Formation Industrialisation Web & Javascript

Cahier de TP

Pré-requis

NodeJS et NPM

Windows

Si une connexion internet est disponible, se rendre sur http://nodejs.org et télécharger la dernière version de NodeJS. NPM est embarqué. Après l'installation, lancer une invite de commande et exécuter `node --version` pour vérifier que la commande `node` est disponible. Faire de même pour `npm`. En cas de problème, vérifier que le dossier d'installation de NodeJS se trouve bien dans le `PATH`, et l'ajouter si nécessaire.

Sans connexion internet, utiliser la version fournie par le formateur dans le dossier `Node`. Décompresser l'archive `node-windows.zip` et ajouter le dossier résultant au `PATH`. Pour plus de faciliter, renommer `node-64.exe` (ou `node-32.exe` sur un système 32 bits) en `node.exe`.

Linux

Si une connexion internet est disponible, se rendre sur http://nodejs.org et télécharger la dernière version de NodeJS. NPM est embarqué. Décompresser, l'archive et ajouter le dossier `bin` au `PATH`.

Sur un système Debian il est possible d'utiliser APT, mais les versions sont un peu anciennes. Les paquets à installer sont `node` et `npm`. La commande dans pour Node est dans ce cas `nodejs` au lieu de `node`.

Sans connexion internet, utiliser la version fourni par le formateur dans le dossier `Node`. Décompresser l'archive `node-v0.10.22-linux-x64.zip` et ajouter le dossier `bin` au `PATH`.

Serveur

Pour tester l'application, il nous faut un serveur. Un serveur NodeJS est fourni dans le répertoire `Server/nodejs`. Pour le lancer, exécuter `node server.js`. Il sert par défaut le dossier `TP0` mais il est possible de modifier le répertoire servi dans `server.json`.

Grunt

Exécuter la commande `npm install -g grunt-cli` pour installer la commande `grunt`. Pour chaque TP, installer Grunt localement pour le projet en se placer dans le répertoire de travail et exécuter `npm install grunt`.

TP 1 : Génération d'un livrable

Nous allons mettre en place un système de build automatisé pour l'application existante ZenContacts. Ce build se compose des étapes suivantes :

- Analyse syntaxique du Gruntfile
- Analyse syntaxique du javascript
- Concaténation de l'ensemble des fichiers javascript en seulement 2 fichiers : un pour les librairies tierce, et un pour l'application elle-même
- Minification des 2 fichiers précédemment créés

Analyse syntaxique Javascript

- Installer `grunt-contrib-jshint`.
- Créer un Gruntfile et configurer la tâche `jshint`.
  - La tâche `jshint` accepte des cibles sous forme de tableaux de string, chaque string étant un *filesystem glob* tel que `js/**/*.js`. Alternativement, la cible peut être un objet contenant un attribut `src` qui contient un chemin ou un tableaux de chemins.
  - Créer une cible `gruntfile` pour Gruntfile et une cible `app` pour les fichiers de l'applications qui se trouvent dans le dossier `js/app`.
- Exécuter `grunt jshint:gruntfile`, et corriger les erreurs s'il y en a.
- Exécuter `grunt jshint:app`, et remarquer les erreurs qui s'affichent.
  - Si le rapport d'erreurs semble trop confu, installer `jshint-stylish` avec `npm` puis insérer `options: { reporter: require('jshint-stylish') }` dans la tâche `jshint` et relancer.
- Il y a beaucoup d'erreurs `'angular' is not defined` ou `'window' is not defined`. Ces erreurs sont des faux positifs puisque ces variables globales sont bien définies. Nous devons indiqué cela à JSHint.
  - Créer un fichier `.jshintrc` dans le même dossier que le Gruntfile.
  - Activer l'option `browser` pour éviter `'window' is not defined`.
  - Pour les variables globales de librairies, JSHint supporte une option `globals` qu'il faut valoriser par une map dont les clés sont les noms des variables et les valeurs sont des booléens indiquant si la variable peut être redéfinie ou non.
  - Ajouter `"globals": { "angular": false }` dans `.jshintrc`.
  - Avant de relancer la tâche, modifier votre Gruntfile pour qu'il utilise le `.jshintrc` en ajoutant l'option `jshint: true` à la cible `app`.
  - Relancer la tâche et vérifier que `window` et `angular` ne posent plus de soucis.
  - Ajouter les globales `markdown`, `Fuse`, et `zenContactApp` (attention cette dernière étant définie par l'application, il faut mettre `true` en valeur).
  - `'$' is not defined` se corrige en activant l'option `jquery`.
- Relancer la tâche. Il ne devrait rester que 2 types d'erreurs.
  - Corriger les `Missing semicolon` en éditant les fichiers incriminés.
  - L'erreur `Use the function form of "use strict"` est intéressante. Pour le moment chaque fichier de l'application commence par `'use strict';`, ce qui active le mode strict. Activer le mode strict en tête de script est une mauvaise pratique puisque cela rend le script impossible à concaténer avec un script non-strict. La bonne pratique veut que chaque script soit englobé par une fonction auto-appelante (`(function(){})()`), qui elle est en mode strict. C'est cela que JSHint nous conseille de faire. Cependant, nous n'allons pas concaténer notre javascript applicatif avec du javascript tierce-partie, et tous les scripts de l'application sont stricts. Il n'y a donc pas de problème : nous pouvons ignorer cett erreur en activant l'option `globalstrict`.
- Relancer la tâche et vérifier qu'aucune erreur ne subsiste.

