COURS ADAPTATIONS

Rapport de projet

27 OCTOBRE 2016

UNIVERSITEE DE NICE/SOPHIA

Site des templiers

Description du rendu

Rapport à livrer AVANT LE jeudi 27 octobre MIDI sur le site Le rapport attendu doit comprendre :

- Une introduction qui décrit et argumente l'exemple choisi
- Une partie par technologie décrite chacune sous la forme d'un Tutorial : Comment avez-vous réalisé l'exemple ? Avec quel outil de développement, de tests ? Comment déploie-t-on et exécute-t-on l'exemple ? Comment teste-t-on les capacités d'adaptations ?
- Une conclusion qui fait une synthèse des 4 expériences.

Lien du site avec sujet :

- http://atelierihm.unice.fr/enseignements/plasticite-des-interfaces/presentation-et-planning/

Table des matières

Introduction

Pour débuter ce rapport nous allons nous pencher sur une description du projet et des choix qui nous ont poussé à choisir ce projet plutôt qu'un autre.

Description du projet

Nous avons décidé de réaliser cette application dans l'optique d'inspirer fidéliser et inciter l'utilisateur à partager ses avis dans le monde du cinéma.

- La fonctionnalité principale disponible depuis la page d'accueil est l'affichage des films dont une séance est disponible dans un cinéma à proximité.
- L'utilisateur peut accéder au détail du film et, par exemple, visionner la bande annonce du film, lire le résumé, voir les différentes heures de projection ainsi que les autres salles projetant ce film.

Explication du choix

Nous avons décidé de réaliser cette application car nous avons trouvé qu'il nous permettrait d'aborder une grande variété d'adaptation. En effet, grâce aux nombreuses fonctionnalités de notre application (système de localisation, de notification, etc....), nous pouvons facilement réaliser différents types d'adaptations tels que :

- L'adaptation au device : notre application devra pouvoir être utiliser sur différents systèmes d'exploitation et différents devices (smartphone, tablette, ordinateur, etc....) ;
- Adaptation à l'utilisateur : notre application pourra notifier l'utilisateur en fonction de son quotidien. Par exemple, proposer une séance de cinéma à la fin de sa journée de travail.
- Adaptation à l'environnement : notre application guidera l'utilisateur au cinéma de son choix ou lui donnera une liste de cinéma à sa portée sur une map.

Technologies utilisées

Dans cette partie chaque personne du groupe détaillera sa technologie sous forme d'un petit tutoriel.

Electron

Electron est un Framework **open source** développé par « **GitHub** », il permet de développer des interfaces graphiques « **Cross Platform** » (Windows, MacOs et linux) en utilisant le « **runtime** » de « **Node.js** » et le navigateur « **Chromium** ».

Lien : http://electron.atom.io/

Electron utilise les technologies du web comme JavaScript, HTML et CSS, il s'occupe de toutes les parties compliquées du développement d'une application pour laisser le développeur s'occuper entièrement de celle-ci.

« Vous trouverez ci-dessous la vidéo de présentation »

Lien : https://www.youtube.com/watch?v=8YP_nOCO-4Q&feature=youtu.be

Technologies ajoutées

Pour aider au développement avec électron j'ai mixé celui-ci avec plusieurs technologies du web pour améliorer la productivité de mon développement.

Cocktail

Développé par notre cher ami Sofiane NAIT OUSLIMANE, « **Cocktail** » est une surcouche à Gulp (gestionnaire de tâches) permettant de faciliter tout le processus de compilation, compression, minimisation, optimisation en proposant des tâches pré faites et simple d'utilisation. Vous pouvez voir son utilisation dans le gulpfile.js

Susy

Pour le projet j'avais besoin de créer des grilles en CSS plusieurs solutions s'offraient à moi comme par exemple Bootstrap, mais dans ce cas on brise la règle qui est de séparer le fond (HTML) de la forme (CSS), qui peut s'avérer un frein lorsqu'on que l'on souhaite faire évoluer le site web (Devoir changer le HTML pour changer le positionnement des éléments).

Susy est un système qui nous permet de gérer la disposition des éléments sans polluer le code HTML. Grace à un system de grille, de breakPoint (Couplé avec un plugin) le tout directement en SASS.

```
main{ @include span(2 of 3); }
sidebar{ @include span(1 of 3); }
@include breakpoint(500px) {
    main{ @include span(10 of 12); }
    sidebar{ @include span(2 of 12 last); }
}
```

Lien : http://susy.oddbird.net/

Sass and Jade

Jade

Jade est un « **engine template** » implémenté avec JavaScript pour Node.js et navigateur. Il permet d'écrire l'html plus simplement.

```
block script
  // build:js-vendors
  script(type="text/javascript", src="../resources/vendors/vendors.js")
  // endbuild
  // build:js
  script(type="text/javascript", src="../js/main.js")
  // endbuild

// *************
// Header
// *************

block header
  .header-block
    .resize-bar
    .windows-panel
      .close-block.icon-block
        span.icon-close
      .maximize-block.icon-block
        span.icon-crop_3_2
      .reduce-block.icon-block
        span.icon-remove
    .user-block
      p Kashin
      span.icon-user
```


Lien : http://jadelang.net/

SASS/SCSS

Sass (Syntactically Awesome Stylesheets) est un langage de génération dynamique de feuilles de style. Il est un « **métalangage de feuilles de style en cascade (CSS)** ». C'est un langage de script qui est interprété en CSS. SassScript est le langage de script lui-même.

```
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

Lien : http://sass-lang.com/

Tutorial

Dans cette partie nous verrons sous la forme d'un petit tutorial comment installer et utiliser le projet dans un premier temps.

Puis nous verrons avec des exemples commentés, comment implémentées les fonctionnalités propres aux adaptations choisies.

Installation

Commençons par installer les dépendances du system.

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**.

Lien : https://nodejs.org/en/

Lors de l'installation de node.js « NPM » est installé avec, il est le **gestionnaire de paquets** officiel de Node.js

Electron

Avec « NPM » installé sur l'ordinateur, l'installation d'Electron est un jeu d'enfant, grâce à cette commande :

```
C:\Users\KashiN
λ npm install -g electron
```

L'option « -g » permet de faire une installation globale dans le system voir lien ci-dessous :

Lien : https://docs.npmjs.com/getting-started/installing-npm-packages-globally

Cocktail

« Cocktail » contient des dépendances spécifiques disponible dans un répertoire privé, il faut donc configurer « NPM » pour lui dire où les trouvées.

