

## Projet final : Movie hub

---

Le but de ce projet est de créer une page affichant une liste de film récupérée via une API REST, on aura également une barre de recherche qui permet de filtrer la liste de films, enfin, lorsqu'on cliquera sur un film. On affichera les détails de celui-ci dans un pop-up.

### Préparation du projet

Comme pour les projets précédents, vous allez cloner un projet **vue-cli** de puis un repository git. Ce repository contient déjà un projet **vue-cli** initialisé, ainsi que l'architecture de base du projet GIT

Url du dépôt : <https://github.com/SaillyMaxence/movies-app>

Une fois le projet cloné. Ouvrez le dans vscode et installez les dépendances via :

**npm install**

Cette commande va installer les dépendances nécessaires pour le projet dans le dossier **node\_modules**.

Pour lancer le projet en local, utilisez la commande

**npm run serve**

### Architecture du projet

En plus des dossiers habituels d'un projet Vue CLI: package.json, public, babel-config.js, (...), on trouve aussi dans le dossier src/components des templates de fichiers .vue qui serviront de base pour les futures parties du TP.

#### **main.js**

Il contient déjà l'instanciation de notre composant App.vue ainsi que les paramètres par défaut de l'application. Inutile de le modifier.

#### **HomeView.vue**

Il s'agit du composant primaire de notre application, celui qui servira de composant principal. Il contient là où nous allons construire notre movie-hub

#### **config.json**

Il s'agit d'un fichier de configuration. Dans ce fichier, il y a plusieurs url

- movie\_list : permet d'afficher les films
- movie\_detail : permet d'avoir les détails d'un film
- photo\_path : permet d'afficher les affiches de films

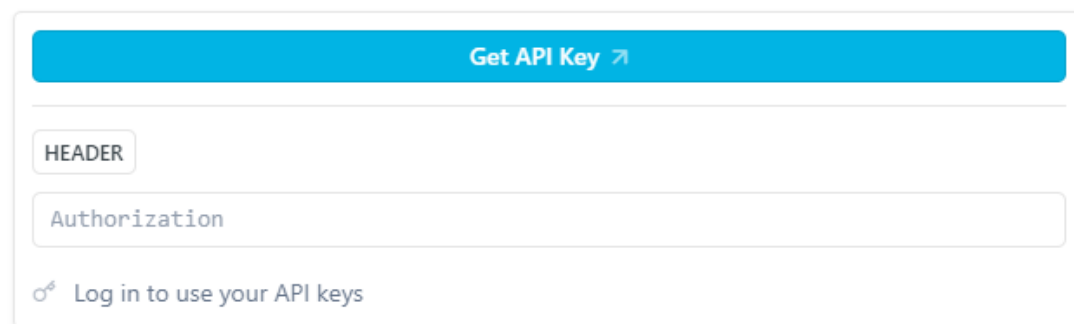
Enfin `api_key` : c'est là où vous allez mettre votre clé d'api pour pouvoir fetch les données. **Attention de ne pas push cette clé dans le rendu final.**

Pour cela vous allez devoir vous rendre sur :

<https://developer.themoviedb.org/reference/intro/getting-started>

Il va falloir que vous vous inscriviez et que vous récupériez votre api key via :

## 2 Authenticate



Get API Key ↗

HEADER

Authorization

🔗 Log in to use your API keys

### Framework CSS

Le projet a été initialisé avec le framework vuetify, celui-ci permet d'avoir un layout responsive comme bootstrap. Je vous conseille de l'utiliser, il y aura des références à la documentation sur ce projet. (<https://vuetifyjs.com/en/>)

Maintenant que nous avons vu la globalité du projet. Nous allons pouvoir coder.

### Les composants

- **MovieList**: le but de ce composant sera d'afficher la liste des films
- **MovieDetail** : son objectif est d'afficher les détails d'un film sélectionné via un pop-up
- **movieSearch** : il s'agit d'une barre de recherche qui permettra de filtrer la liste des films en fonction de la recherche.
- **HomeView** : c'est le composant qui va recevoir les 3 composants du dessus
- **MoviePagination** : Pour les plus rapides. Ce composant va permettre de créer une pagination et de changer de page d'api

### Le composant HomeView

Nos différents composants vont être appelés au même endroit et communiqueront entre eux. Nous pourrions directement les mettre dans `App.vue` mais il est de convention de n'appeler qu'un composant dans l'`App.vue`. De plus structurellement parlant il est plus logique de créer un composant définissant un movie-center et toutes ses fonctionnalités dans le cas où il faudrait le faire évoluer...

