ECMA SCRIPT 6

MODULES DEVELOPPEMENT & PRODUCTION WORKFLOW

Nicolas Fazio
@FazioNico

PRÉSENTATION DE LA FORMATION

Jour 1

- Utilisation des Classes ES6

Jour 2

- Présentation de la syntaxe et utilisation des modules ES6
- Comment organiser et hiérarchiser son application et de son code

Jour 3

- Découverte et utilisation des outils de développement.

Jour 4-5

- Mise en production d'une application Javascript ES6

JOUR 1

LES CLASSES

UTILISATION DES CLASSES

Les classes JavaScript ont été introduites avec ECMAScript 6. Elles sont un « sucre syntaxique » par rapport à l'héritage prototypal. En effet, cette syntaxe n'introduit pas un nouveau modèle d'héritage dans JavaScript ! Elle fournit uniquement une syntaxe plus simple et plus claire pour créer des objets et manipuler l'héritage.

resources: https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes

CLASS DEFINITION

```
class MaClass {

        constructor(){
                this.title = 'Hello MaClass'
                this.maMethode()
        }

        maMethode(){
                console.log(`<h1>${this.title}</h1>`)
        }

}
let maClass = new MaClass()
```

resources: https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes

TP

step1 + step2 + step3

https://github.com/FazioNico/cours-es6

JOUR 2

LES MODULES

PRÉSENTATION DU SYSTEME DE MODULES ES6

Les modules permettent d'encapsuler du code JS dans un package ou namespace tout en permettant si besoin, un accès extérieur à certaines propriétés ou fonctions.

Un module ES6 est un fichier contenant du code JavaScript. Il n’y a pas de mot-clé spécifique module et un module est lu comme un script.

resources: https://hacks.mozilla.org/2015/08/es6-in-depth-modules/

SYNTAXE DES MODULES ES6

import et **export**

L'utilisation des modules se fait avec les fonctions import et export.

resources: https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/export
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Instructions/import

EXPORT

Définition d'un module ES6 exportable

```
// date.lib.js

export const today = New Date()

export function shortDate() {
        return  today.getDate() + '/' +
                today.getMonth() + 1  + '/' +
                today.getFullYear()
}
```

IMPORT

Utilisation d'un module ES6 exportable

```
import { today, shortDate } from './date.lib.js';

        console.log(today)          // Mon Sep 12 2016 09:11:23
        console.log(shortDate())    // 12/09/2016
```

JOUR 2 (SUITE)

STRUCTURE D'UNE APPLICATION

COMMENT ORGANISER ET HIÉRARCHISER SON APPLICATION ET DE SON CODE

- Création d'un environnement de développement
- Séparation des fichiers d'application et de templates
- Organisation des modules
- Création d'un répertoire de production

POURQUOI?

Vous avez certainement déjà codé en JS et votre application est généralement créer et déclarée dans un seul fichier.

Cela ne pose pas vraiment de problème pour une petite application, mais comment allez vous faire pour maintenir une application comme votre travail de projet, dont le code peut contenir plus de 2000 lignes ? Comment retrouver facilement et rapidement les classes, methodes, events, fonctions etc...

C'est là qu'intervient la séparation des fichiers !

ENVIRONNEMENT DE DÉVELOPPEMENT

Zone de développement de l'application. C'est uniquement dans cette endroit que vous aller coder.

On organise cette zone en différentes sections:

app (core)
src (ressources)
www (fichiers statiques)

DOSSIER APP

Contient tous les fichiers JS de l'application.

```
dev
├── app
    ├── app.js
    ├── pages
    │   ├── my-module
    │       ├── class.js
    │       ├── class.ui.js
    ├── components
    │   ├── my-module
    │       ├── class.js
    │       ├── class.ui.js
    ├── providers
        ├── my-module
            ├── class.js
```

DOSSIER SRC

Contient toutes les ressources de l'application.

```
dev
  ├── src
      ├── bower_components
      |   ├── jquery
      |   |     ├── jquery.min.js
      ├── css
      |   ├── style.css
      ├── img
          ├── img1.png
          ├── img2.png
```

DOSSIER WWW

Contient tous les fichiers statiques de l'application.

```
dev
  ├── www
      ├── index.html
```