```
C:\Users\KashiN
λ npm config set registry                    repository/npm-public/
```

« Cocktail » possède un template vide d'application Electron prête à l'emploie avec jade et SASS de configurer.

Une fois le git du projet clone il suffit d'utiliser la commande suivante :

```
C:\Users\KashiN
λ npm install
```

Cette commande installe tous les paquets dont le projet est dépendant.

Pour lancer rien de plus simple :

```
C:\Users\Kashin\Desktop\WorkSpace\Inspire\electron (master) (Inspire@0.0.1)
λ gulp --watch --platform desktop --electron
```

Sans Cocktail

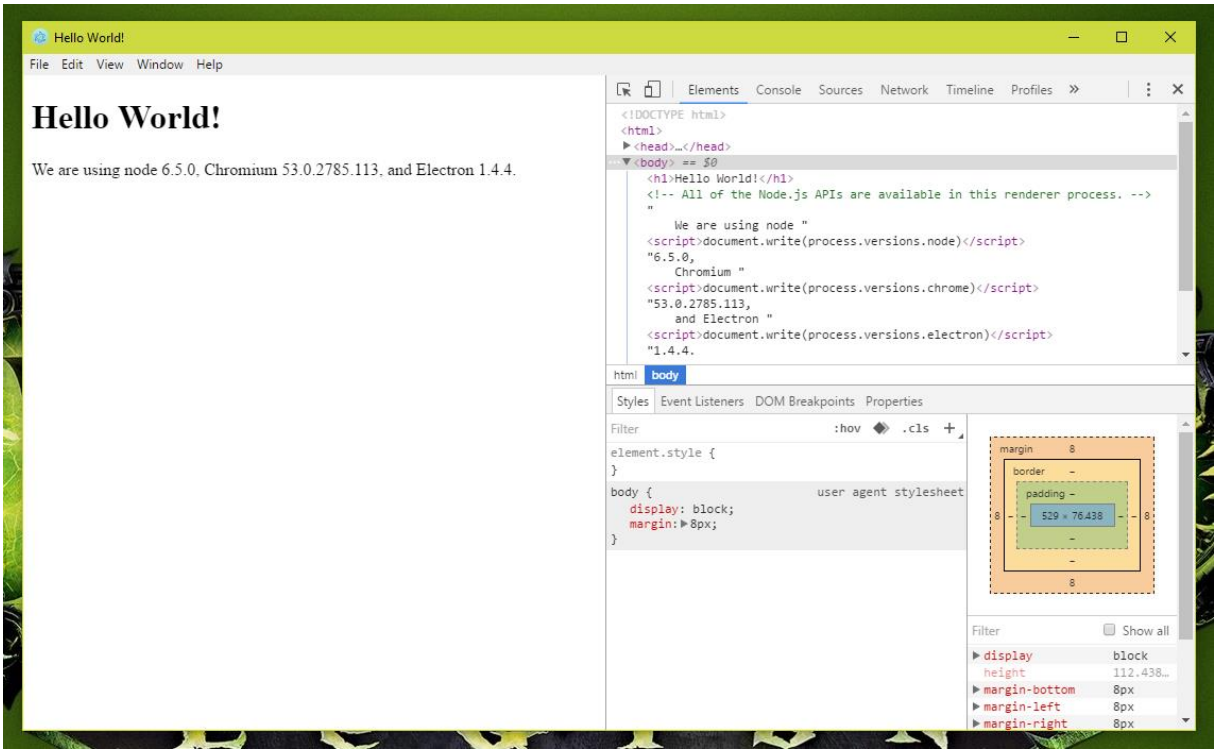
Imaginions que je n'avais pas « Cocktail » à disposition l'installation d'Electron est tout aussi facile.

```
# Clone the Quick Start repository
$ git clone https://github.com/electron/electron-quick-start

# Go into the repository
$ cd electron-quick-start

# Install the dependencies and run
$ npm install && npm start
```


- La première commande clone leur répertoire qui contient une application de base.
- Ensuite dans le dossier cloné nous allons installer les dépendances du nouveau projet.
- Puis il ne nous reste plus qu'à lancer l'application



Electron application basique

Dans cette partie nous verrons comment est implémentée basiquement une application « Electron »

Structure des dossiers

Généralement une application « Electron » possède une structure de cette forme :

```
mon-application/
        |------ package.json
        |------ main.js
        |------ index.html
```


Package.json

Le fichier « package.json » contient les informations de base de l'application comme par exemple son nom.

```
{
        "name"    : "Mon-application",
        "version" : "0.1.0",
         "main"    : "main.js"
}
```

Main.js

```
const {app, BrowserWindow} = require('electron')

// Keep a global reference of the window object, if you don't, the window will
// be closed automatically when the JavaScript object is garbage collected.
let win

function createWindow () {
  // Create the browser window.
  win = new BrowserWindow({width: 800, height: 600})

  // and load the index.html of the app.
  win.loadURL(`file://${__dirname}/index.html`)

  // Open the DevTools.
  win.webContents.openDevTools()

  // Emitted when the window is closed.
  win.on('closed', () => {
    // Dereference the window object, usually you would store windows
    // in an array if your app supports multi windows, this is the time
    // when you should delete the corresponding element.
    win = null
  })
}

// This method will be called when Electron has finished
// initialization and is ready to create browser windows.
// Some APIs can only be used after this event occurs.
app.on('ready', createWindow)

// Quit when all windows are closed.
app.on('window-all-closed', () => {
  // On macOS it is common for applications and their menu bar
  // to stay active until the user quits explicitly with Cmd + Q
  if (process.platform !== 'darwin') {
    app.quit()
  }
```

Le fichier « index.js » sera lancé au lancement de l'application, il s'occupera de créer les fenêtres et de la partie contrôle de l'application

Index.html

Finalement le fichier « index.html » sera affiché dans l'application « Electron ».

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Hello World!</title>
  </head>
  <body>
    <h1>Hello World!</h1>
    We are using node <script>document.write(process.versions.node)</script>,
    Chrome <script>document.write(process.versions.chrome)</script>,
    and Electron <script>document.write(process.versions.electron)</script>.
  </body>
</html>
```

Réalisation des adaptations.

Maintenant que nous avons une application qui fonctionne il ne nous reste plus qu'à implémenter les fonctionnalités voulues pour répondre au **besoin d'adaptation**.

Adaptation multiplateforme

Pour répondre à cette adaptation en plus d'utiliser « d'Electron » j'ai eu recourt à « Susy ».

De base « Electron » me permet de viser des **plateformes différentes** (Windows, MacOs et linux). Grâce au module Node « OS » il est très simple de savoir sur quel Os on se trouve.


```
const os = require('os');
var osArch = os.arch();
var osPlatform = os.platform();
```

Lien : https://nodejs.org/api/os.html#os_os_release

Ensuite il ne me reste plus qu'à ajouter une class CSS sur mon application pour styliser mon affichage sur chaque Os.

```
▶ <div class="app windows">…</div> == $0
```

