arinfo
École des Métiers du Numérique
GROUPE I-MAGINER

Titre professionnel Développeur Web & Web Mobile

Le framework Vue JS

formateur : Hugo VIROT

- **Vue JS est un des frameworks front-end JavaScript les plus populaires. Il est également léger et simple à apprendre. A vous de jouer !**

Objectifs

- Créer une application front-end grâce à VueJS
- Utiliser des données provenant d'une API
- Continuer à progresser en JavaScript

Durée

- 6 jours

Présentation

Vue JS a été créé en 2013 par le chinois Evan You. Développeur travaillant chez Google sur des projets en Angular (framework JS nettement plus complexe), il décide de reprendre les fonctionnalités d'Angular qui lui plaisent, tout en les incluant dans un nouveau framework bien plus léger et facile à maîtriser. C'est ainsi qu'a été créé VueJS, qui est depuis devenu de plus en plus populaire au fil des années.

Vue JS (prononcez "view") est comme son nom l'indique centré sur la vue restituée à l'utilisateur. Il est principalement dédié à la création d'interfaces utilisateur et de **SPA (Single Page Applications** : applis web/mobile tenant sur une seule page).

C'est un framework exclusivement front-end : on ne peut pas gérer de base de données avec Vue (comme avec Laravel par exemple). Cependant, il permet de **gérer des données externes provenant d'une API (Application Programming Interface,** ou Interface de programmation).

On peut aussi le coupler à un framework back-end JS (ou utilisant une autre technologie) : en effet, **un projet Vue n'est pas "monobloc", c'est un ensemble constitué d'un ou plusieurs composants indépendants.** Il est ainsi possible d'**ajouter des composants Vue sur un site web existant.** On peut ainsi très bien imaginer un site fonctionnant avec Vue pour le front-end et Laravel pour le back-end.

La dernière version de Vue JS en date est **Vue JS 3 (sorti en septembre 2020).** C'est celle-ci que nous allons étudier.

Fonctionnement

A) Le modèle MVVM

Le framework PHP Laravel, que nous avons étudié ensemble, est basé sur le modèle MVC : Model-Vue-Controller. **Vue, quant à lui, fonctionne sur le modèle MVVM (Model-Vue-Vue-Model).** Le principe est légèrement différent. Voyons son fonctionnement.

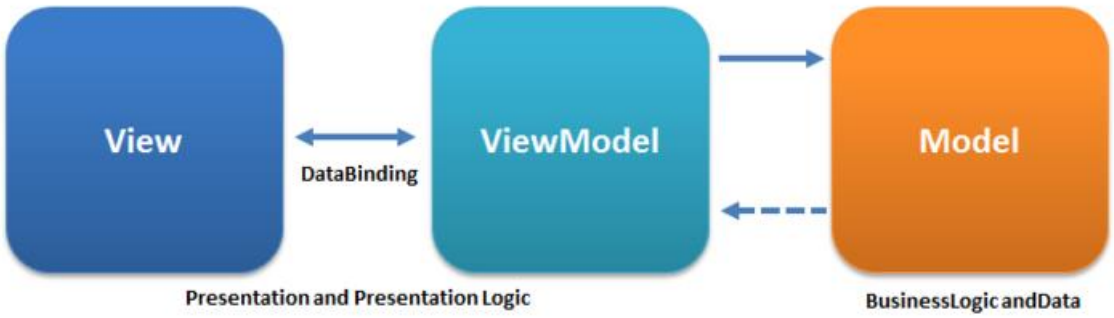


Schéma de fonctionnement du modèle MVVM (source : Wikipédia)

- **Le Model récupère des données via des appels API.** Son rôle est ainsi similaire au Modèle de Laravel (la base de données en moins).

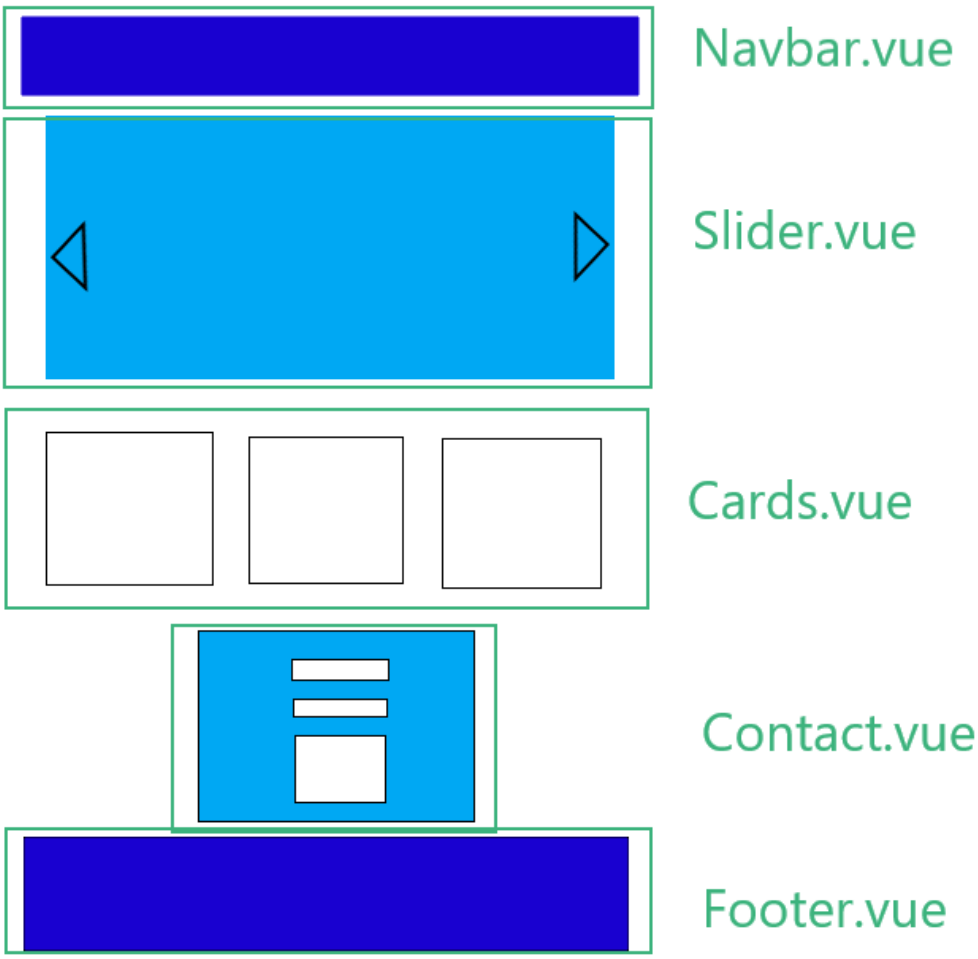
- Ici, pas de Controller : **c'est le ViewModel qui joue ce rôle. Il utilise les données du Model pour les envoyer dans la View** (rendu visuel d'un composant, situé entre les balises <template></template>), qui les affiche. **Le ViewModel fait donc l'intermédiaire entre le Model et la View.**