Dans ce composant, je vous invite à importer les futurs composants que nous allons créer. Pour le moment vous pouvez importer les composants présents dans le dossier **components**

## Le composant moovieList

Ce composant sert à afficher une liste de film. C'est par lui que nous allons commencer à travailler.

Pour commencer à le tester n'oubliez pas de l'intégrer dans votre fichier moovieList.vue afin qu'il s'affiche.

### Partie script

Dans le cas du script, à la création du composant nous allons utiliser la bibliothèque axios afin de faire une requête vers l'URL de l'API. Vous devriez sûrement vous aider d'un hook de cycle de vie.

Pour récupérer la liste des films il faudra utiliser l'url **movie\_list** du fichier config (n'hésitez pas à l'importer) , N'oubliez pas de mettre la votre clé d'api, sinon vous ne pourrez pas récupérer les données

```
axios.get('https://api.github.com/user', {
  headers: {
    'Authorization': `token ${access_token}`
  }
})
.then((res) => {
  console.log(res.data)
})
.catch((error) => {
  console.error(error)
})
```

(Exemple d'appel api avec headers)

N'hésitez pas à tester l'API depuis Postman afin de voir ce qu'elle renvoie ou via un console.log().

Dans cette réponse il y a la liste des films que nous stockerons dans une propriété de data du nom de votre choix.

. C'est cette propriété que nous utiliserons ensuite dans le **<template>**.

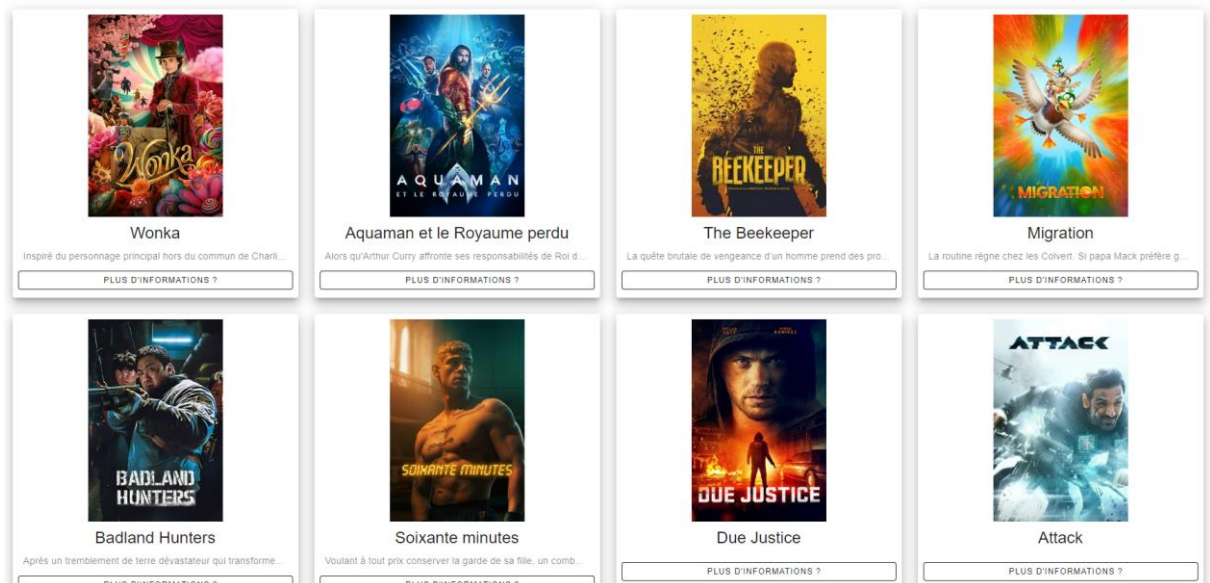
Pour générer les images dans le <template> il va nous falloir aussi récupérer photo\_path dans le fichier config et la stocker également dans une propriété data du nom de votre choix

Pour cela n'hésitez pas à importer le fichier config.json dans votre composant.

### Partie template

Ici, je vous laisse libre intégration, le but est d'afficher la liste de films que vous avez stockés dans votre propriété data.

