TP 15: jQuery, Node.js et frameworks

Objectifs:

- Comprendre les bases d'usage de la bibliothèque jQuery
- Découvrir Node.js et l'installation de modules avec npm
- Découvrir la création d'une Single Page App avec Augular et React

---

Convertir un composant jQuery en JavaScript/DOM pur

Références:

- You Might Not Need jQuery
- Plain JS

Exercice 1: convertir un composant accordéon

- Tester et analyser le jsfiddle d'exemple
- Forker ce jsfiddle, puis le modifier pour qu'il ne contienne plus de jQuery:
  i. Fonction associée au clic sur *un seul* menu
  ii. … puis sur *chaque* menu (à l'aide d'une boucle)
  iii. Cacher les éléments .content
  iv. Afficher l'élément .content du .menu qui a été cliqué

Exercice 2: convertir un autre composant (au choix)

Exemples de composants:

- Slider: Simple Slider ou Craftyslide
- Nombre animé: codepen, jsfiddle, ou github

Node.js et npm

Introduction

En début d'année, nous avons vu qu'il existait différents types d'interpréteurs JavaScript.
Exemples:

- la console de Chrome permet d'exécuter des instructions JavaScript de manière interactive.
- un fichier JavaScript intégré dans une page web est exécuté dès l'ouverture de cette page dans un navigateur.

Il existe un autre moyen d'exécuter du code JavaScript: Node.js.
Node.js est un logiciel basé sur le moteur d'exécution intégré dans Google Chrome: V8. Il est constitué de deux commandes exécutables depuis le "terminal" (la console du système d'exploitation): node et npm.
La commande node permet d'exécuter du code JavaScript, alors que npm permet de télécharger puis installer des modules (appelés *packages*) de manière locale (dans le dossier en cours) ou globale (=> modules accessibles depuis n'importe quel dossier).
Node.js est couramment utilisé pour développer des serveurs web (aussi appelé *back-end*, dans le cas de développement d'applications client-serveur telles que les applications web), mais permet aussi la réalisation de scripts ayant accès à toutes les fonctions du système d'exploitation: système de fichiers, accès illimité aux ressources de n'importe quel réseau (dont le web), etc…
npm est couramment utilisé pour installer les dépendances, c'est à dire les modules nécéssaires par un programme JavaScript qui sera exécuté par Node.js.
À noter que la plupart des frameworks *front-end* actuels recommandent l'usage de la commande npm pour les télécharger sur votre machine, ainsi que vous installer les éventuelles bibliothèques, composants et autres extensions dont votre application auront besoin.

Installer et tester Node.js sur sa machine

Après avoir ouvert le terminal, taper node pour exécuter la version interactive de Node.js.
Pour lancer le "terminal":

- sous Windows, appuyer sur la touche "Windows", taper "cmd" puis presser ENTRÉE.
- sous Mac, presser Cmd-Espace, taper "term" puis presser ENTRÉE.

Si la commande node n'est pas trouvée par le système, il faudra installer Node.js depuis son site officiel, puis ré-ouvrir le terminal avant de poursuivre la manipulation.
Une fois Node.js lancé, vous pouvez taper des instructions, comme dans la console de Chrome Dev Tools.
Par exemple: tapez 1+1; pour voir le nombre 2 s'afficher en retour.
Pour quitter l'interpréteur Node.js, pressez Ctrl-C.

Exécuter un fichier .js avec Node.js

Nous allons d'abord créer un fichier test.js très simple, en tapant la commande suivante dans le terminal:

```
echo "console.log('bonjour');" >test.js
```

Sous Windows, vous aurez peut-être besoin de retirer les guillemets.
Ensuite, pour exécuter ce fichier, il suffit alors de taper la commande suivante depuis le terminal:

```
node test.js
```

Vous devriez alors voir s'afficher la ligne suivante:

```
bonjour
```

À noter que les symboles tels que window, document, alert() et getElementById() ne sont pas définis par Node.js, car ils sont seulement fournis lorsqu'un programme JavaScript s'exécute depuis un navigateur Web !

Pour aller plus loin avec Node.js

Tutorials de grafikart:

- Qu'est ce que NodeJS ?
- Installation de NodeJS
- Créer un serveur avec NodeJS
- Utilisation de modules
- Création d'un serveur Web avec Express (ou depuis site officiel)
- Serveur de tchat avec Socket.io

Ou suivre le cours Node.js sur Openclassrooms.

Single Page Apps avec Augular et React

Front-end VS *back-end*

Une application Web moderne consiste en:

- un client / *front-end*: programme exécuté par le navigateur de chaque utilisateur,
- et un "serveur web" / *back-end*: programme exécuté sur une machine appelée "serveur", qui a pour mission de répondre aux requête des "clients" (et donc de chaque utilisateur).

Comme nous l'avons vu, un client "web" est généralement constitué de fichiers HTML, CSS et JavaScript ayant accès à l'API du DOM fournie par le navigateur. À moins qu'il utilise des moyens de persister ses données (ex: à l'aide de cookies, LocalStorage, ou requêtes AJAX vers un serveur), son "état" (c.a.d. toutes les données de l'utilisateur saisies dans la page, et valeurs des variables) est réinitialisé à chaque rafraichissement de la page.
Un "serveur web" est rattaché à une adresse IP (ex: 127.0.0.1, ou nom de domaine) et un port (ex: 80), et permet de réagir et répondre à des requêtes HTTP envoyées par des "clients" à différents chemins (ex: /posts) et avec différentes méthodes (ex: GET, POST, etc…). Un "serveur

web" peut être implémenté en nombreux langages: PHP, Java, ASP, etc… Node.js peut être utilisé pour exécuter un serveur web implémenté en JavaScript.
Il est relativement facile et rapide de développer un client web simple: il suffit d'une page HTML associée à un fichier JavaScript.
Par contre, dès qu'une application web contient plusieurs écrans et/ou modes (ex: utilisateur connecté ou pas), elle devient vite complexe à écrire et maintenir.

Frameworks et bibliothèques front-end

Pour mieux structurer le code et le fonctionnement d'applications web riches, nombreux *frameworks* et bibliothèques (*libraries*) ont été développés. Ils sont généralement inspirés du principe d'architecture **MVC** (Modèle-Vue-Controlleur, cf définition sur Wikipedia).
Notamment, pour aider au développement de clients web, les technologies suivantes sont actuellement très en vogue:

- Angular (en JavaScript ou TypeScript), développé par Google,
- React, développé par Facebook,

(Il en existe nombreuses autres: Polymer, Ember.js, Vue.js…)
Suivez les guides liés à ces deux technologies pour comprendre leur fonctionnement par la pratique.

Solutions hybrides

Il existe aussi des bibliothèques et frameworks permettant de faciliter le lien entre front-end et back-end:

- Firebase permet de créer une base de données sur internet, et d'y accéder directement depuis un client web, et se présente donc comme une alternative simple à un serveur web / back-end. Solution valable quand le plus gros de son application s'exécute dans le navigateur. (consulter slides et vidéos de mon cours de l'an dernier et/ou guide officiel)
- Basé sur Node.js et MongoDB, Meteor permet de créer des applications web JavaScript *isomorphiques*. C'est à dire que des parties du code peuvent s'exécuter à la fois côté serveur et/ou côté client (dans le navigateur). Un de ses gros avantages est de permettre l'intégration de composants gérant à la fois la partie "client" et la partie "serveur", ex: bouton d'identification d'utilisateur avec stockage automatique dans la base de données de l'application. (suivre tutorial sur grafikart)

Ces deux solutions sont très pratiques pour réaliser des prototypes de manière rapide / efficace.