- **Les données sont liées entre la View et le ViewModel : on parle de Data Binding. Chaque modification des données dans la View se répercute dans le ViewModel.**

- **Avec Vue, les 3 parties Model, ViewModel et View se retrouvent souvent dans le même fichier** (contrairement à Laravel, qui sépare strictement les composantes du modèle MVC en plusieurs fichiers distincts : les modèles, les vues et les contrôleurs)**.**

B) Les composants

Comme de nombreux frameworks JavaScript, Vue JS utilise des **composants** pour composer (justement !) les pages qu'il affiche. **On ne conçoit plus le site sous forme de pages entières comme en HTML/CSS ou en PHP, mais sous forme de composants réutilisables, qui affichent chacun une petite partie de la page**



Exemple de découpage d'une page basique en composants

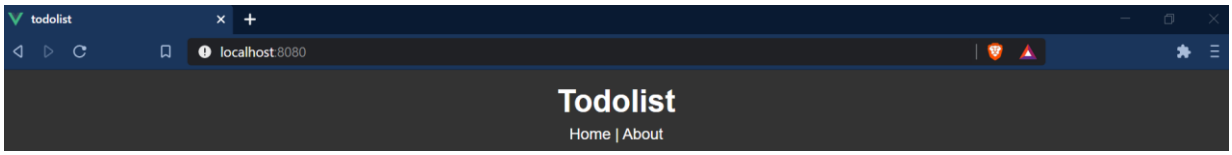
Comme on peut le voir sur le schéma, les composants sont des fichiers en .vue (extension des fichiers Vue JS).

```vue
<template>
  <header class="header">
    <h1>TodoList</h1>
    <div id="nav">
      <router-link to="/">Home</router-link> |
      <router-link to="/about">About</router-link>
    </div>
  </header>
</template>

<script>
export default {
  name: 'Header'
}
</script>

<style scoped>
  .header {
    background: #333;
    color: #fff;
    text-align: center;
    padding: 10px;
  }
  .header a {
    color: #fff;
    padding-right: 5px;
    text-decoration: none;
  }
</style>
```

Exemple de composant : un header très simple avec son propre style : **Header.vue**



Rendu dans le navigateur (nécessite également un fichier App.vue pour cela)

(source : https://medium.com/weekly-webtips/introduction-vue-js-with-a-simple-project-665bad37e0b)

Voici le schéma détaillé **du cycle de vie d'un composant** (source : doc officielle). **Vue JS le créé lorsqu'il a besoin d'être affiché, et le supprime lorsqu'il n'est plus nécessaire** (exemple : changement d'affichage).

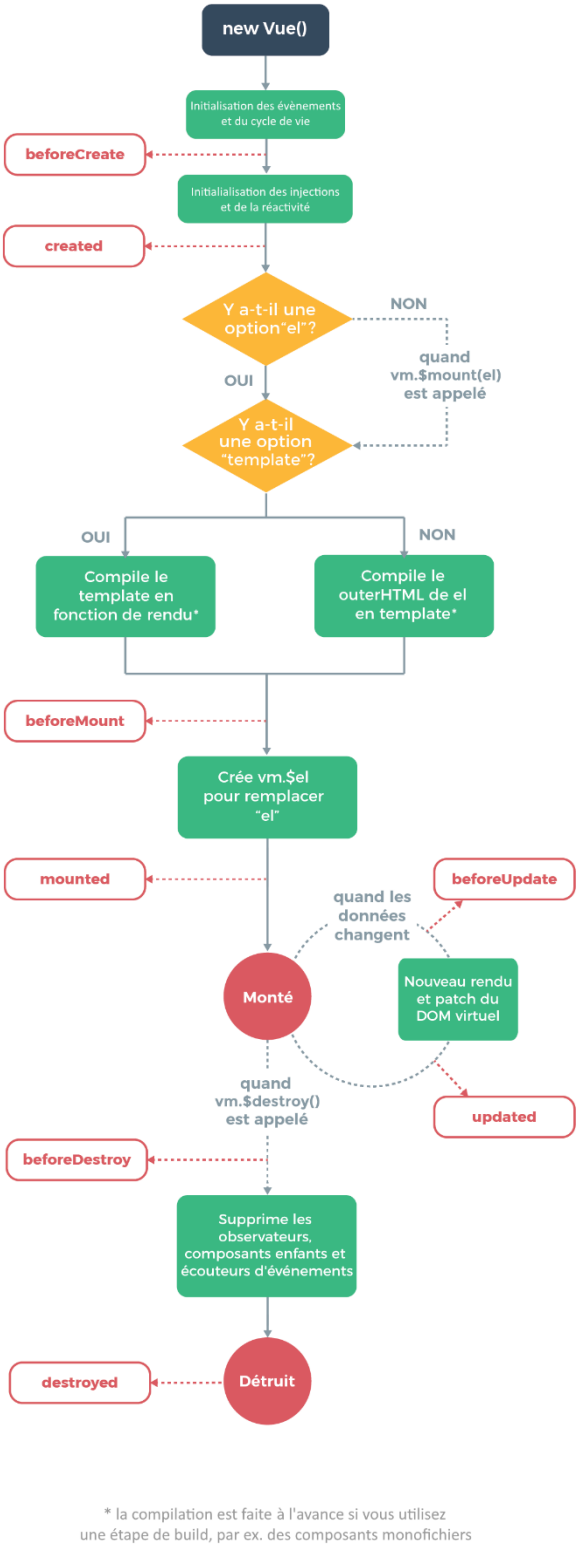


Schéma du cycle de vie d'un composant Vue JS

(source : https://fr.vuejs.org/v2/guide/instance.html#Hooks-de-cycle-de-vie-d%E2%80%99une-instance)

On peut déclencher des actions à certains moments précis du cycle de vie, en utilisant des fonctions portant les noms des étapes écrites en rouge sur le schéma : *created*, *mounted*, etc.