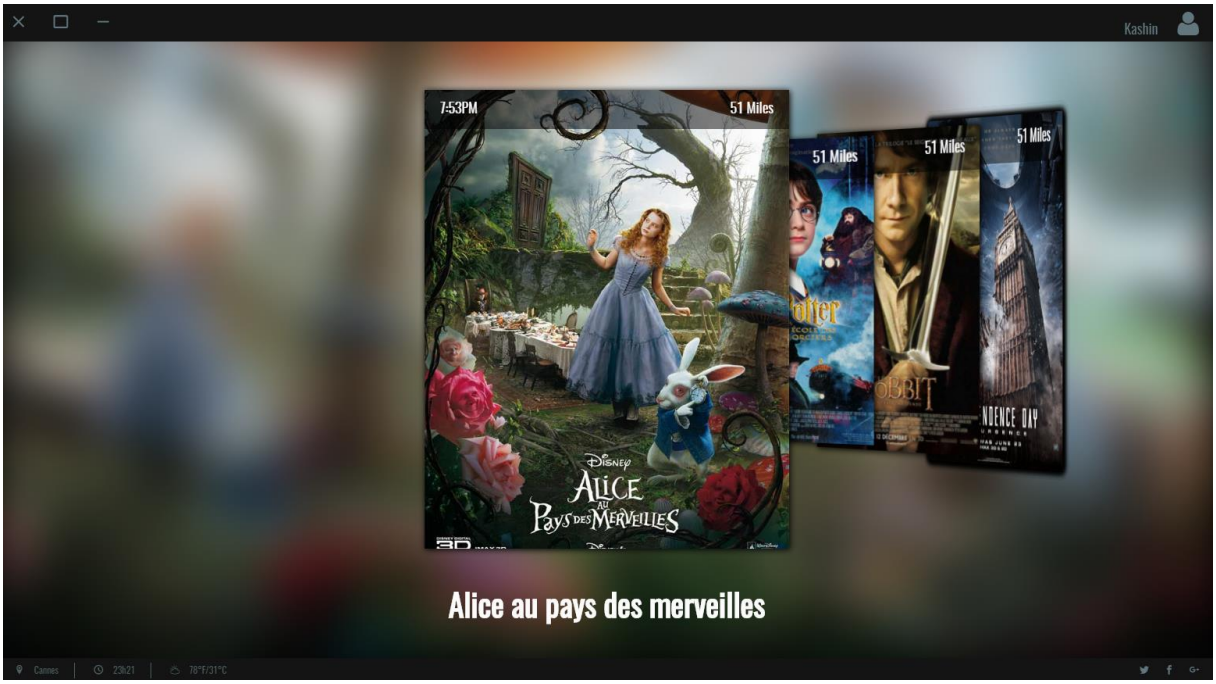
Couplé à ça, « Susy » me permet de gérer **différentes tailles d'écrans** allant du plus petit au plus grand.

Comme nous pouvons voir dans cet exemple grâce à « Suzy » j'adapte facilement la vue de mes éléments à la taille de la fenêtre.

```
@include from('d') {
    transform: perspective(3000px) rotateY(45deg) translateX(- 200%);
}
```

Sur cet exemple je préviens que lorsque l'écran sera plus grand que « **1280px** » l'animation changera.

J'ai aussi activé le « FrameLess » pour rendre l'application plus adaptable sur desktop.



Lien : http://electron.atom.io/docs/api/frameless-window/

Quant à « Electron » j'ai déjà certaines limitations à l'adaptation, en effet le portage de certaines fonctionnalités sur Linux laissent encore à désirer et de nombreux bug sont reportés dans les forums.

Adaptation Localisation

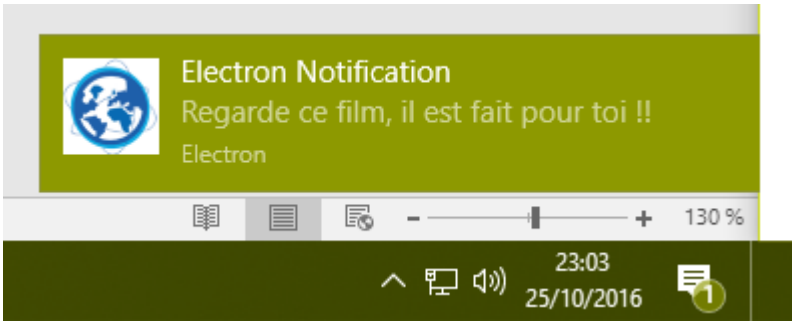
Ici nous nous trouvons directement sur un exemple de limitation. Suite aux mises à jour de « Chromium », « Electron » doit adapter son API aux nouveautés, mais cela peut prendre du temps.

La fonction de localisation n'est actuellement pas disponible, du a des mises à jour de « Chomium », le bug a été reporté sur les forums.

Adaptation notification

Pour utiliser les notifications du Systems, « Electron » nous propose d'utiliser l'objet « Notification » d'HTML5.

Lien : https://developer.mozilla.org/fr/docs/Web/API/notification

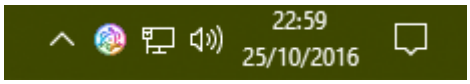


Avec comme particularité que nous n'avons pas à demander l'autorisation à l'utilisateur.

```
/**
 * Notify the user with a specific message.
 * @param message The message for the user.
 */
function doNotify(message) {
    Notification.requestPermission().then(function (result) {
        if (canNotify) {
            var myNotification = new Notification('Electron Notification', {
                'body': message,
                'icon': `file://${__dirname}/../resources/images/touch/icon-128x128.png`
            });
            canNotify = false;
            setTimeout(function () {
                canNotify = true;
            }, 30000);
        }
    });
}
```


« J'ai rencontré quelques bugs mais je pense qu'ils provenaient de mon ordinateur. »

Pour améliorer l'adaptation notification et lui donnée tout son sens sur desktop j'ai mis l'application dans le « Tray » de l'os (A savoir qu'il n'y en a pas de base sous la plupart des distributions linux). Quand l'application est dans le « Tray » elle continue à notifier l'utilisateur si par exemple une séance de cinéma est en promotion.



Lien : http://electron.atom.io/docs/api/tray/

Ionic 2

Introduction

Ionic 2 est un framework permettant de développer des applications hybrides au **look and feel** proche des différents OS suivant : Android, Windows, iOS

Ce framework repose notamment sur **Cordova**, qui exécute notre application grâce à une **webview** et nous permets d'accéder à des capacités natives telles que les **capteurs** ou encore des entrées (Contacts, Sms, ...). Celui-ci repose aussi sur un framework basé sur le pattern web-component, à savoir **Angular 2**.

L'intérêt principal d'un tel framework est de pouvoir **capitaliser** sur les compétences acquises en Html, Css et Javascript.

Ionic 2 est actuellement en **release candidate**, c'est pourquoi il n'est pas impossible de rencontrer des problèmes lors du développement.

Installation

Vous devez au préalable installer node (>= 5.5.0) et npm (>= 3.3.12).

(Pour vérifier les versions présentes sur vos machines : node -v et npm -v)

Vous devez par la suite installer Ionic et Cordova grâce à la commande suivante :

- npm install -g ionic cordova

Création d'un projet

Si l'installation s'est déroulée sans encombre, il suffit d'utiliser la commande ci-dessous afin de créer un nouveau projet vide et **installer les dépendances** :

- ionic start myApp blank --v2

Vous déplacez dans votre nouvelle application :

- cd myApp

Vous pouvez dès maintenant démarrer votre application grâce à la commande suivante :

- ionic serve

Ou bien celle-ci, si vous désirez afficher les 3 OS en même temps :

- ionic serve -l

Ionic met à disposition un générateur que l'on peut utiliser de la manière suivante :

- ionic g page myPage
- ionic g provider myProvider
- ionic g component myComponent

Configuration

Le principal intérêt d'un tel framework est comme dit précédemment, de pouvoir **déployer** sur différentes cibles, cependant nous devons définir nos cibles, comme suit :

- ionic platform add android

Les plateformes disponibles sont indiquées sur le site d'Ionic, mais voici les principales :

- Android, Windows, Ios, browser

Outils de développement et de tests

Un éditeur de texte tel que Atom, Sublime Text ou encore un IDE comme Webstorm font très bien l'affaire pour le développement.

