

Développement d'application mobile

Rapport

M2 MIAGE 2020/2021

Arthur Debar - Alexis Petit

Introduction	3
Fonctionnalités	3
Page d'accueil	3
Page de recherche	3
Page Top	4
Page FilmDetail	4
Choix d'architecture	5
Arborescence	5
Framework CSS : Bulma	5
API utilisée : IMDB	5
Plugins Capacitor	6
Share	6
Splashscreen	6
Conclusion	9

Introduction

Dans le cadre du module de développement d'application mobile nous devons réaliser une application en utilisant le framework VueJS. De plus, cette application devait utiliser Capacitor un framework open-source permettant la création d'applications Web Native à partir d'une webapp. Pour réaliser cela, nous avons choisi de créer l'application Movue qui fournit des informations cinématographiques en se basant sur l'API REST d'[IMDb](https://www.imdb.com/).

Le dépôt git : <https://github.com/Alexis-Petit/movue>

1. Fonctionnalités

1.1. Page d'accueil

Cette page liste les films étant actuellement au cinéma (s'il n'y en a qu'un ce n'est pas un bug mais le contexte actuel qui fait que...) ainsi que les films qui vont sortir prochainement. Pour récupérer ces résultats nous avons créé une méthode `init()` appelant les méthodes `movieInTheaters()` et `movieComingSoon()` elle-même appelant l'api avec les deux urls suivantes :

- <https://imdb-api.com/en/API/InTheaters/>
- <https://imdb-api.com/en/API/ComingSoon/>

Les données sont ensuite stockées dans deux tableaux distincts.

La méthode `showCardFilmDetails(cardFilm)` nous permet d'accéder aux composants `CardFilmDetails` avec l'id du film passé en paramètre.

Concernant les mécaniques demandées pour le projet vous retrouverez notamment dans la page d'accueil des rendus de listes avec `v-for` pour afficher la liste de tous les films au cinéma et qui vont sortir prochainement.

1.2. Page de recherche

La page de recherche nous retourne tous les films que l'API trouve pour titre passé en paramètre.

Concernant les mécaniques demandées pour le projet vous retrouverez notamment dans la page recherche le binding d'attributs `v-bind` pour contrôler les changements d'états du bouton de recherche :

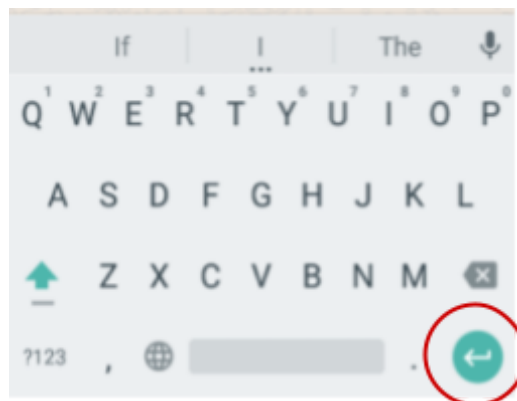
- état désactivé si la requête est vide ;
- état activé lorsque la requête est non-vide ;
- état loading lorsque la requête est lancée.

Nous utilisons également du rendering conditionnel avec une combinaison de v-if et de v-else-if pour contrôler l'affichage du résultat de la recherche :

- v-if : si le résultat contient au moins un film
- v-else-if : si le résultat est vide.

Il n'y a pas de v-else car par défaut, si ni les conditions du v-if, ni celles du v-else-if ne sont remplies alors on ne veut rien afficher.

Dans notre première implémentation, nous nous sommes rendus compte que même après avoir saisi une requête dans la barre de recherche, l'appui sur la touche "Entrée" du clavier ne déclenchait pas la recherche. Il en était de même sur le clavier Android.



Pour obtenir le comportement attendu, nous avons dû rajouter un binding d'évènement sur notre input spécialement pour la touche entrée. Cet évènement fait alors appel à la même fonction que le bouton de recherche.

```
<input @keyup.enter="onSearchClick()"...
```

1.3. Page Top

La page top recense les top 10 des films et séries en tendances ainsi que les films les plus populaires.

1.4. Page FilmDetail

Cette page appelle uniquement le composant *CardFilmDetail.vue* en lui passant en props l'id du film sélectionné.

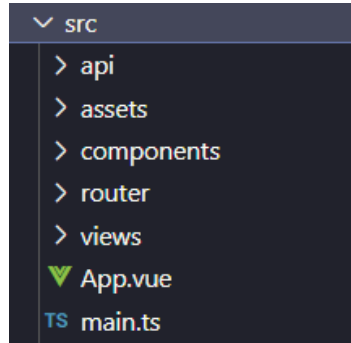
Concernant les mécaniques demandées pour le projet vous retrouverez notamment dans *CardFilmDetail.vue* un binding de style avec notamment :

```
img {  
  cursor: pointer;  
}
```

permettant d'afficher la main de la souris au passage sur une image pour notifier à l'utilisateur que l'image est cliquable.