Pour être utilisé, un composant doit être importé, référencé, puis instancié dans les balises template d'un autre composant (voir ci-dessous).

```
1  <template>
2  <div>
3    <h1 class="pt-5 font-weight-light">Films français</h1>
4    <SortButtons :movies="movies" @sort-movies="sortMovies"/>
5    <MoviesList :movies="movies" :loading="loading" :errored="errored" />
6  </div>
7  </template>
8
9  <script>
10
11 import MoviesList from "./utils/MoviesList.vue";
12 import axios from "axios";
13 import SortButtons from "./utils/SortButtons.vue";
14
15 export default {
16   name: "FrenchMovies",
17   components: {
18     MoviesList,
19     SortButtons
20   },
21   data() {
22     return {
23       movies: [],
```

Exemple d'instanciation de deux composants dans un autre

Ici, nous sommes dans le composant "*FrenchMovies.vue*", qui affiche des films français (application type Allociné). **Il fait appel à deux composants dans sa Vue** (partie template) : ***SortButtons*** (boutons de tri) et ***MoviesList*** (Affichage de la liste des films via une boucle). Les deux composants sont d'abord :

- **Importés** (lignes 11 et 13)
- **Référencés dans la propriété components** (lignes 17 à 19) du composant *FrenchMovies*

Ensuite, ils peuvent être instanciés dans la partie template, à chaque fois via une balise portant le nom du composant (lignes 4 et 5).

C) Data et props : la gestion des données dans un composant

Pour gérer des données dans un composant, **2 cas de figure** sont possibles :

1. **Le composant récupère lui-même ses données (le plus souvent via un appel API)**

Dans cette situation, **une propriété data(), déclarée dans la partie "export default" du composant, va contenir une ou plusieurs variables**. **Le résultat de l'appel API** (il peut y avoir plusieurs appels) **va être stocké dans une** (ou plusieurs) **variable**(s) **déclarée**(s) **dans data**.

```
data() {
  return {
    movies: [],
    loading: true,
    errored: false,
  };
},
```

Data du composant *FrenchMovies*

Ici, nous sommes dans le composant *FrenchMovies*. **Les films récupérés via l'appel API seront stockés dans la variable *movies***. Le chargement, *loading*, est défini par défaut à true (il sera défini à false quand l'appel API est terminé). Errored sera défini à true en cas d'échec de l'appel API.

2. **Les données sont passées depuis le composant parent**

Dans ce cas, on déclare - toujours dans la partie **"export default"**, une donnée appelée **props** (abréviation de **properties**, **propriétés** en anglais). **Il s'agit d'un**

tableau qui contient toutes les variables nécessaires au composant. Elles vont lui être passées par le composant parent, qui va l'instancier.

```vue
<script>
export default {
  name: "MovieCard",
  props: [
    "id",
    "index",
    "title",
    "release_date",
    "vote_average",
    "overview",
    "poster_path",
  ],
```

1) Props du composant *MovieCard*

```vue
<ul>
  <li class="list-unstyled" v-for="movie in movies" :key="movie.id">
    <MovieCard
      :id="movie.id"
      :title="movie.title"
      :poster_path="movie.poster_path"
      :release_date="movie.release_date"
      :vote_average="movie.vote_average"
      :overview="movie.overview"
    />
  </li>
</ul>
```

2) Instanciation de *MovieCard* dans le composant parent *MoviesList*

Ici, on a deux composants :

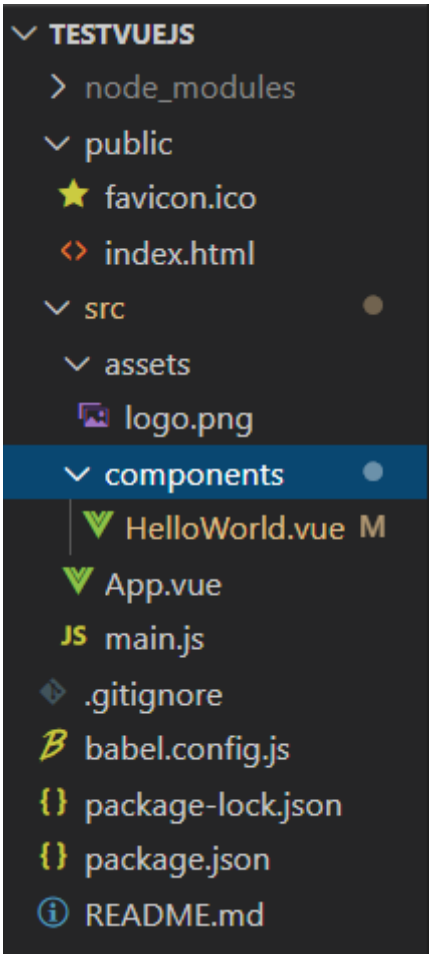
- **MoviesList** : **boucle sur une liste de films** / ici : composant **parent**
- **MovieCard** : **affiche les infos d'un film** dans une "card" bootstrap / ici : composant "**enfant**"

MoviesList va "sous-traiter" l'affichage de chaque film à MovieCard, dont c'est le rôle.

- **Dans sa partie template, MoviesList boucle sur les films, contenus dans la variable *movies*** (qu'il reçoit lui-même en tant que props par son parent).

- **A l'intérieur de cette boucle, il fait appel à MovieCard en lui passant les props nécessaires** (voir capture 1)**. Ces props ont été définies dans MovieCard lui-même** (voir capture 2). Elles sont obtenues par MoviesList en partant de **l'alias** "**movie**", choisi pour accéder dans la boucle aux infos de chaque film.
- MovieCard est ainsi **appelé et affiché autant de fois qu'il y a de films.**

D) Les fichiers et l'arborescence d'un projet Vue JS (template de base)



Arborescence du template de base (tout est affiché sauf node_modules)

- App.vue : situé dans le dossier src, **c'est le composant principal, qui fait appel (successivement ou non) à tous les composants présents dans l'application**. Dans cet exemple, il n'y en a qu'un, appelé "**HelloWorld**".

```vue
<template>
  <div id="app">
    <img alt="Vue logo" src="./assets/logo.png">
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>

<script>
import HelloWorld from './components/HelloWorld.vue'

export default {
  name: 'App',
  components: {
    HelloWorld
  }
}
</script>

<style>
#app {
  font-family: Avenir, Helvetica, Arial, sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
  text-align: center;
  color: #2c3e50;
  margin-top: 60px;
}
</style>