Afin de pouvoir tester notre application, un **navigateur** tel que Chrome suffit. De plus en appuyant sur F12 ou Ctrl+Maj+i, vous pouvez accéder à l'inspecteur de Chrome.

Afin de tester l'application sur un **émulateur** comme android par exemple, il suffit d'utiliser la commande suivante :

- ionic emulate android

Et comme pour la version navigateur, vous pouvez accéder à l'inspecteur de Chrome via l'url suivante :

- chrome://inspect/#devices (copier – coller ce lien sur chrome)

Adaptations

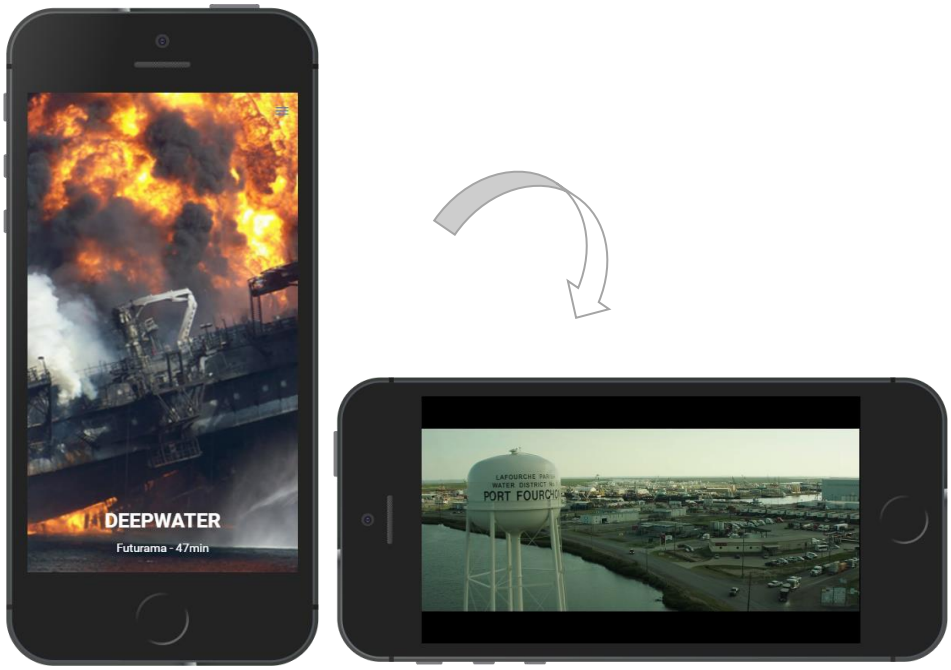
Adaptation au device :

- Affichage d'un trailer lorsque le device est en mode paysage (Gyromètre)

Notre application comme son nom l'indique consiste à inspirer l'utilisateur en lui suggérant des films projetés proche de lui. Cependant, la plupart des utilisateurs visionnent la bande annonce d'un film afin d'avoir un avis et de potentiellement acheter un ticket.

La plupart des vidéos visionnées sur smartphone et tablette le sont en mode paysage, en effet l'utilisateur va au préalable lancer la vidéo puis faire pivoter son device en mode paysage.

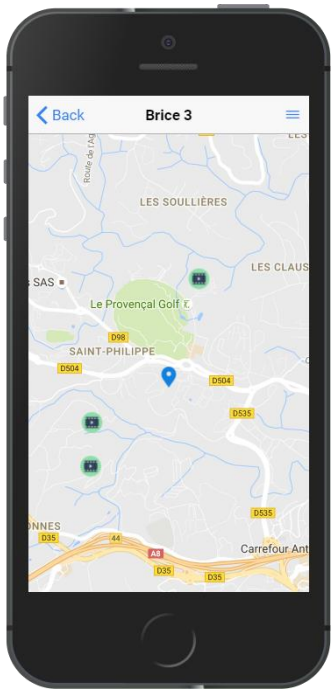
Cependant grâce aux capteurs du device nous pouvons déterminer si celui-ci est en mode paysage ou portrait et ainsi déclencher la lecture de la vidéo en plein écran, ainsi nous supprimons une action superflue.



Adaptation à l'environnement :

- Géolocalisation de l'utilisateur afin d'afficher les cinémas proches (Capteur GPS)

Avant de pouvoir acheter un ticket, il faudrait mieux connaitre les cinémas qui nous entoures, c'est pourquoi à l'aide du capteur GPS disponible sur notre device nous pouvons déterminer les cinémas les plus proches de notre utilisateur et ainsi proposé des résultats plus probant. (Les cinémas les plus proche à un moment T ne seront pas forcément ceux les plus proches lorsque l'utilisateur est sur son ordinateur … à domicile)



Adaptation à l'utilisateur :

- Filtrage des films proposés selon les préférences de l'utilisateur

Enfin, il est possible de trier/filtrer les films proposés à l'utilisateur selon ses préférences.

Adaptions futures envisageables :

- Filtre/trie sur plus de paramètres (Cinéma, Acteur, Auteur, …)
- Mise en commun avec l'entourage de l'utilisateur
- Notifications selon la position de l'utilisateur à un moment T (Proposition de popcorn juste avant que la séance commence, Donner son avis quelques minutes après la fin de la séance)
- Mise en mode avion/silencieux/nuit, lorsque la séance est sur le point de commencer.

Déploiement

Nous développons une application hybride, il est donc préférable de tester notre application directement sur notre **device**, pour cela il suffit de brancher le device en question et d'utiliser la commande suivante pour android :

- ionic run android

En ce qui concerne iOS, il faudra vous munir d'un Mac ou bien passer par une solution en ligne.

Angular Js

Angular est un framework web développé par Google. C'est la version 2.0 qui est utilisé pour Inspire, cette nouvelle version d'Angular est orienté mobile avec des performances améliorées par rapport à la première version mais également l'avantage d'être component based, comme nous allons le voir plus tard dans ce tutorial.

Installation des outils et mise en place du projet

Pour simplifier la mise en place du projet et accélérer le développement, nous utilisons l'utilitaire Angular CLI. Cet outil, développé et maintenu par la communauté d'utilisateurs d'Angular, fonctionne en ligne de commande (d'où son nom CLI - Command Line Interface) et permet de créer une base de projet Angular, lancer les tests unitaires, lancer un serveur local pour tester l'application, et bien plus encore.

Les deux pré requis indispensable pour travailler sur un projet Angular sont NPM en version 3 ou supérieur ainsi que NodeJS en version 4 ou supérieur. Pour vérifier qu'ils sont installés vous pouvez utiliser les commande *npm -v* et *node -v* dans un terminal.

```
C:\Users\Cédric\Documents\git\Inspire\angular>npm -v
3.8.3

C:\Users\Cédric\Documents\git\Inspire\angular>node -v
v5.10.0
```

Fig. 1 : Vérification de la version de NodeJS et NPM

Si vous n'avez pas ces outils ou n'avez pas une version à jour sur votre ordinateur, téléchargez la dernière version de nodeJS sur le site officiel de l'éditeur (cf. https://nodejs.org/en/), cet installateur va se charger d'installer à la fois nodeJS et NPM. Une fois l'installation terminé, vérifier que l'installation se soit déroulé correctement en testant la version de NPM et NodeJS comme expliqué ci-dessus. (Dans le cas où vous travaillez sous Windows, n'oubliez pas d'ajouter les exécutables de NPM et NodeJS au PATH pour y avoir accès dans le terminal).