RÉPERTOIRE DEV COMPLET

```
dev
  ├── app
  |   ├── app.js
  |   ├── pages
  |   |   ├── my-module
  |   |       ├── class.js
  |   |       ├── class.ui.js
  |   ├── components
  |   |   ├── my-module
  |   |       ├── class.js
  |   |       ├── class.ui.js
  |   ├── providers
  |   |   ├── my-module
  |   |       ├── class.js
  ├── src
  |   ├── bower_components
  |   |   ├── jquery
  |   |         ├── jquery.min.js
  |   ├── css
  |   ├── img
  |       ├── img1.png
  |       ├── img2.png
  ├── www
      ├── index.html
```

ZOOM SUR LE FICHIER DEV/APP.JS

```
// dev/app.js

import { HomePage } from './pages/home/home.js';

class MyApp {

  constructor(){
    this.appBody = document.getElementsByTagName("app")[0];
  }

  start(){
    // init HomePage
    let homePage = new HomePage(this.appBody);
  }

}
let app = New MyApp()
app.start();
```

ZOOM SUR LE FICHIER DEV/APP/PAGES/HOME/HOME.JS

```
// dev/app/pages/home/home.js

 export class HomePage {

   constructor(appBody){
     this.appBody = appBody
     this.pageTitle = 'Hello world!';
     this.initUI();
   }

   initUI(){
     // remove all section before display UI
     if(document.getElementsByTagName("section")[0]){
       document.getElementsByTagName("section")[0]
       .parentNode.removeChild(document.getElementsByTagName("section")[0])
     }
     // create page skeleton
     let pageSkeleton = `
       < section>
         <h1>${this.pageTitle}</h1>
       < /section>`;
     // add page skeleton in body
     this.appBody.insertAdjacentHTML( 'afterbegin', pageSkeleton )
```

```
    }
}
```

SÉPARER HTML ET JS

Il est possible de séparer le HTML de votre code JS.
Cela permet une meilleures organisation et facilite le travail.

SÉPARER HTML ET JS
DEV/APP/PAGES/HOME/HOME.JS

```
// dev/app/pages/home/home.js
import {homeSkeleton} from "./home-ui";
export class HomePage {

  constructor(appBody){
        this.appBody = appBody
        this.pageTitle = 'Hello world!';
        this.initUI();
  }
  initUI(){
        // remove all section before display UI
        if(document.getElementsByTagName("section")[0]){
          document.getElementsByTagName("section")[0]
          .parentNode.removeChild(document.getElementsByTagName("section")[0])
        }
        // create page skeleton
        let pageSkeleton = this.getPageSkeleton();
        // add page skeleton in body
        this.appBody.insertAdjacentHTML( 'afterbegin', pageSkeleton )
  }

  getPageSkeleton(){
        // return page skeleton
```

```
        let data = {}; // create obj to pass data
        data.pageTitle = this.pageTitle // asigne data
        return  homeSkeleton(data);
  }
}
```

SÉPARER HTML ET JS

DEV/APP/PAGES/HOME/HOME-UI.JS

```
// dev/app/pages/home/home-ui.js

export function homeSkeleton(data){
        // return page skeleton
        return  `
                < section class="container">
                        <h1>${data.pageTitle}</h1>
                < /section>
        `;
}
```

TP

step4
séparer les classes du projet en plusieurs fichiers et les organiser correctement
https://github.com/FazioNico/cours-es6

TESTER L'APPLICATION SUR CHROME...

SUR FIREFOX...

ET SUR SAFARI...

MAIS POURQUOI CELA NE FONCTIONNE PAS?

RAPPEL

Les fonctionnalités d'Es6 ne sont pas encore complètement implémentée dans les navigateurs actuel
:-(

Il vas donc devoir trouver une solution pour débugger l'application...

Solution 1: oublier la séparation des fichier
Solution 2: Transformer tout notre code Es6 en Es5

NOTRE CHOIX

SOLUTION 2:
TRANSFORMER TOUT NOTRE CODE ES6 EN ES5

Mais comment faire? On vas pas tout réécrire non?
En plus Es5 c'est dépassé!
On utilise Es6 dans cette formation!!!!

NOTRE CHOIX

SOLUTION 2:
TRANSFORMER TOUT NOTRE CODE ES6 EN ES5

Pas de panique!!!

On vas coder en Es6 toute notre application!
Mais on vas délivrer du code transformé en Es5 pour pouvoir utiliser toutes les fonctionnalités d'Es6.
Et on vas rien réécrire!!
On vas utiliser un super outil qui le fait pour nous.

:-)

JOUR 3

INTRODUCTION À NODE JS

PRÉSENTATION DE NODE JS

Node.js est un environnement d'assez bas niveau permettant d'exécuter du javascript non plus dans le navigateur web mais sur le serveur.