```

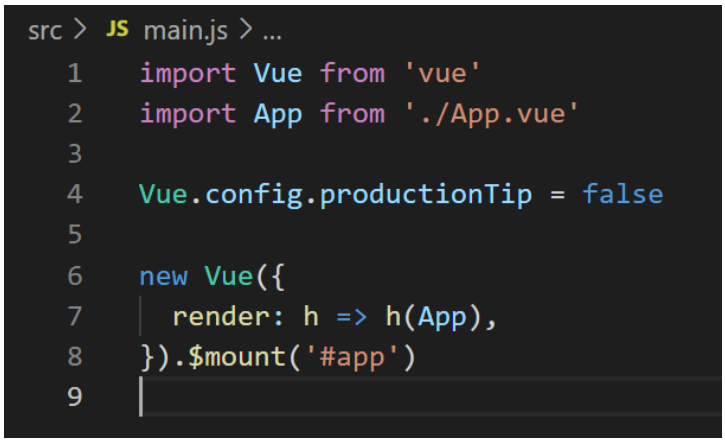
- **index.html** : situé dans le dossier **public**, **c'est la page html de base. Il contient une div qui comporte l'id app : elle correspond à la partie "template" de notre fichier App.vue.**

public > index.html > ...

```html
<!DOCTYPE html>
<html lang="">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <link rel="icon" href="<%= BASE_URL %>favicon.ico">
    <title><%= htmlWebpackPlugin.options.title %></title>
  </head>
  <body>
    <noscript>
      <strong>We're sorry but <%= htmlWebpackPlugin.options.title %> doesn't work properly withou
    </noscript>
    <div id="app"></div>
    <!-- built files will be auto injected -->
  </body>
</html>

```

- **main.js** : **fichier “racine” de l’application. C’est lui qui instancie l’application, et l’injecte dans la div comportant l’id “app” de l’index.html.** Il peut aussi contenir le code du Vue Router, si utilisé.



- **HelloWorld.vue** : les composants se trouvent dans le dossier **src/components**. Celui-ci est très simple : **il affiche simplement le message (msg) défini dans App.vue (l.4), juste après son instanciation.**

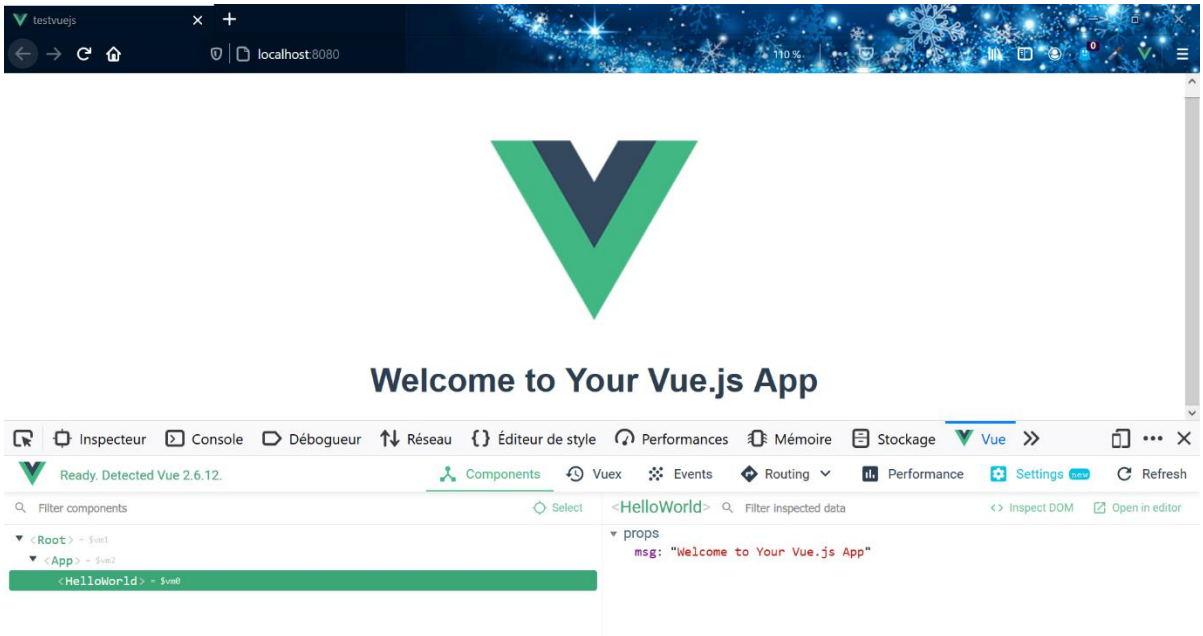
src > components > HelloWorld.vue > {} "HelloWorld.vue"

```vue
<template>
  <div class="hello">
    <h1>{{ msg }}</h1>
  </div>
</template>

<script>
export default {
  name: 'HelloWorld',
  props: {
    msg: String
  }
}
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
h3 {
  margin: 40px 0 0;
}
ul {
  list-style-type: none;
  padding: 0;
}
li {
  display: inline-block;
  margin: 0 10px;
}
a {
  color: #42b983;
}
</style>
```

Voici le résultat de tout cela dans le navigateur :



En inspectant avec les **Devtools VueJS** (extension contenant des outils de développement VueJS, disponible pour Chrome et Firefox), on remarque **une arborescence dans les composants** : on peut ainsi remonter jusqu'à HelloWorld (Root -> App -> HelloWorld), qui contient la variable msg, attribuée via App.vue.

Remarques :

- Les **images** sont stockées dans le dossier **src/assets**
- Comme avec Laravel, un projet VueJS contient également un **package.json** (court) et un **package-lock.json** (détaillé) avec la **liste des dépendances**.

E) Le routage

Pour le **routage**, il vous faudra installer le package npm **vue-router** :

```
npm install vue-router
```

Très utile, **il permet de simuler une navigation avec plusieurs routes**, avec des **"pseudo-URL"** dans la barre d'adresse et des **changements d'affichage** dans le composant App (**même si en pratique, on reste dans l'index.html et dans le composant App**). Cela permet de créer une **SPA** (**Single-Page Application**) proposant plusieurs affichages différents. Voir les liens utiles pour plus d'infos.

F) La syntaxe Vue

Vue utilise une syntaxe particulière, avec des éléments spéciaux intégrés directement au html et appelés directives, tels que :

- **V-for** pour les boucles for

```
<div v-for="item in items">
  {{ item.text }}
</div>
```

- **V-if** et **v-else** pour les conditions

```
<p v-if="seen">Maintenant vous me voyez</p>
```

- **{{ variable }}** pour afficher une variable dans une vue, entre des balises html (la variable doit être déclarée dans l'objet data correspondant)

```
<span>Message: {{ msg }}</span>
```

- **V-bind:variable** ou son raccourci **:variable** (idem mais pour l'intérieur des balises html)

```
<div v-bind:id="dynamicId"></div>
```

(source : doc officielle : https://vuejs.org/guide/essentials/template-syntax.html)

Rendez-vous sur cette page pour en savoir plus !