Il est maintenant temps d'installer Angular CLI, pour cela rien de plus simple : rendez-vous dans un terminal et lancer la commande : *npm install -g angular-cli* puis attendez la fin de l'installation. Rendez-vous maintenant dans le dossier ou vous souhaitez créer votre application Angular, toujours en utilisant la console, et lancer la commande *ng new Inspire --style=sass.* Angular CLI va créer un nouveau projet appelé Inspire et comportant déjà la structure d'une application Angular 2.

Comme indiqué par l'option "style" lors de la création de l'application nous utilisons ici le preprocessors SASS. Il est également possible d'utiliser du CSS simple pour réaliser une application Angular 2. SASS est utilisé ici par préférence, en effet il permet de définir des mixin simplifiant les media queries, d'utiliser des variables, ou encore d'avoir une meilleur structure grâce au nesting.

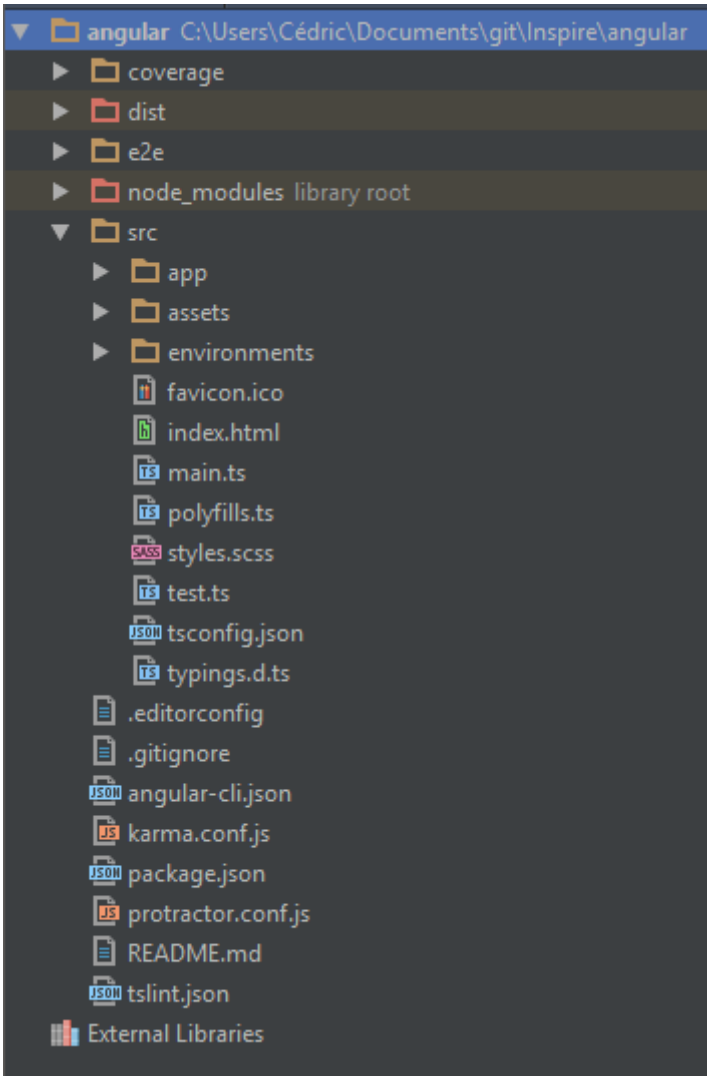


Fig. 2 : Structure de notre application Angular 2

Développement de l'application

Maintenant que la structure de l'application est en place, nous pouvons commencer le développement de celle-ci. Comme toute application web, le point de départ de l'application est le fichier *index.html*.

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8">
  <title>Inspire</title>
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">

</head>
<body>
  <app-root>Loading...</app-root>
</body>
</html>
```

Fig.3 Fichier index.html de notre projet

La partie important se trouve dans le body, comme nous le voyons sur la figure 3, la seule balise html est *app-root.* Cette balise est le point de départ de la logique Angular et son contenu va être changer dynamiquement.

Le point de départ de l'application Angular est quant à lui le composent appelé "app". C'est ce composant qui est le parent de tous les autres composants que nous allons créer.

```
router-outlet
<router-outlet></router-outlet>
```

Fig.4: Template du composant app

Comme on peut le voir sur la figure 4, notre application ne contient que la balise *router-outlet*. Ceci s'explique par le fait que j'ai décidé de séparer mon application en pages. Cette balise indique que les pages accédées via le routage vont être "render" dans ce fichier. Etant donné que le composant app est également ce qui va être dynamique remplacé dans le app-route du index.html, notre application va afficher les pages générés par le routage.

```
@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    ShowComponent,
    HeaderComponent,
    DetailViewComponent,
    DetailTabsComponent,
    FilmDescriptionComponent,
    FilmCommentComponent,
    NotificationComponent,
    SettingComponent
  ],
  imports: [
    BrowserModule,
    FormsModule,
    HttpModule,
    RouterModule.forRoot([
      { path: 'show/:id', component: DetailViewComponent },
      { path: 'setting', component: SettingComponent},
      { path: '', component: HomeComponent },
      { path: '**', component: AppComponent }
    ])
  ],
  providers: [ShowService],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Fig.5: Snippet du fichier app.module.ts de notre application

Sur la figure 5 nous voyons la configuration de notre application effectuée dans le ngModule du composant *app*. C'est ici que tous les composants utilisés au sein de l'application doivent être importés, il est également possible d'importer des modules et de les configurer comme je l'ai fait pour le router ou j'ai configuré chaque route existante.

Maintenant que notre routage est en place, nous avons un composant par page. Chacun de ces composant va également utiliser des sous composants.

Lancer et tester l'application

L'application peut être testé via des tests unitaire, chaque composant à un fichier "spec" qui lui est associé. Pour lancer les tests il faut utiliser la commande *ng test*, ce projet étant petit et ne comprenant que peu de logique, j'ai décidé de ne pas écrire de test unitaire.

Pour déployer l'application sur un serveur local, il faut lancer la commande *ng serve* à la racine du projet, l'application est ensuite déployée et accessible à l'adresse http://localhost:4200/.

Comme nous l'avons vu précédemment, l'application est composée de 3 pages, la page d'accueil qui affiche la liste des séances de cinéma à proximité de l'utilisateur, la page de détail qui montre les détails d'une séance, et la page de configuration.

Capacités d'adaptation de l'application

Vous souhaitez utiliser l'application avec votre téléphone portable, ou votre ordinateur ? Pas de problème, vous pouvez faire les deux ! L'interface de l'application est responsive et permet une utilisation agréable du site web peu importe l'appareil utilisé et la taille de l'écran.