En effet, Node.js permet, au même titre que PHP ou Java, d'écrire des scripts qui vont vous permettre d'interagir avec les ressources de votre machine.

https://nodejs.org

PRÉSENTATION DE NODE JS

Nous allons utiliser Node.js uniquement pour concevoir un environnent de développement efficace et ordonné.

Nous n'irons pas plus loin dans l'utilisation de Node.js car ce n'est pas le sujet de cette formation.

PRÉSENTATION DE NODE JS

Node.js va nous permettre d'installer des outils de développement qui vont grandement simplifier notre travail.

Ces outils se présentent sous forme de **module node.js** et nous allons devoir télécharger les modules dont nous aurons besoin pour concevoir notre environnement de développement.

INSTALLER NODE JS

MAC OSX

installer Node Version Manager (nvm)

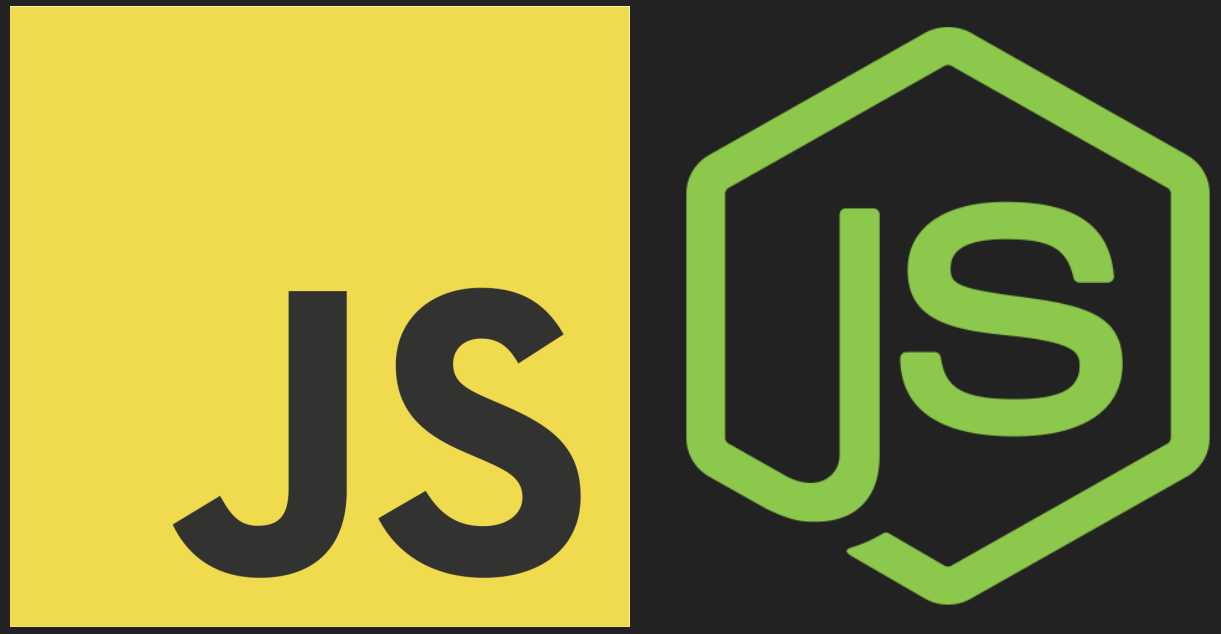
https://github.com/creationix/nvm

puis Node.js

```
$ nvm install node
```

TP

- Installer NVM
- Installer Node.js



ENVIRONEMENT DE DEVELOPPEMENT NODE.JS

DÉMARER NODE.JS

Node.js est et NVM sont maintenant installé sur votre machine.
Vous pouvez dès présent utiliser NVM pour lancer Node avec la commande:

```
$ nvm use 7
```

INITIALISER VOTRE PROJET

```
$ npm init
```

Cette commande va générer un fichier package.json qui décrit la configuration de votre projet.

Chaque module de NPM est configuré ainsi, ce qui fait que votre projet est par définition un paquet au même titre que les autres et donc potentiellement publiable sur NPM

INITIALISER VOTRE PROJET

NPM est un gestionnaire de paquet. L'usage le plus courant que l'on en fait est l'installation de dépendances depuis la plateforme npm.

```
$ npm install
```

ou

```
$ npm install --save
```

ou

```
$ npm install --save-dev
```

Logique et simple, il en va de même pour la suppression :

```
$ npm uninstall
```

TP

Step 5

-initialiser votre projet avec NPM
(utiliser le nom: '< your-name >-es6-training')

https://github.com/FazioNico/cours-es6

INTRODUCTION GULP

GULP TASKS RUNNER

Gulp est un outil qui est spécialiser dans le lancement de tâches.
C'est pour cela que l'on dit que Gulp est un Tasks Runner.