F) Quelques infos complémentaires

- **Vue ne dispose pas de librairie CSS intégrée** (comme Bootstrap dans Laravel). Vous pouvez utiliser Bootstrap (via CDN ou NPM), il existe aussi une version de Bootstrap créée spécialement Vue JS (avec une syntaxe particulière) : https://bootstrap-vue.org/docs
- Il est extrêmement léger (23ko !)
- On peut l'intégrer très facilement dans un projet existant en ajoutant une des balises script suivantes :

```HTML
!-- Version développement. Celle-ci donne des avertissements utiles sur la console -->
script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
```

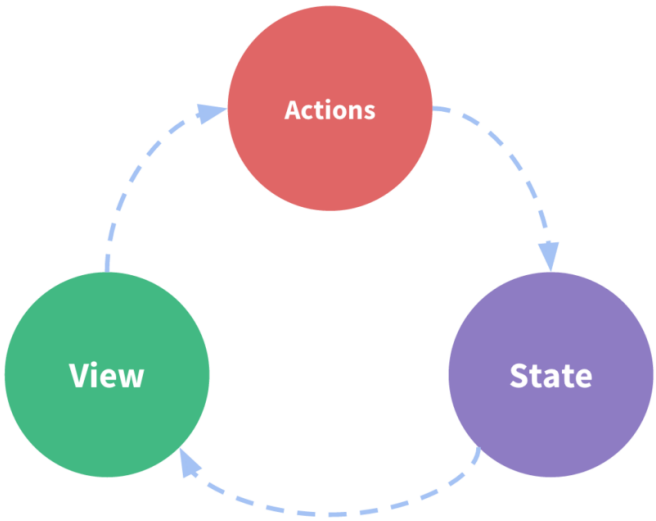
ou :

```HTML
<!-- Version production. celle-ci est optimisée pour la taille et la vitesse -->
<script src="https://cdn.jsdelivr.net/npm/vue"></script>
```

G) La mise en place d'un state global avec VueX

Dans VueJS, comme dans React et ReactNative (avec Redux)**, il est possible de mettre en place un state global. Il s'agit d'un objet qui va stocker des variables accessibles dans tous les composants**. **On l'implémente grâce à la librairie VueX** (il existe aussi une autre librairie similaire, récente, appelée Pinia). Le fonctionnement est le suivant :

- **State** : **contient les variables globales,** qui vont être affichées (et éventuellement modifiées) dans les vues
- **Vue/view** : **c'est la partie front-end : le rendu visuel de votre page**. On y définit le **HTML et le CSS** de la page, et on y **affiche les données issues du state** (elles peuvent être modifiées).
- **Action** : **fonctions présentes dans le fichier de base de VueX (store.js), qui modifient le state global. Elles sont déclenchées par des actions de l'utilisateur** : cliquer sur un bouton, ajouter un élément à une liste, effectuer un tri, etc. **Ces changements vont se répercuter sur la vue.**



Fonctionnement des interactions d'un composant VueJS avec le state de VueX (source : https://www.numendo.com/blog/framework/vue-js-framework-javascript/)

Nous allons utiliser VueX dans le projet Vue JS réalisé. Le but est de conserver les données récupérées par les différents appels API (en l'occurrence des listes de films), afin de ne pas répéter le même appel à chaque chargement d'un

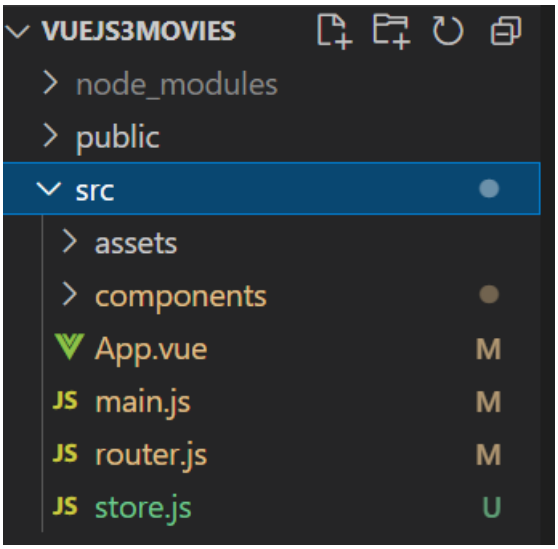
composant. Nous verrons que l'on peut également placer ces données dans le **local storage** (stockage local) afin de les conserver même si l'utilisateur actualise la page ou ferme l'onglet ou le navigateur.

Pour mettre en place Vue X, il faut d'abord **l'installer** :

```
PS C:\wamp64\www\test> npm i vuex

added 1 package, and audited 944 packages in 2s
```

Ensuite, pour l'implémenter, on va **créer un fichier store.js dans le dossier src**.



Ce **store** (« magasin » en anglais) permet **de rendre accessible le state dans les différents composants**, et également **de le modifier**.

Avant de remplir ce fichier, on va tout de suite **installer l'exension VueXPersist,** qui permet de stocker le state dans le local storage :

```
PS C:\wamp64\www\test> npm i vuex-persist

added 2 packages, and audited 946 packages in 2s
```

Le store va s'organiser en plusieurs parties :

1. **Imports** des éléments nécessaires (lignes 2 et 3)
2. **Initialisation** de **Vue X Persist** (lignes 7 à 10)
3. **Déclaration du state par défaut** (lignes 12 à 18)
4. **Début de la création du store** en lui-même avec ses options, d'abord le **choix du state** (lignes 20 à 30)
5. **Ajout des mutateurs** (mutations en anglais) : **fonctions qui vont modifier le state** (lignes 32 à 62)
6. **Ajout des getters** : **fonctions servant à accéder aux variables du state**, comme en programmation objet (POO) (lignes 64 à 84)
7. **Ajout des actions** : **fonctions permettant de déclencher une modification du state global en appelant un mutateur** (on peut aussi appeler les mutateurs directement sans passer par des actions) (lignes 86 à 90)
8. **Plugins** : on précise ici qu'on utilise Vue X Persist (indispensable pour qu'il fonctionne) (ligne 92)

Pour modifier le state depuis un composant, on a deux solutions :

- **Appeler directement un mutateur**
- **Appeler une action, qui va elle-même faire appel à un mutateur**

La différence est la suivante : **appeler directement un mutateur se fait de façon synchrone** (exécution en même temps que le reste du code), alors que **l'action est exécutée de façon asynchrone** (en parallèle de l'exécution du reste du code).