Fig.6 : Exemple du responsive design sur la liste de film de la page home

Comme nous le voyons sur la figure 6, la liste des séances est affichée différemment en fonction de la taille de l'écran. Mais ce n'est pas la seule adaptation, en effet, Inspire s'adapte également au nombre de séances a affiché grâce à une boucle effectué avec la directive ngFor. Il serait même possible de prendre en charge le lazy loading, c'est à dire ne charger que quelques séances et en charger davantage quand l'utilisateur arrive vers le bas de la liste avec le scroll. Cependant, cette technologie n'est pas très utile ici puisque nous ne pouvons pas dépasser la vingtaine de séances. (Il n'y a jamais plus d'une vingtaine de films à l'affiche simultanément)

La deuxième adaptation mise en avant dans Inspire est la prise en compte de la position de l'utilisateur pour afficher les séances dans les cinémas à proximité. Ainsi, en fonction de la position ou l'application est utilisé, les séances affichées ne seront pas les mêmes. L'unique gêne pour l'utilisateur, dans le cas d'une utilisation sur ordinateur, est le fait qu'il doit accepter que le site puisse accéder à sa position, par mesure de sécurité. Celui-ci se voit donc obligé de cliquer sur "accepter" dans la popup qui apparait quand il arrive sur le site.

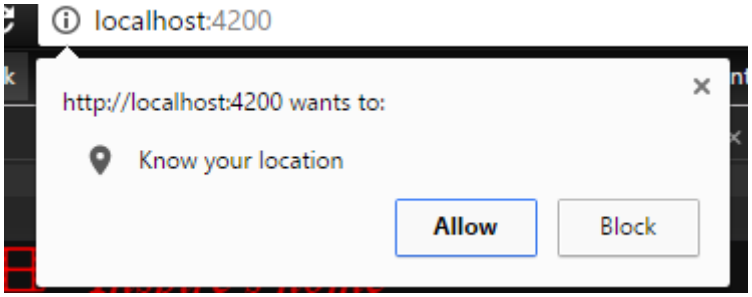


Fig.7: Popup demandant l'autorisation d'utiliser la localisation

Dans le cas où l'utilisateur refuse l'accès, ou si le navigateur n'est pas compatible avec ce type de service, l'utilisateur devra rentrer manuellement sa ville.

Une autre adaptation qui concerne cette fois ci l'utilisateur et ses préférences, en se rendant dans l'onglet *settings*, il peut choisir le type de film qu'il apprécie.

Fig.8: Paramètres de l'application

Par exemple, si l'utilisateur n'aime pas les films d'horreurs, il peut simplement décocher *Horror* et les séances de film d'horreur ne lui seront plus proposées. Cette adaptation à l'utilisateur a été implémenté dans la démo d'Inspire pour montrer l'exemple mais elle n'est pas la seul. L'application finale permettrait par exemple à l'utilisateur de choisir son enseigne de cinéma préféré, la distance maximale qui le sépare du cinéma, la note minimum donnée au film par les autres utilisateurs pour qu'il soit proposé, et bien plus encore… L'application, bien que très simple d'utilisation est flexible et s'adapte à l'utilisateur.

Enfin, comme nous en avons déjà parlé, Angular 2 est component based. Ceci donne la possibilité d'adapter les composants pour les réutiliser dans différentes situations.

Fig.9: Adaptation du composant header

Comme nous le voyons sur la figure 9, le composant header est ici utilisé à la fois dans la vue *home* et dans la vue *détail.* Cette adaptation, qui nécessite au composant header d'avoir un paramètre avec le texte à afficher, permet de ne pas réécrire un code similaire à celui qui existe déjà.

Les limites d'Angular 2

Les deux principales limites auxquelles j'ai dû faire face au cours du développement sont les suivantes :

Premièrement, Angular 2 est encore récent, il n'est pas rare d'expérimenter des bugs ou des fonctionnalités qui ne fonctionnent pas comme elles le devraient. Par exemple les animations de page sont encore en mode expérimental et je n'ai pas réussi à les exploiter convenablement. Toujours dû à la jeunesse de cette version et au nombre important de changement entre les releases opérées par Google, la documentation n'est que partiellement à jour.

La documentation officielle tout comme les aides que l'on peut trouver sur des sites tiers correspondent souvent à des anciens versions d'Angular 2. De ce faite les tutorial ou explications ne fonctionne plus sur la plus récente des versions et il est parfois difficile de trouver les informations nécessaires à l'utilisation des fonctionnalités d'Angular.

Ensuite, une limite qui est partagé avec tout autre web application est le faite que nous n'avons pas accès aux interactions natives (aussi bien sous desktop que mobile). Par exemple, une des fonctionnalités d'Inspire est d'envoyer une notification pour rappeler à l'utilisateur de noter un film qu'il a été voir. Cette fonctionnalité est impossible à l'aide d'Angular car il n'y a aucun moyen d'envoyer une notification à l'utilisateur s'il n'utilise pas le site.

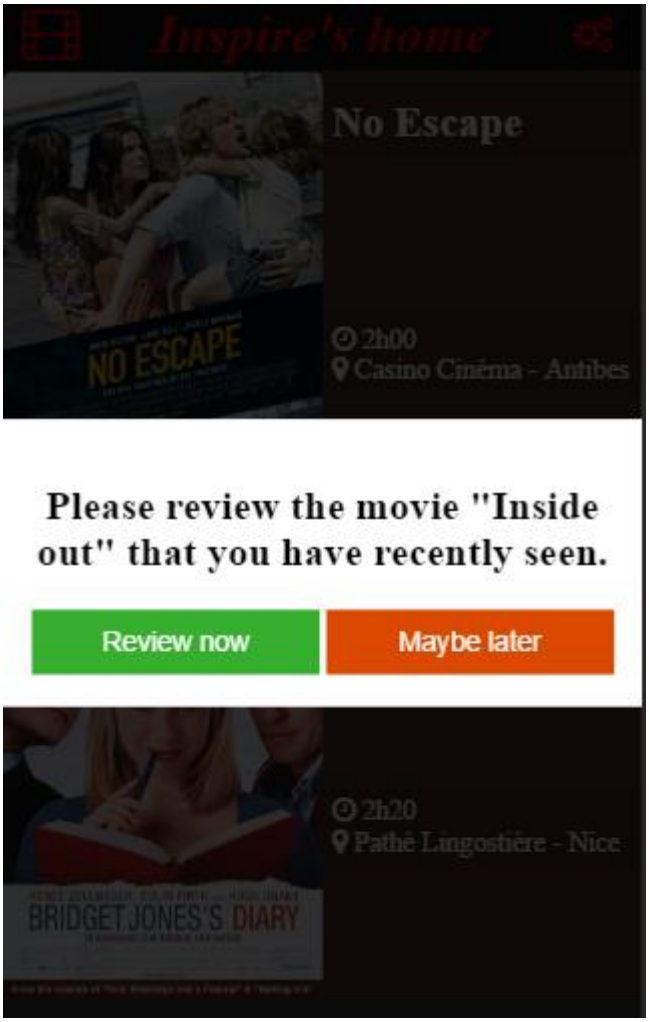


Fig.10: Alert sur la page *home*

Comme on le voit sur la figure 10, les notifications ont de ce fait été remplacées par des alertes in-app, qui sont moins efficaces pour inciter l'utilisateur à laisser son avis sur un film car il ne verra cette alerte que s'il utilise l'application.