Concaténation

- Installer `grunt-contrib-concat`.
- Configurer la tâche `concat`.
  - Une cible de `concat` est un object contenant un tableaux de *filesystem glob* source `src` et un chemin de destination `dest`.
  - Attention la concaténation est effectué dans l'ordre du tableau donné à `src`. Il est important de respecter l'ordre de chargement des fichiers. Par exemple il est impératif que `angular.js` apparaissent avant `angular-animate.js`. Se référer à `index.html` pour trouver l'ordre précis.
  - Ajouter une cible `app` qui concatène les fichiers javascript applicatifs (dossier `js/app/`) et qui enregistre le résultat dans un dossier `target`.
  - Ajouter une cible `thirdparty` qui concatène tous les fichiers javascript des librairies (tous les dossiers se trouvant dans `js/` sauf `app/`).
- Exécuter `grunt concat` et vérifier le résultat.

Minification

- Installer `grunt-contrib-uglify`.
- Configurer la tâche `uglify`.
  - La configuration a la même structure que `concat`. Créer une cible `app` et une cible `thridparty`.
- Exécuter la tâche et vérifier le résultat.
- Editer `index.html` pour qu'il référence les fichiers minifiés.

- Lancer l'application et vérifier son fonctionnement.

Bonus

- Ajouter une tâche composite par défaut qui exécute les 3 tâches.
- Utiliser `grunt-contrib-clean` pour supprimer le dossier `target`. La tâche `clean` n'a pas besoin de cibles, il est possible de la définir simplement par un tableau de chemins à supprimer.
- Vérifier le fonctionnement du build complet avec `grunt clean default`.

TP2 : Gestion des dépendances

Dépendances de développement avec NPM

Ecrire un fichier `package.json` qui liste les dépendances du build créé pendant le TP1, afin que tout puisse être installé avec un simple `npm install`.

Dépendances côté client avec Bower

- Installer Bower à l'aide de NPM.
- Utiliser Bower pour télécharger toutes les dépendances de l'application ainsi qu'enregistrer la liste des dépendances dans un `bower.json` : jQuery, Bootstrap, Markdown, Fuse, Angular, Angular Resource, Angular Route, Angular Cookies, et Angular UI Utils.
- Pour tester le `bower.json`, supprimer `bower_components` puis exécuter `bower install`. Vérifier que toutes les dépendances sont présentes.
- Adapter la tâche `concat` du Gruntfile pour qu'elle utilise les fichiers téléchargés par Bower.

TP3 : Tests

Comme nous avons déjà rédigé des tests lors de la formation AngularJS, nous allons ici nous contenter d'intégrer l'exécution de ceux-ci à Grunt.

- Mettre à jour la configuration Karma (fichier `test/karma.conf.js`) pour qu'elle charge les fichiers tierce-partie depuis Bower.
- Installer `grunt-karma`.
- Installer les plugins suivants : `karma-jasmine`, `karma-chrome-launcher`, `karma-firefox-launcher`.
- Configurer la tâche `karma`.
  - Les cibles de la tâche `karma` sont des objets contenant au moins l'attribut `configFile` égal à un chemin vers un fichier de configuration Karma. Afin que Karma ne bloque pas le build, il faut ajouter l'attribut `singleRun` et lle valoriser à `true`.
- Exécuter la tâche.

L'ouverture des navigateurs par Karma peut devenir gênante sur un poste de développement. Pour l'éviter, le plus simple est d'utiliser le navigateur headless PhantomJS.

- Installer PhantomJS à l'aide de NPM.
- Installer le plugin PhantomJS pour Karma (son nom est `karma-phantomjs-launcher`).
- Modifier la configuration Karma.
- Relancer `grunt karma` pour vérifier le fonctionnement.
- Ajouter la tâche `karma` à la tâche par défaut.

TP4 : Rechargement à chaud

Pour ce TP nous allons mettre en place LiveReload.

- Installer l'extension LiveReload pour le navigateur de travail.
- Installer `grunt-contrib-watch`.
- Configurer la tâche `watch`.
  - Activer l'option `livereload` pour toutes les cibles.
  - Créer une cible `js` qui surveille les fichiers javascript et qui lance l'analyse syntaxique et les tests quand cela arrive.

    - Créer une cible `tests` qui surveille les tests qui les relance.
    - Créer une cible qui surveille les fichiers HTML (`index.html` et le contenu du dossier `view`). Il n'y a aucune tâche à lancer.
- Lancer la tâche puis faire une modification quelconque afin de tester le rechargement.

Bonus

- Ajouter une cible qui surveille le Gruntfile lui-même.