Vous pouvez-vous rapprocher de vuetify et regarder les composants **v-row**, **v-col**, **v-card**



(Exemple d'intégration)

## Le composant movieDetail

Ce composant sert à afficher le détail d'un film lorsque l'on cliquera sur celui-ci. Cela sous-entend qu'il va y avoir un lien entre les deux composants.

Pour commencer à tester ce composant. N'oubliez pas d'intégrer celui-ci dans HomeView

### Partie <script>

Le composant movieDetail nécessitera une props **movieId** qui recevra l'id du film sélectionné dans la page movieList. Ainsi lorsque l'on appellera notre div on pourra l'adapter au film passé dans la props.

A la **création** du composant nous allons utiliser la bibliothèque **axios** afin de faire une requête vers l'URL **movie\_detail** en concaténant l'id du film passé en props. Ce résultat sera stocké dans une propriété data du nom de votre choix.

Pour générer les images dans le **<template>** il va nous falloir aussi récupérer **photo\_path** et la stocker dans une propriété du data.

Nous aurons une **méthode closeDetail()** qui enverra un **événement** au composant parent afin d'indiquer qu'il faut fermer/cacher le composant **<MovieDetail>**.


### Partie <Template>

Vous êtes assez libre sur la manière de construire l'HTML de la carte d'identité du Film. Vous trouverez ci-dessous une liste non exhaustive de ce que vous pourriez afficher. A vous de voir quelles informations vous cherchez à afficher.

- Image du film
- Nom
- Notes
- Production
- Description

N'oubliez pas d'ajouter un bouton à votre pop-up pour appeler la fonction `closeDetail()` définie dans le `<script>`.

Wonka







Comédie Familial Fantastique

Note :

★ ★ ★ ★ ★ ★ ★ ☆ ☆ ☆

Production :



Domaine d'animation

Résumé :

Inspiré du personnage principal hors du commun de Charlie et la Chocolaterie, le best-seller de Roald Dahl qui est aujourd'hui l'un des livres pour enfants les plus vendus de tous les temps, "Wonka" est une histoire merveilleuse retraçant la jeunesse de Willy Wonka, et comment il est devenu ce grand inventeur, magicien et chocolatier que nous connaissons aujourd'hui. Dans ce spectacle saisissant et d'une créativité sans limites, les spectateurs découvriront un Willy Wonka jeune, la tête débordant d'idées et bien décidé à changer le monde à coups de délicieuses chocolateries. Il vous montrera que les meilleures choses de la vie commencent par un rêve, et qu'en rencontrant Willy Wonka, tout devient possible.

[FERMER](#)

(Exemple d'intégration)

Ci-dessus les composants vuetify utilisé sont : `v-avatar`, `v-img`, `v-chip`, `b-button`, `v-rating`.

## Ajouter un évènement à MovieList

C'est lors d'un clic sur un élément de la liste de film que s'affichera le composant **<MovieDetail>**. La communication partira donc de **MovieList** et remontera jusqu'à **<MovieDetail>** pour cela dans votre fichier **<MovieList>**, il faut que lorsque vous cliquez sur un film, une fonction **sendMovie()** soit déclenchée. Elle prendra en paramètre le film sélectionné.

Cette fonction effectuera un **\$emit** nommé **showMovieDetailEmit** pour communiquer avec son parent.

## Modifier l'appel à movieDetail dans le fichier homeView.vue

Il ne faut pas oublier de rajouter les attributs et événements à notre appel de **<movieDetail>**

La **props movieID** et l'écoute de l'évènement **closeDetail**.

Une condition d'affichage du composant si un élément de la liste a été cliqué ou non. Vous pourriez

Utiliser une data booléenne pour savoir cela.

Maintenant que nous avons ajouté l'émission d'un événement **showMovieDetailEmit** dans **<MovieList>**, il faut aussi l'écouter et appeler une fonction en réponse au niveau de l'appel du composant.

N'oubliez pas que l'évènement envoie les informations d'un film de la liste. Dans ces informations il y a l'**id** du film en question. C'est peut-être cette id que nous aurons besoin de passer en **props** de notre composant **<MovieDetail>**.

A vous de réfléchir comment relier tout cela. Dites vous simplement que la communication de l'info part de **<MovieList>** atteint **<HomeView>** et doit aller jusqu'à **<MovieDetail>** via sa **props**.