Ainsi, **on va préférer la 2ème solution, l'action, pour deux raisons :**

- **Cela ne ralentit pas la page** (par exemple si la modification du state est liée à un appel API, ce qui est le cas pour nous)
- **Cela fait partie des bonnes pratiques de développement** dans le cadre de l'implémentation d'un state global (pas seulement en Vue JS, mais également en React par exemple). Le code est plus lisible et plus organisé.

Remplissez le fichier store.js comme ceci :

main.js M store.js U

src > store.js > store > mutations

```
// // on importe createStore et on initialise le store
import { createStore } from 'vuex'
import VuexPersist from 'vuex-persist'

// vuex-persist permet de stocker le state dans le local storage
// => en cas d'actualisation de la page, on garde la connexion utilisateur
const vuexPersist = new VuexPersist({
    key: 'vuejs3movies',
    storage: window.localStorage
})

const defaultState = {
    popularMovies: null,
    frenchMovies: null,
    americanMovies: null,
    lastMovies: null,
    top50Movies: null
}

export const store = createStore({
    state() {
        // si un state est stocké dans le localStorage, on l'utilise
        // sinon : state par défaut (avec variables vides)
        if (localStorage.state) {
            return localStorage.state
        }
        else {
            return defaultState
        }
    },

    mutations: {
        // stocker les films les plus populaires dans le state
        storePopularMovies(state, payload) {
            state.popularMovies = payload
        },

```

```
38          // stocker les films français dans le state
39          storeFrenchMovies(state, payload) {
40              state.frenchMovies = payload
41          },
42
43          // stocker les films américains dans le state
44          storeAmericanMovies(state, payload) {
45              state.americanMovies = payload
46          },
47
48          // stocker les derniers films dans le state
49          storeLastMovies(state, payload) {
50              state.lastMovies = payload
51          },
52
53          // stocker les films du top 50 dans le state
54          storeTop50Movies(state, payload) {
55              state.top50Movies = payload
56          },
57
58          // réinitialiser le state
59          resetState(state) {
60              Object.assign(state, defaultState)
61          }
62      },
63
64      getters: {
65          getPopularMovies(state) {
66              return state.popularMovies
67          },
68
69          getFrenchMovies(state) {
70              return state.frenchMovies
71          },
72
73          getAmericanMovies(state) {
74              return state.americanMovies
```

```
75          },
76
77          getLastMovies(state) {
78              return state.lastMovies
79          },
80
81          getTop50Movies(state) {
82              return state.top50Movies
83          },
84      },
85
86      actions: {
87          resetState() {
88              store.commit("resetState")
89          }
90      },
91
92      plugins: [vuexPersist.plugin]
93  })
94
95
```

Ca y est, votre store est en place ! A présent, il vous reste à **l'implémenter et à l'utiliser dans chaque composant qui réalise un appel API** (FrenchMovies.vue, AmericanMovies.vue, App.vue, etc).

Le principe est le suivant (exemple avec le composant FrenchMovies) :

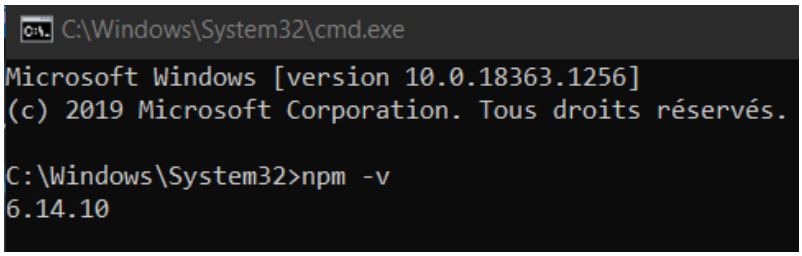
1) **On lance l'appel API** (ici, on récupère les derniers films français)
2) **On stocke les films récupérés dans le state** (variable frenchMovies)
3) **Immédiatement et au prochain chargement du composant : on affiche les films, qui sont maintenant disponibles** => on ne relance pas l'appel API (pas besoin)

Pour réussir cela et pour plus d'informations sur Vue X, consultez la documentation officielle : https://vuex.vuejs.org/

Utilisation

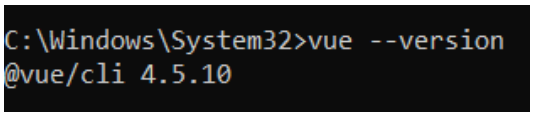
A) Commencer un nouveau projet VueJS

1. **Vérifiez que NodeJS et NPM sont bien installés sur votre pc.** Pour cela, ouvrez un terminal et tapez **npm –v.** Si c'est le cas, le numéro de la version apparaît, comme ici :



Dans le cas contraire, il vous faut les installer. Rendez-vous sur ce site : https://nodejs.org/fr/ . Choisissez la première version disponible ("recommandée pour tous les utilisateurs").

2. **Installez VueCli (= Vue Command Line Interface), la ligne de commande Vue JS.** Elle va nous servir à initialiser un nouveau projet en une seule commande. Tapez la commande suivante : **npm install -g @vue/cli.** Après l'installation, vous pouvez taper **vue –version** pour vérifier que tout est ok. Vous devez obtenir quelque chose comme ceci :



3. **Vous pouvez maintenant lancer un nouveau projet, en tapant vue create monprojet** (uniquement des minuscules). **Plusieurs options sont proposées** : le template par défaut de Vue version 2, celui de la version 3, et une configuration manuelle. **Choisissez la 2ème option (Vue JS 3) et validez.**

```
Vue CLI v4.5.10
? Please pick a preset: (Use arrow keys)
> Default ([Vue 2] babel, eslint)
  Default (Vue 3 Preview) ([Vue 3] babel, eslint)
  Manually select features
```

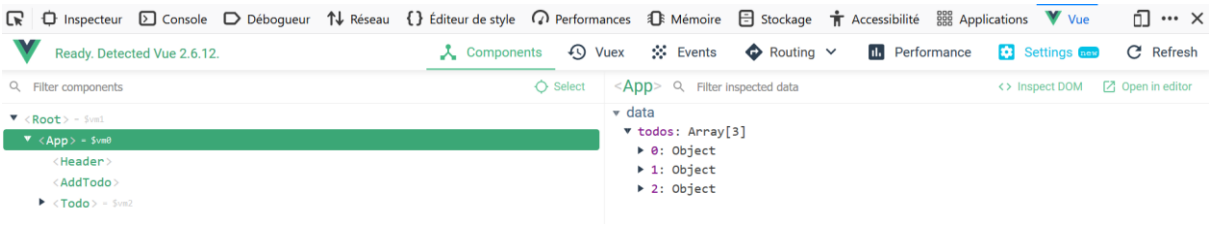
4. **Placez-vous dans le dossier créé** en faisant **cd monprojet.**
5. **Lancez le serveur de développement** avec **npm run serve.**

Votre projet est initialisé !