2. Choix d'architecture

2.1. Arborescence



- Le dossier *api* contient la classe *movie.ts* nous permettant d'interagir avec l'API d'IMDb:
- Le dossier *assets* contient les images utilisées par la webapp et la feuille CSS globale du projet
- Le dossier *components* contient les différents composants utilisés au sein de notre application tel que le menu du header, les cartes contenant les films, celles plus détaillées et celles moins détaillées.
- Le dossier *router* contient la classe *index.ts* contient le routeur de notre application.
- Le dossier *views* contient les différentes pages de notre application tel que *home.vue*, *search.vue*, *top.vue* et *filmDetail.vue*.

2.2. Framework CSS : Bulma

Nous avons choisi de choisir le framework Bulma pour ce projet car nous l'avons vu en cours et nous souhaitons perdre le moins de temps possible à mettre en place le style.

2.3. API utilisée : IMDB

IMDB offre un abonnement gratuit à son API avec une formule de 100 requêtes par jour autorisé. D'autres API proposent des offres gratuites mais la simplicité et le large choix de requêtes disponibles nous a conduit à choisir celle-ci.

Les appels effectués à l'API se trouvent dans le fichier *movie.ts*, celui-ci exporte les méthodes utilisées au sein de l'application et utilise les variables d'environnement définies dans le fichier *.env*. Ce dernier liste les urls utilisées dans l'application ainsi que la clé d'api (qui devrait normalement être passé dans la chaîne d'intégration afin de ne pas la voir en clair dans le code).

Dans notre application nous avons donc fait appel aux requêtes suivantes :

- <https://imdb-api.com/en/API/InTheaters/>
- <https://imdb-api.com/en/API/ComingSoon/>

- <https://imdb-api.com/API/YouTubeTrailer/>
- <https://imdb-api.com/fr/API/Title/>
- <https://imdb-api.com/en/API/SearchTitle/>
- <https://imdb-api.com/en/API/MostPopularMovies/>
- <https://imdb-api.com/en/API/MostPopularTVs/>
- <https://imdb-api.com/en/API/Top250Movies/>
- <https://imdb-api.com/en/API/SearchTitle/>

2.4. Plugins Capacitor

2.4.1. Share

Le plugin Share est implémenté dans la page détaillée d'un film. Au clique nous pouvons partager la page de notre site via messenger, signal etc... Ceci étant, pour que le partage marche parfaitement il faudrait que notre site soit mis en ligne ce qui n'est pas le cas.

2.4.2. Splashscreen

Créer les images d'icônes et de splashscreen peut être fait à la main mais c'est laborieux. Nous avons utilisé la librairie capacitor-resources pour générer automatiquement les différentes icônes et splashscreens correspondant aux différentes tailles d'écrans requises par Android.

Voici les étapes de son utilisation :

- créer une image d'icône d'application : PNG mesurant 1024x1024 pixels nommé icon.png ;
- créer une image de splashscreen : PNG mesurant 2732x2732 pixels nommé splashscreen.png ;
- placer les deux images dans un dossier /resources à la racine du projet ;
- lancer les commandes suivantes :
 - \$ npm i -g capacitor-resources → installation globale du package
 - \$ capacitor-resources -p android → génération des ressources spécifiques à la plateforme Android, et transfert automatique de ces ressources dans le projet.

La version actuelle du plugin splashscreen capacitor comporte un problème majeur : lors de l'affichage du splashscreen, l'image subit deux mises à l'échelle verticales successives. Ceci rend l'affichage du splashscreen peu attrayant. Ce problème est connu (<https://github.com/ionic-team/capacitor/issues/1627>) et les manières de le résoudre ne sont pas idéales. Pour palier à ce problème, nous avons choisi de passer le premier affichage du splashscreen afin d'arriver directement sur l'image mise à l'échelle une seconde fois.

Les deux tableaux suivants illustrent d'abord le problème rencontré, puis la solution que nous avons appliquée. Les images d'illustration sont tirées du issue GitHub en lien plus haut.

