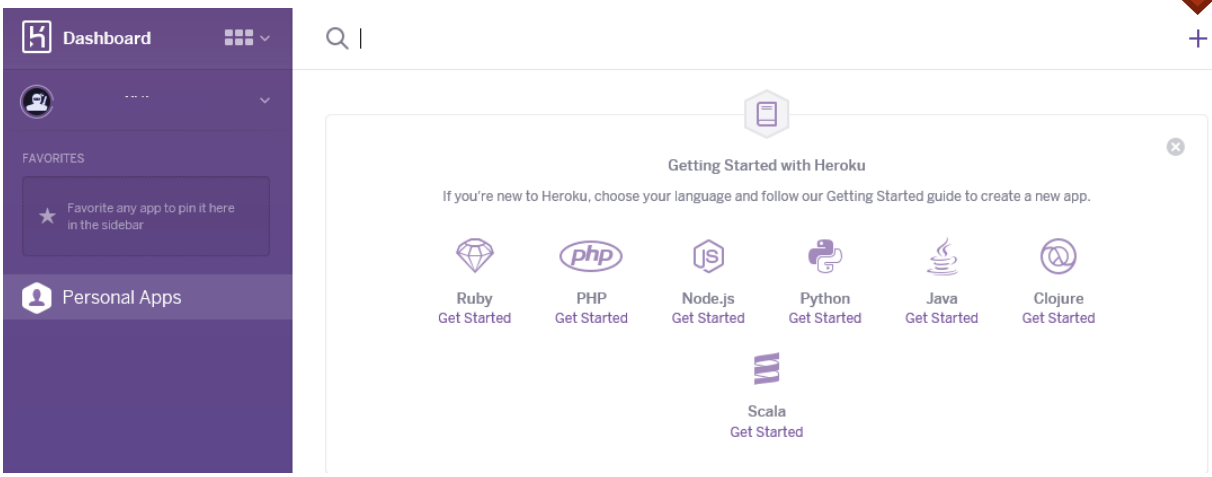
Vue

```
yo meanjs:angular-view <view-name>
```