Gulp n'est pas capable de grand chose d'autres que de lancer des tâches préalablement déclarées et de passer d'une tâche à l'autres.

http://gulpjs.com

GULP INSTALLATION

Gulp est un module NPM et est donc téléchargeable via votre CLI
Installation sur votre machine:

```
$ npm install -g gulp
```

ENVIRONEMENT DE DEVELOPPEMENT NODE.JS

INTRODUCTION

Maintenant que vous avez les outils installé sur votre machine, nous allons pouvoir commencer à monter notre **stack de développement**.
C'est le terme à la mode pour designer les outils de travail d'un développeur Front-End.

C'est partit!!

AJOUTER LES DEPENCENCE GULP AU PROJET

Gulp à été installé en global sur votre machine mais vous devez aussi l'ajouter aux dépendances de votre projet afin que celui-ci puisse utiliser Gulp.

Voici la ligne de commande:

```
$ npm install --save-dev gulp gulp-util
```

On profite pour installer aussi gulp-util qui est l'utilitaire de Gulp.

AJOUTER LES DEPENCENCE GULP AU PROJET

Il faut maintenant ajouter un fichier de configuration pour Gulp.
C'est dans ce fichier que nous allons créer nos diverses tâches.

Dans votre CLI à la racine de votre projet:

```
$ touch gulpfile.js
```

Ensuite, ouvrez le fichier gulpfile.js et y ajouter:

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

Cela permet d'importer Gulp dans notre fichier de configuration.

UTILISER GULP AVEC SON PROJET - BABEL

Maintenant que Gulp est ajouter à notre projet, nous allons pouvoir l'utiliser pour lancer un autre outil qui va transformer notre code Es6 en code Es5...
tout seul!

Il s'appel Babel

```
 $ npm install --save-dev gulp-babel
$ npm install --save-dev babel-preset-es2015
$ npm install --save-dev babelify
$ npm install --save-dev vinyl-transform vinyl-source-stream
```

https://babeljs.io

UTILISER GULP AVEC SON PROJET - BABEL

Il faut ensuite, comme pour Gulp, importer Babel dans notre fichier de configuration (gulpfile.js) pour pouvoir utiliser cet outil.

Ajouter ces lignes au fichier gulpfile.js :

```
var babel = require('gulp-babel');
var babelify = require('babelify');
var transform = require('vinyl-transform');
var source = require('vinyl-source-stream');
```

Babelify est un plugin de Babel qui permet de convertir Es6 en Es5!
C'est pour utiliser entre autre ce plugin que nous faisons tous cela...

UTILISER GULP AVEC SON PROJET - BROWSERIFY

Il nous faut aussi un outil qui nous permet notamment d'utiliser la syntaxe require() de CommonJS dans le navigateur

Il s'appel browserify

```
$ npm install --save-dev browserify
```

Ajouter cette lignes au fichier gulpfile.js :

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

http://browserify.org

TP

Step 6

- installer Gulp en global sur votre machine
- ajouter Gulp comme dépendence à votre projet
- ajouter également Babel + Babelify + Browserify + Vinyl-Transform + vinyl-source-stream
- créer fichier de config pour Gulp
- (créer fichier .gitignore et y ajouter le repertoire 'node_modules' si vous utilisez Git)

https://github.com/FazioNico/cours-es6

UTILISER GULP AVEC SON PROJET

CRÉER UN TÂCHE GULP

Il est maintenant temps de créer notre 1ère tâche Gulp.