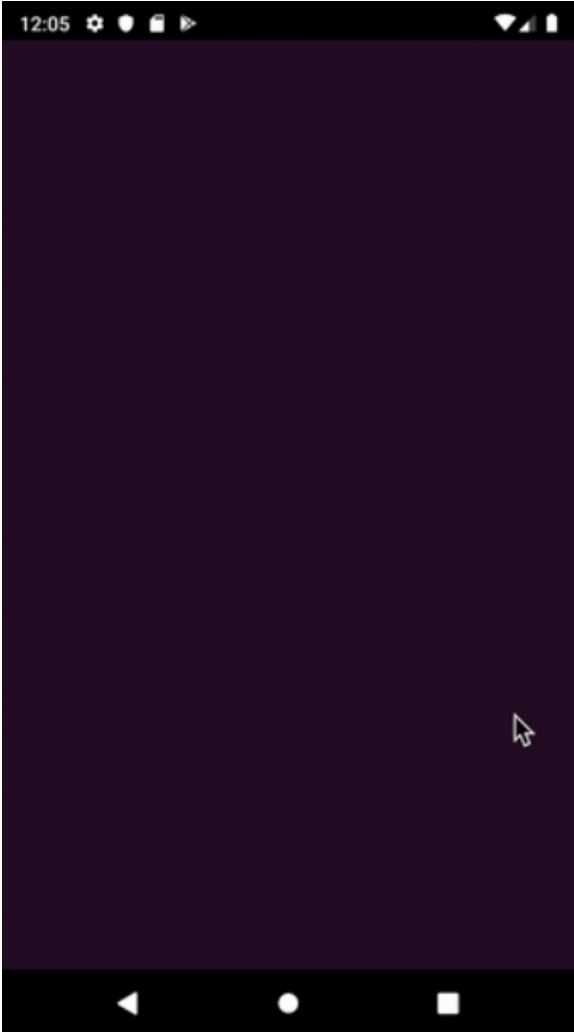
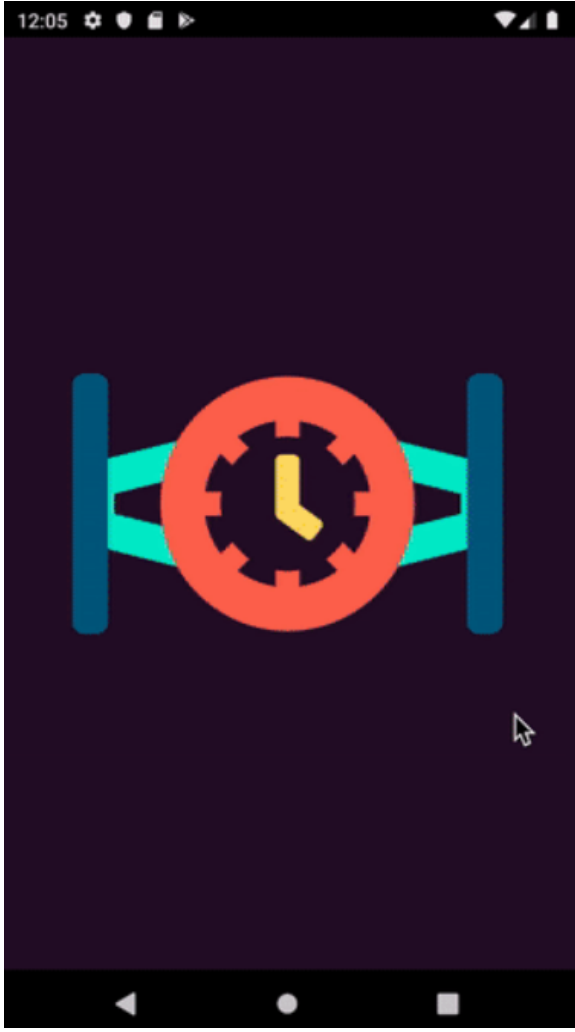
Illustration du problème de scaling



Affichage immédiat au lancement de l'application : l'image de splashscreen est adaptée verticalement pour prendre en compte la barre de notification en noir en haut de l'écran



Après 1 seconde : l'image est étendue verticalement pour venir passer en transparence sous la barre de notification

Illustration du fix	
	
Affichage immédiat : on affiche la couleur du fond de notre splashscreen, mais pas le splashscreen en tant que tel.	Après 1 seconde : un fondu fait apparaître de façon fluide le logo mis correctement à l'échelle.

Pour mettre en place ce fix, il suffit de faire la modification suivante dans le fichier `movue\android\app\src\main\res\values\styles.xml` :

<pre><style name="AppTheme.NoActionBarLaunch" parent="AppTheme.NoActionBar"> <item name="android:background">@drawable/splash</item> </style></pre>	<pre><style name="AppTheme.NoActionBarLaunch" parent="AppTheme.NoActionBar"> <item name="android:background">#5d00ff</item> </style></pre>
Ce style est généré automatiquement par le plugin splashscreen, la valeur par défaut est l'image de notre splashscreen.	On remplace la référence à l'image du splashscreen par # la couleur de fond de notre splashscreen : #5d00ff.

Conclusion

Ce projet nous a permis de mettre en application les concepts vus pendant le cours de développement web avec le framework Vue 3. Ainsi, nous avons pu utiliser les fonctionnalités de base qui permettent de rendre une application Vue réactive.

Il aurait été intéressant d'aller plus loin dans notre implémentation des concepts avancés de Vue, tels que l'utilisation d'un store avec Vuex, ou encore intégrer réellement du typage fort avec Typescript. Nous n'avons pas non plus pris le temps d'écrire des tests avec un outil comme Jest par exemple.

Utiliser Capacitor était une bonne ouverture sur les possibilités ouvertes par les librairies permettant le développement d'applications mobiles hybrides. Le choix de plugins offerts par Capacitor est vaste et cette technologie fait maintenant partie des solutions que nous pourrions envisager en cas de besoin de développement natif.