B) Installer les Devtools (outils de développement)

- Pour Firefox : https://addons.mozilla.org/fr/firefox/addon/vue-js-devtools/
- Pour Chrome : https://chrome.google.com/webstore/detail/vuejs-devtools/nhdogjmejiglipccpnnnanhbledajbpd

Ils seront ensuite accessibles **en ouvrant la console** (“inspecter l’élément” ou F12), **tout à droite : onglet « Vue ».**



Ils permettent de visualiser :

- L’arbre des composants
- Les variables utilisées dans chaque composant
- Les routes
- etc

Pensez à les utiliser, ils sont très pratiques !

Consignes

Avant de commencer les exercices ci-dessous, prenez le temps de parcourir :

- La doc officielle
- Les liens utiles
- D'autres tutos ou vidéos de votre choix

1) Exercice simple : affichage d'une liste de clients

Il est temps d'attaquer votre premier projet VueJS !
Vous allez suivre ce tuto pour afficher une liste de clients.

https://medium.com/quick-code/how-to-create-a-simple-vue-js-app-in-5-minutes-f74fb04adc01

N'hésitez pas à ajouter du style et à personnaliser l'ensemble une fois que tout fonctionne.

2) Application avec données API (5,5 jours)

Vous allez réaliser une application de type "Allociné" affichant des informations sur les films : résumé, date de sortie, note moyenne, etc. **Celle-ci va devoir afficher les données récupérées depuis l'API TheMovieDB** (un site qui se rapproche du célèbre IMDB : Internet Movie DataBase).

1. Pour commencer, **créez un compte sur** https://www.themoviedb.org/

2. Ensuite, **suivez la procédure suivante pour obtenir une clé API** : https://developers.themoviedb.org/3/getting-started/introduction

3. **Créez un nouveau projet Vue JS 3** en ligne de commande avec Vue CLI (voir "utilisation" plus haut)

4. **Créez un dépôt GIT** vierge et **connectez-le** à votre projet (voir cours GIT si besoin). NB **: le projet ne peut pas tourner sur les Github pages, ce n'est plus du statique** (il faut un serveur pour l'héberger).

Les contraintes sont les suivantes :

- **Le projet devra intégrer Vue X qui va permettre de mettre en place un store pour stocker les films récupérés par les différents appels API (voir pages 16 à 21)**

- **Le projet devra utiliser Vue Router : installation à faire, config et routes à mettre en place dans main.js, et liens dans la navbar (qui sera dans le Header)**

- **L'affichage de base (Header + films avec le plus de votes + Footer) sera mis en place dans le composant principal App.vue, entre les balises template évidemment. Seul l'affichage du centre de cette partie template de App.vue va changer, via Vue Router, grâce au code suivant (à implémenter dans votre projet) :**

App.vue

src > App.vue > { } "App.vue"

```vue
<template>
  <div id="app">
    <HeaderNav />

    <div v-if="$route.path == '/'">
      <!-- si la route est / (racine du site) -->
      <h1 class="pt-5 font-weight-light">
        Vos films préférés sont sur VueJS Movies !
      </h1>
      <MoviesList :movies="movies" :loading="loading" :errored="errored" />
    </div>

    <div v-else>
      <!-- si la route est différente de / -->
      <router-view :key="$route.fullPath"></router-view>
    </div>

    <FooterApp />
  </div>
</template>
```

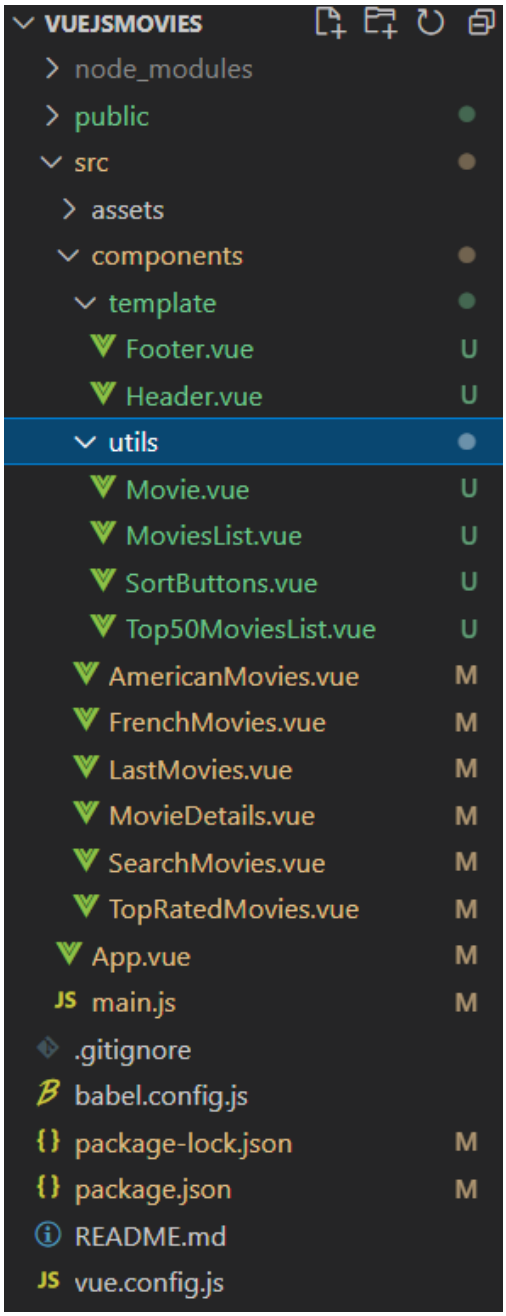
Exemple :

- Je suis sur l'accueil, **je clique sur "films français"** dans la navbar
- **L'affichage change : j'ai maintenant les films français dans la partie centrale de la page** (le header et le footer n'ont pas bougé)
- **L'adresse dans la navbar a également changé** : je suis passé de la racine du site (" **/** ") à "**/FrenchMovies**"

On simule ainsi une navigation avec plusieurs pages, mais tout se passe dans l'index.html et dans le composant App.vue. Il est très important de comprendre cela.