## Le composant MovieSearch

Nous allons maintenant créer une barre de recherche qui permettra de filter la liste des films.

### Partie <template> du movieSearch

Il s'agit d'une banale barre de recherche. Le template possèdera donc un champ `<input />` (ou `v-text-field` si vous souhaitez utiliser **vuex**). Vous pouvez y ajouter des attributs que l'on retrouve souvent dans ce cas: un **placeholder** par exemple. Vous aurez besoin de stocker la valeur de ce champ dans une **data**. N'oubliez pas la directive **v-model** !

### Partie <script> du movieSearch

Nous souhaitons que notre page se mette à jour juste en tapant dans le champ de recherche, pas besoin de validation, on changera tout en temps réel.

Pour cela il va falloir que l'on émette un évènement pour indiquer à la liste qu'une recherche est en cours.

D'ordinaire on émet des évènements via la directive `v-on`. Mais si je veux que l'évènement soit envoyé dès que la valeur de l'input change il y a un moyen bien plus simple avec une nouvelle propriété de l'instance vue: **watch**

La propriété **watch** est assez simple à comprendre, vous définissez quelle valeur vous souhaitez surveiller, par exemple une data. Si cette data est modifiée de quelque manière que ce soit (ici via notre v-model de l'input) alors le watcher s'active et exécute le code indiqué dans sa fonction.

Dans l'exemple ci-dessous, dès que la data `rangeVal` est modifiée, alors le code contenu dans la fonction s'exécute et alimente la data `rangeVal`.

```
<input type="range" v-model="rangeVal">
<p>{{ rangeVal }}</p>
```

```
const app = Vue.createApp({
  data() {
    rangeVal: 70
  },
  watch: {
    rangeVal(val){
      if( val>20 && val<60) {
        if(val<40){
          this.rangeVal = 20;
        }
        else {
          this.rangeVal = 60;
        }
      }
    }
  }
})
```

Vous trouverez plus d'informations sur la propriété **watch** dans la documentation de vueJS.

Maintenant que nous avons vu cet exemple l'idée dans notre projet est de surveiller la valeur data liée au champ `<input>` et dès qu'elle est modifiée émettre un événement `searchMovieEmit` auquel on joindra la valeur de notre recherche.

### Modification du composant `homeView`

Notre appel à `<MovieSearch>` devra écouter l'émission de l'évènement `searchMovieEmit` déclaré précédemment. Il déclenchera une méthode `setMovieSearch` qui renseignera une **data** avec la valeur récupérée de l'évènement. Cette data sera ensuite passée dans une nouvelle **props** du `<MovieList>`.

### Modification du `MovieList`

Afin d'affiner notre liste il va falloir passer la recherche récupérée depuis `<MovieSearch>` comme **props** du composant `<MovieList>`.

Dans le template il va maintenant falloir afficher uniquement si le nom du film en question contient la valeur de la recherche. Mais si vous essayez d'implémenter un **v-if** au même niveau qu'un **v-for** vous risquez de vous heurter à de gros problèmes. On pourrait créer des div supplémentaires mais il y a un moyen bien plus simple : et si nous faisons que la condition se passe dans une propriété **computed** et qu'ensuite cette propriété **computed** soit utilisée par le **v-for** ?

Définissez une propriété **computed** `movieFiltered` qui représentera le tableau de la liste des films mais filtrés. Cette propriété vérifiera si la valeur de recherche est vide ou non.

Si elle est vide alors il retournera le tableau du data qui stockait jusqu'ici notre liste de film. Ce qui reviendra au comportement que l'on a depuis le départ.

Si la valeur de recherche est renseignée nous allons utiliser le package `lodash` qui permet de faire des opérations complexes sur des tableaux. La fonction de `lodash` que nous utiliserons sera la fonction `filter`.

Nous préciserons comme tableau de départ la liste de nos films non modifiés récupérée

depuis l'API. Et nous ferons le filtrage de ce tableau sur la propriété **title**. La fonction `filter` renverra un nouveau tableau contenant uniquement les éléments qui match avec notre recherche. C'est le résultat de cette fonction **filter** que nous retournerons dans notre propriété **computed**

Vous trouverez plus d'informations sur le package **lodash** et ses fonctions dans la documentation.

Du côté du `<template>` dans notre boucle nous utiliserons désormais notre propriété **computed** au lieu de  
notre data.