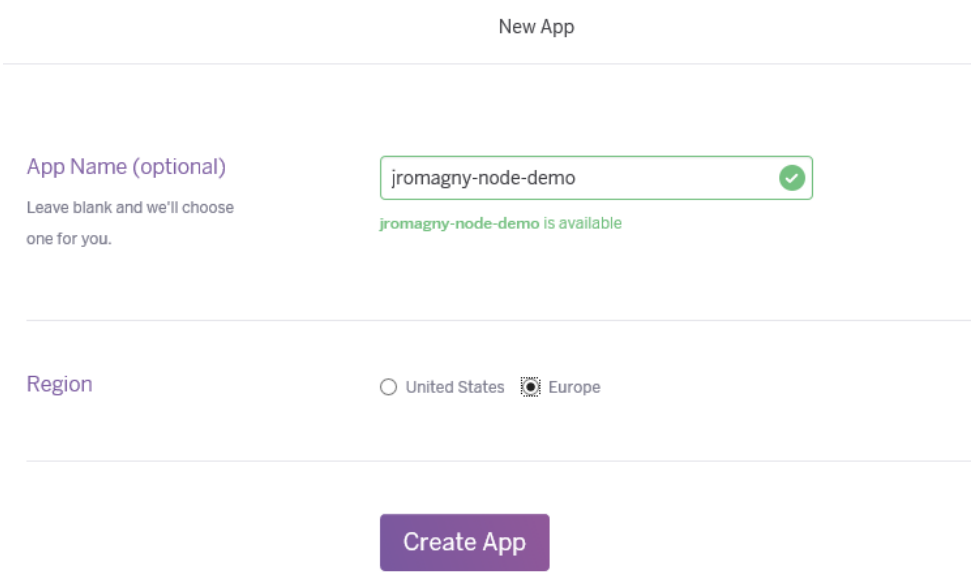
10. Publication

➔ Heroku

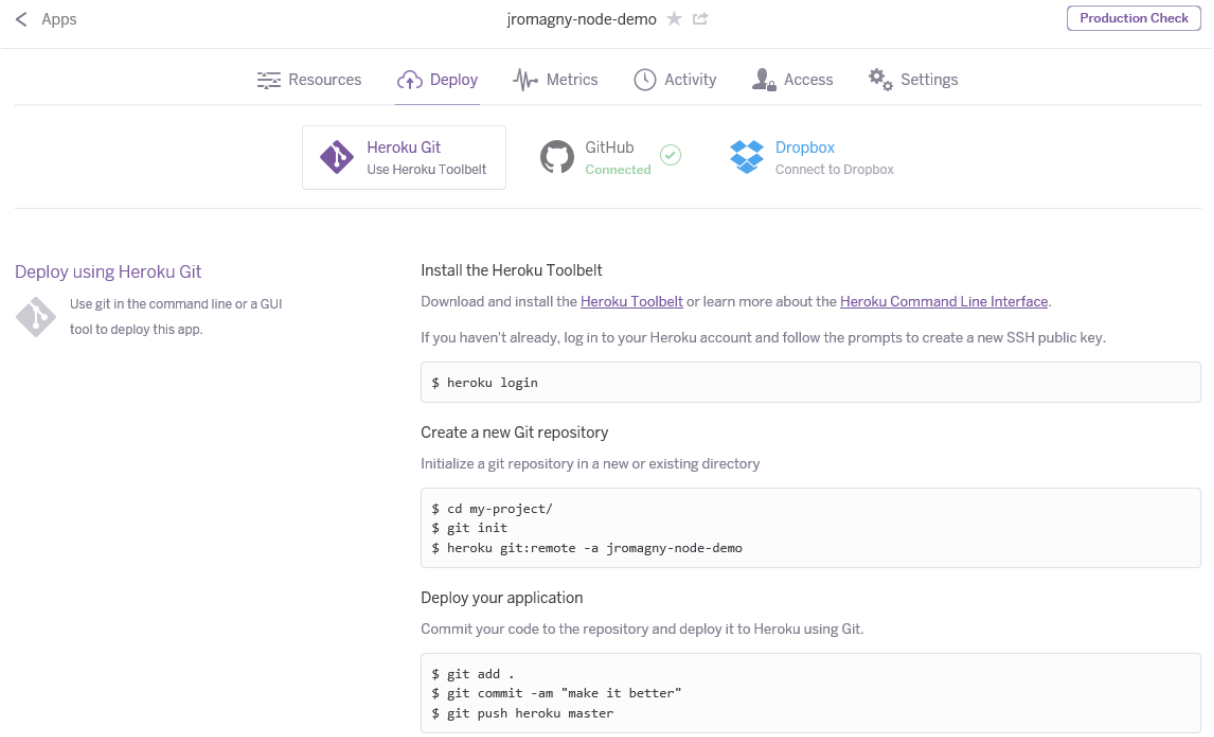
1. Créer un compte
2. Créer une nouvelle application



3. Donner un nom d'application



4. Plusieurs méthodes de déploiement disponibles



Préparation à la publication du projet

1. « package.json »

```
{
    "name": "NodejsAuthDemo",
    "version": "0.0.0",
    "description": "NodejsAuthDemo",
    "main": "server.js",
    "author": {
        "name": "romagny13",
        "email": ""
    },
    "engines": {
        "node": "0.12.x",
        "npm": "2.7.x"
    },
    "dependencies": {
        "body-parser": "^1.12.3",
        "ejs": "^2.3.1",
        "express": "^4.12.3",
        "mongoose": "^4.0.2",
        "passport": "^0.2.1",
        "passport-local": "^1.0.0"
    }
}
```

Ajouter une section « engines » avec les versions de node et npm (que l'on peut obtenir avec les commandes « node --version » « npm --version » depuis une invite de commande). Le « x » en fin « arrondit »

2. « procfile.js »

Créer un fichier « procfile.js » à la racine du projet et y ajouter le nom du serveur

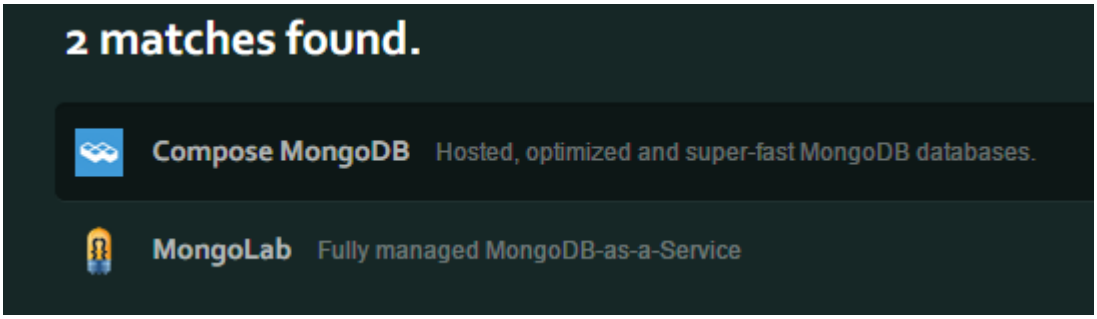
```
web: node app.js
```

3. Dans le serveur « server.js » ou « app.js » selon le nom donné, modifier le port

```
var port = process.env.PORT || 3000;
app.listen(port);
console.log("Listen on " + port + "...");
```

Port de test

4. MongoDB addons (option « free » mais demande quand même de rentrer des informations de paiement☹)



Permet de créer une base. On obtient également les informations de connexion (Créer un utilisateur minimum) et modifier la chaine de connexion dans le projet.

```
mongoose.connect('mongodb://.....');
```