Bootstrap

Bootstrap est un Framework Html/JS/CSS facile à prendre en main et très pratique quand il s'agit de rendre son design responsif. Il s'appuie sur un système de grille à 12 colonnes et de classes permettant de les manipuler.

Cette partie va vous expliquer pas à pas comment installer et utiliser BootStrap, ainsi qu'un exemple de déploiement et d'exécution d'un projet.

Installation

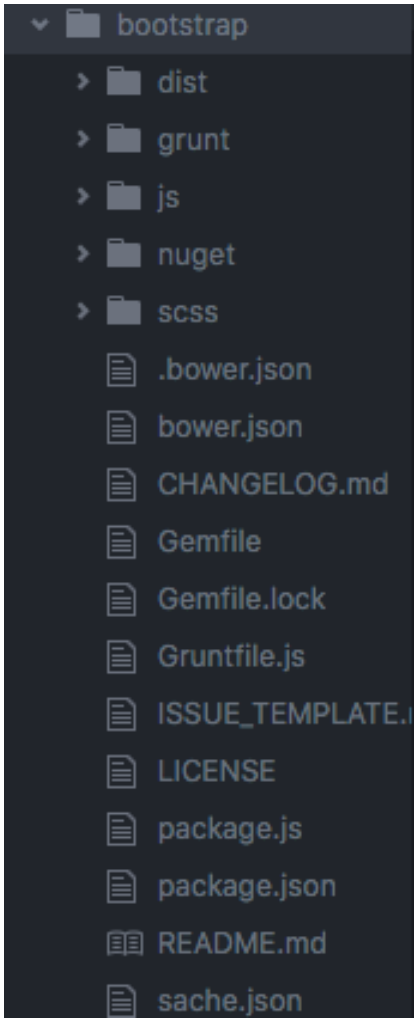
Générale

Tout d'abord, voici un lien pour découvrir Bootstrap : http://getbootstrap.com/getting-started/

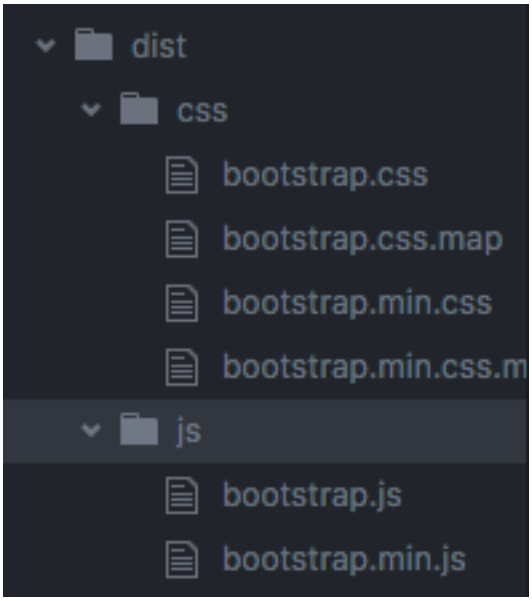
Il y a plusieurs façons d'installer BootStrap :

- En utilisant Bower avec une commande : bower install BootStrap
- En utilisant NPM compris dans node.js avec une commande : npm install bootstrap@3
- En utilisant Composer avec une commande : composer require twbs/bootstrap

Suite à cela vous obtiendrez un dossier bootstrap contenant :



Ce qui nous intéresse se trouve dans le dossier “dist” et “js” :



Ce sont les fichiers JS et CSS qui nous permettront d’utiliser Bootstrap dans notre application.

Mes technologies

J’ai décidé personnellement d’utiliser différentes technologies en plus de BootStrap :

- Cocktail
- Sass et Jade pour la partie HTML et CSS
- Jquery et MediaElement pour la partie JavaScript

La plupart de ces technologies ont déjà été expliqué dans la partie “Electron”, je ne reviendrai donc pas dessus.

Jquery et MediaElement sont tous deux des librairies javascript. Jquery facilite l’écriture du javascript et MediaElement facilite l’utilisation des différents types de média.

Comme j’utilise Cocktail, il me suffit de me placer à la racine de mon projet et d’utiliser la commande

‘gulp --watch’ pour builder et exécuter mon application. Cela ouvrira automatiquement une page internet avec votre page.

```
MBP-de-titanium:~ titanium$ cd Desktop/
MBP-de-titanium:Desktop titanium$ cd Adaptation\ des\ interfaces/
MBP-de-titanium:Adaptation des interfaces titanium$ cd Inspire/
MBP-de-titanium:Inspire titanium$ cd bootstrap/
MBP-de-titanium:bootstrap titanium$ gulp --watch
```

Réalisation de l'application et explication de l'adaptation

Maintenant, passons à la partie réalisation de l'application et de l'adaptation.

J'ai pour cela créé un fichier jade, un fichier Sass et un fichier JS.

Fichier Jade

Voici mon fichier Jade :

```
block content
  .btn
  - each movie in [{title:"Gladiator",trailer:"https://www.youtube.com/watch?v=Q-b7B8tOAQU&html5Mode(true)"},
    .Text_Visibility.Trailer(id=movie.title.replace(' ',"")+'_video')
      video(width='100%', height='100%', autoplay='false',id=movie.title.replace(' ',"")+'trailer')
        source(src=movie.trailer, controls="True",type="video/youtube")

    .col-xs-12.col-sm-6.col-md-4.col-lg-3.movie(id=movie.title.replace(' ',""))
      .img-responsive
      img(src="resources/images/"+movie.title+".jpg", alt=movie.title, title=movie.title)
      p= movie.title

    .Text_Visibility.Text(id=movie.title.replace(' ',"")+'_resume')
      p= "Resume de "+movie.title
      p="Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore e
      p= "Seance"
      .Seance
        - for (var i = 0; i < 4; ++i)

            table
              tr
                td
                  p=" 54 rue de lorem "

                td
                  p=" 4 km "
                td
                  p=" 13 h "
```

Essayons de le comprendre ensemble :

La boucle

Tout d'abord la boucle va nous permettre de générer pour chaque film les informations que nous souhaitons avoir : le poster, la vidéo, un résumé et les séances.

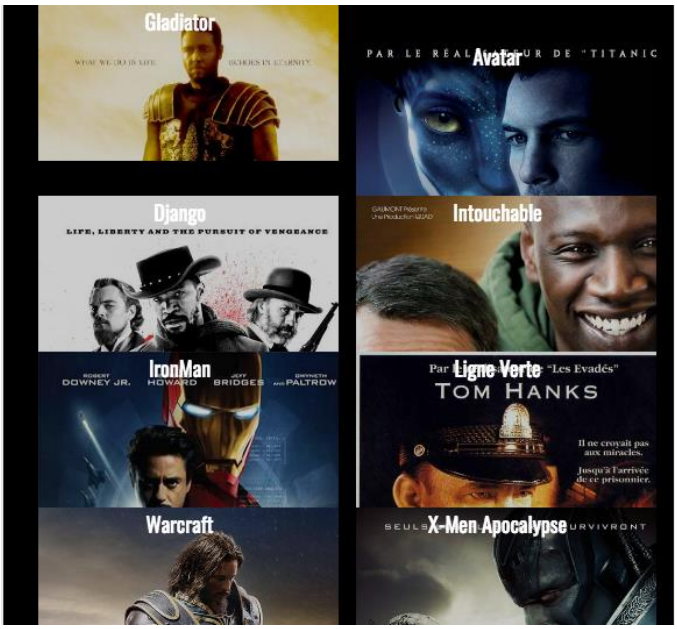
Les classes

Les classes vont avoir deux utilités : rendre notre site plus jolie et responsive et ensuite de réaliser différentes animations grâce au javascript.