Son nom: 'build-js'
Elle devra s'occuper:

- trouver tous les fichiers JS Es6 de notre projet
- lire le contenu de ses fichiers
- convertir tout le code en Es5
- faire qu'un seul fichier avec tout le code de l'application
- créer un répertoire de développement avec arboresance
- y mettre le fichier JS

Voici comment elle se présente:

UTILISER GULP AVEC SON PROJET

Ajouter cette lignes au fichier gulpfile.js :

```
// importer les modules NPM
  var gulp = require('gulp');
  var babel = require('gulp-babel');
  var babelify = require('babelify');
  var browserify = require("browserify");
  var transform = require('vinyl-transform');
  var source = require('vinyl-source-stream');
```

UTILISER GULP AVEC SON PROJET

Ajouter cette lignes au fichier gulpfile.js :

```
// importer les modules NPM
var gulp = require('gulp');
var babel = require('gulp-babel');
var babelify = require('babelify');
var browserify = require("browserify");
var transform = require('vinyl-transform');
var source = require('vinyl-source-stream');

// Config of project folders
var config = {
  desDir: './dist' /* répértoire de destination (prod) */
}
```

UTILISER GULP AVEC SON PROJET

Ajouter cette lignes au fichier gulpfile.js :

```
// importer les modules NPM
var gulp = require('gulp');
var babel = require('gulp-babel');
var babelify = require('babelify');
var browserify = require("browserify");
var transform = require('vinyl-transform');
var source = require('vinyl-source-stream');

// Config of project folders
var config = {
  desDir: './dist' /* répértoire de destination (prod) */
}
// Task to build JS files
gulp.task("build-js", function(){
  // task instruction...
});
```

UTILISER GULP AVEC SON PROJET

Ajouter cette lignes au fichier gulpfile.js :

```
// importer les modules NPM
var gulp = require('gulp');
var babel = require('gulp-babel');
var babelify = require('babelify');
var browserify = require("browserify");
var transform = require('vinyl-transform');
var source = require('vinyl-source-stream');

// Config of project folders
var config = {
  desDir: './dist' /* répértoire de destination (prod) */
}
// Task to build JS files
gulp.task("build-js", function(){
 return browserify("dev/app/app.js",{
    debug: true
  })
  .transform(babelify.configure({
    presets : ["es2015"]
```

```
  }))

  .bundle()
  .pipe(source("bundle.js"))
  .pipe(gulp.dest(config.desDir + '/js'));
});
```

TP

Step 7

- Créer un tâche pour build les fichiers JS Es6
- modifier l'url du fichier JS de l'application dans le fichier index.html
- ajouter également Babel + Babelify + Browserify + Vinyl-Transform + vinyl-source-stream
- tester l'application

https://github.com/FazioNico/cours-es6

JOUR 4-5

ES6 DEVELOPPEMENT & PRODUCTION WORKFLOW

ALLER PLUS LOIN AVEC GULP

Vous avez vu comme Gulp peut être puissant.
Mais ce n'est encore rien!!!

On peut aller beaucoup plus loin que cela...
Accrochez-vous, c'est parti!

GULP - GESTION DES FICHIERS HTML

```
$ npm install gulp-remove-html-comments --save-dev
```

```
// gulpfile.js

var removeHtmlComments  = require('gulp-remove-html-comments');

gulp.task("copy-html", function(){
  return gulp.src(['./dev/**/*.html'])
  .pipe(rmHtmlComments())
  .pipe(gulp.dest(config.desDir))
});
```

Maintenant utilisez l'url **'dist/index.html'** pour afficher l'application

TP

Step 8

- Créer un tâche pour copier les fichiers HTML dans le répértoire de production: 'dist'
- modifier l'url du fichier JS de l'application dans le fichier index.html
- tester l'application

https://github.com/FazioNico/cours-es6

GULP - LIVE RELOAD

GULP - LIVE RELOAD

```
$ npm install --save-dev browser-sync
```


```
// gulpfile.js
var browserSync = require('browser-sync').create();
var reload = browserSync.reload;
// Task to run local server
gulp.task("startServer",  function() {
  browserSync.init({
    server: {
        baseDir: config.desDir
    },
    notify: true
  });
});

// Task to watch wich file is changing
// and load the right task
gulp.task('watch', function() {
  // watch js file changes
  gulp.watch('./dev/app/**/*.js', ['build-js']);
  // watch all html template file changes
  gulp.watch('./dev/**/*.html', ['copy-html']);
});
```

```
// add this in task you want for reload
```

```
// in the end of the task
.pipe(reload({stream:true}))
```

https://www.browsersync.io

BABEL

Browsersync

STACK FRONT-END

ON PEUT ALLER ENCORE PLUS LOIN AVEC GULP...

Gestion des dépendances Bower, des fichier sass, css, images, etc...

Je vous invite à regarder ce fichier qui contient quelques tasks qui vous seront utiles:

https://github.com/FazioNico/simple-es6-front-ent-stack/blob/master/gulpfile.js

DES QUESTIONS?

FIN ;-)