ADAPTATION

Particularité ici : vous pouvez noter les classes 'col-xs','col-sm','col-md' et 'col-lg'. Ce sont toutes des classes de BootStrap qui nous permettront de replacer les éléments en fonction de la taille de l'écran : il s'agit de la partie responsive. Le chiffre à coté nous renseigne sur la disposition

des différentes div en fonction de la taille : par exemple ‘col-xs-12’ signifie que chaque div occupera les 12 colonnes de la grille lorsque l’écran est de taille ‘xs’.



Pareillement, ‘col-sm-6” signifie que chaque div prendra 6 colonnes, il y en aura donc deux côtes à côtes.

On peut ainsi s’adapter à la taille de l’écran et afficher une interface plus plaisante en fonction de celle-ci.

Limite : malheureusement, BootStrap ne gère pas encore toute les tailles d’écrans. Bien que BootStrap 4 a inclut une classe pour gérer les devices de taille phablette (grand smartphone), il n’y a pas d’échelon dès que l’on dépasse 1200 px. Il n’y a donc pas d’adaptation possible pour l’instant pour les très grandes tailles d’écran (bureau tactile, retina, 4K).

Autre classe intéressante venant de BootStrap : ‘img-responsive’ qui comme son nom l’indique, intègre au mieux une image afin que celle-ci soit responsive par rapport au device.

Fichier Sass

Ce fichier est le fichier de style qui permet la mise en page de mon application :

```
@media (max-width: 768px) {
  .movie {
    height: 25%;

  }
  .movie_all_screen{
    height: 50%!important;
      width: 100%;
      p{
        position: relative;
        display: none;
        z-index: 1;
      }
  }
  .movie_no_screen{
    display: none;
  }
}
@media  (min-width: 768px) and (max-width: 992px) {
  .movie {
    height: 50%;
  }
  .movie_all_screen{
    height: 50%!important;
    width: 100%;
    p{
      position: relative;
      display: none;
      z-index: 1;
```


Hormis le style normal, que je ne vais pas détailler, j'aimerai attirer votre attention sur les media query '@media' qui proviennent de Bootstrap. A quoi ça sert ? cela permet de modifier les propriétés d'une classe en fonction de la taille de l'écran, en donnant un intervalle de taille pour l'écran.

Par exemple, ici, pour tout écran de largeur inférieure à 778 px , les posters ont une largeur égal à 25% de la largeur total du container. En revanche, pour une largeur d'écran entre 768 pixel et 992 pixel, les posters ont une largeur égale à 50% de la largeur global.

Couplé avec les différentes classes vue au-dessus, il est donc possible de modifier les propriétés de chaque classe en fonction de la hauteur et de la largeur de l'écran.

Fichier JS

J'ai actuellement deux fonctions dans mon js.

```
$(document).ready(function(){
  setInterval(function() {
    $(".btn").after(
      '<div role="alert" class="alert alert-success">'+
      '<button type="button" data-dismiss="alert" aria-label="Close" class="close"><span aria-hidden="'
      '<h4 class="alert-heading">Hey Buddy!</h4>'+
      '<p>Aww yeah, 50 % off on your next ticket !</p>'+
    '</div>')

  }, 10000);
});

//on click event on each poster
$(".movie").click(function(){
  //create a new media for the video
  var player = new MediaElementPlayer('#'+$(this).attr("id")+'trailer');
  //add or remove the class for the transition
  $(this).toggleClass("transform");
  // this class is use to know if i have the video on the screen or not
  $(this).toggleClass("minus");
  if($(this).hasClass("minus")){
     player.play();
  }else{
    player.pause();
  }
  //when i click on a poster , hide the other movie or on contraty show them
  $("#"+$(this).attr("id")+'_video').toggleClass("Text_Visibility");
  $('#' + $(this).attr("id") + '_resume').toggleClass("Text_Visibility");
  $(".movie").not($(this)).toggleClass("transform");
```

Une pour réaliser une petite adaptation de notification (lié à la classe alerte de BootStrap), une autre pour ajouter des classes ou les retirer en fonction d'un événement utilisateur.

Ici, en fonction du poster sur lequel on click, je vais cacher les autres posters, afficher les informations liées au poster sur lequel on a cliqué et créer un Player pour jouer la vidéo du trailer.

La classe 'minus' permet de jouer ou mettre en pause la vidéo en fonction de si la personne veut retourner à la liste de film ou au contraire afficher les informations.

La fonction au-dessus est jouée au chargement de la page et ensuite jouée toute les 10 secondes pour afficher une notification (une alerte Bootstrap).

On pourrait imaginer un système de notification pour les séances de cinéma, en ajoutant de la donnée dans une div d'alerte Bootstrap.

Tester le côté responsive

Pour tester le côté responsive de l'application, vous pouvez utiliser un outil de développement sur le navigateur (F12) et allez dans l'onglet 'device'. Vous pourrez ainsi utiliser différentes résolutions pour visualiser le côté responsive de l'application.

Conclusion

Comme nous l'avons vu au cours du développement de ce rapport chaque technologie apporte un claire avantage. Angular 2 permet d'avoir une application web component based et réactive. Une grande communauté utilise le framework ce qui permet de trouver facilement des tutoriels et de l'aide. La seul difficulté avec l'utilisation d'Angular 2 est le nombre important de mise à jour. Des mise à jour qui souvent changent les méthodes d'interactions avec le framework et rendent les tutoriels et la documentation outdated jusqu'à leur mise à jour. Les deux technologies cross-platform Ionic et Electron sont elles orientées sur la praticité pour le développeur. Bien que les performances soit légèrement dégradées par l'apport de la surcouche la possibilité d'avoir une application disponible nativement sur plusieurs platforms est clairement un avantage. Enfin, bootstrap permet de simplifier la mise en place du design de l'application et même s'il ne gère pas parfaitement toutes les tailles d'écran il permet d'accélérer le développement.

La réalisation de l'application nous a donc permis de découvrire les limites de chacune de nos technologies. Par combinaison de nos expériences nous avons pu en tirer la conclusion que chaque technologie répond à un besoin particulier. Utiliser une technologie cross platform tel qu'Ionic 2 ou Electron permet en effet d'avoir accès aux fonctionnalités native du système d'exploitation. Cependant cet avantage vient avec l'inconvénient de la légère perte en performanc. Dans le cas ou l'application n'a pas l'utilité des fonctionnalités natif, il est donc inutile d'exploiter ces technologies et il est préférable de se cantonner à une application web. Bootstrap de son côté peut être utilisé pour faciliter le développement de l'interface et peut être utilisé avec toutes les autres technologies étudier lors de ce projet.

Pour résumer, l'utilisation d'une technologie cross platform est a conseillé, uniquement si l'utilité des fonctionnalités native se fait ressentir. De leur côté Bootstrap et Angular 2 peuvent être conseillés pour tout projet même si leur manque de maturité reste un problème.