- **Affichages ou "pages" requises** :
  - Accueil (films les plus populaires au niveau du nombre de votes)
  - Films sortis en 2022
  - Films français
  - Films américains
  - Les 50 films les mieux notés
  - Rechercher un film

- **Arborescence attendue** :



Dans le dossier **components** :

- Un dossier **template** contenant le **Header** et le **Footer**
- Un dossier **utils** avec les **composants dits "utilitaires"** (appelés par les composants "principaux") :
  - **Movie** (affiche un film dans une "card" bootstrap, appelé par MoviesList)
  - **MoviesList** (affiche la liste des films via une boucle)
  - **SortButtons** (boutons de tri par note / date / titre)
  - **Top50MoviesList :** affiche la liste des films du top50 (similaire à MoviesList avec des éléments en plus)
- A la **racine**, les **composants "principaux" correspondant aux différents affichages**:
  - **AmericanMovies** : films américains du moment
  - **FrenchMovies** : films français du moment
  - **LastMovies** : films de l'année en cours uniquement
  - **MovieDetails** : détails d'un film (au moins titre, langue, slogan, description et bande-annonce)
  - **SearchMovies** : rechercher un film en saisissant une ou plusieurs lettres dans un input
  - **TopRatedMovies** : Top 50 des films les mieux notés
- Possibilité de **rechercher un film** (via le composant **SearchMovies**)

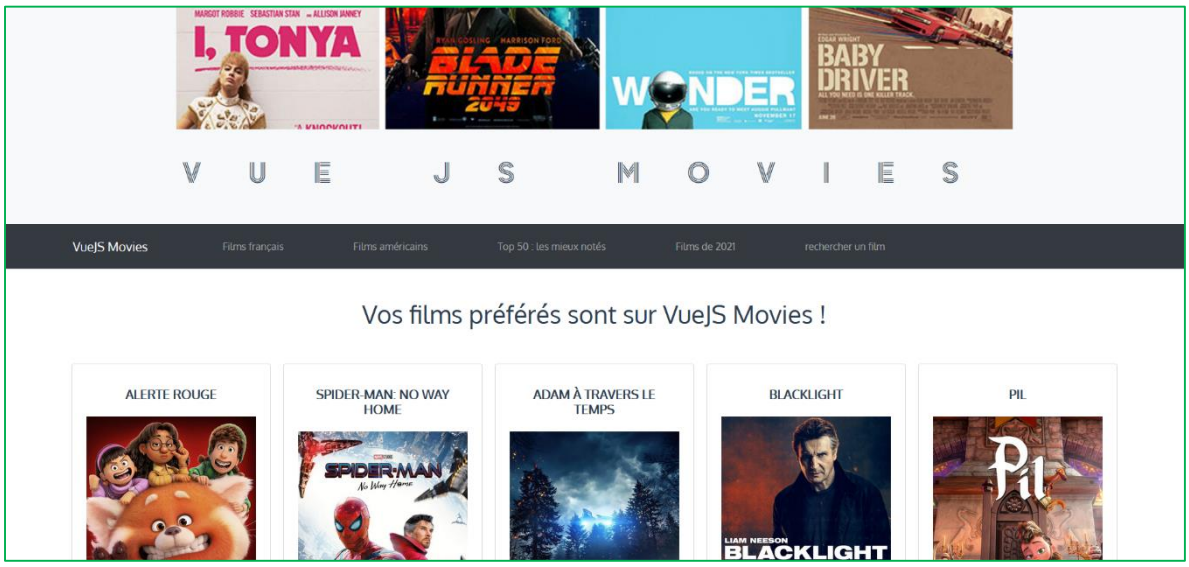
- Possibilité de **trier les films** par titre, date de sortie et note, sur chacune des pages (via **SortButtons**)

Le projet est à rendre pour le mercredi 6 juillet à 13h30 (envoyer au formateur le lien du dépôt Github).

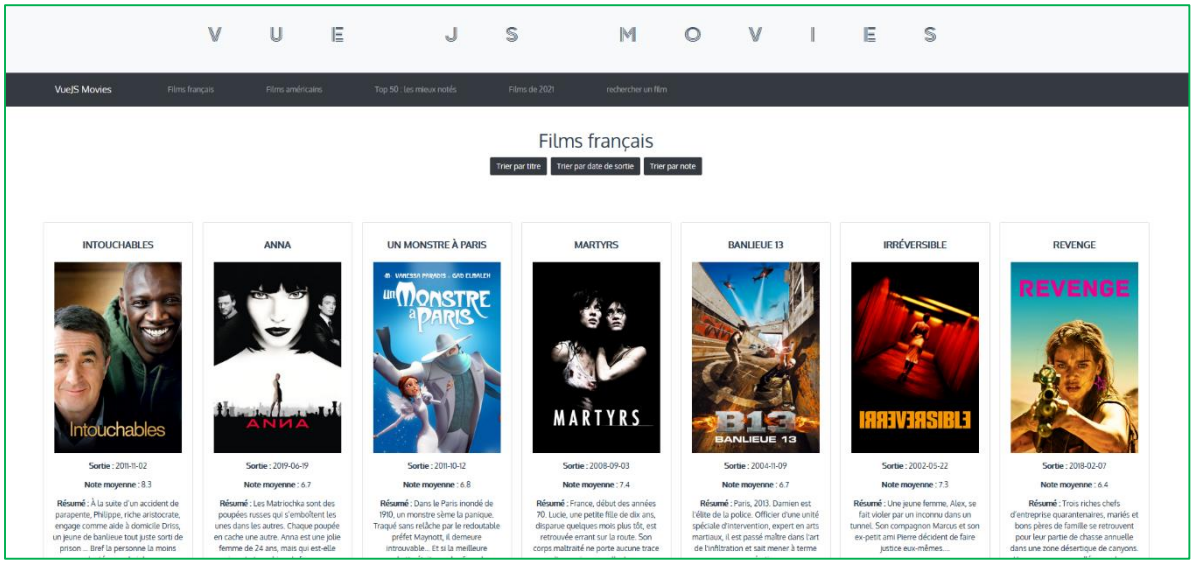
Pensez à faire un commit à chaque avancée !

Exemple de résultat (captures parfois zoomées pour être plus lisibles, le header est le même sur toutes les pages) :

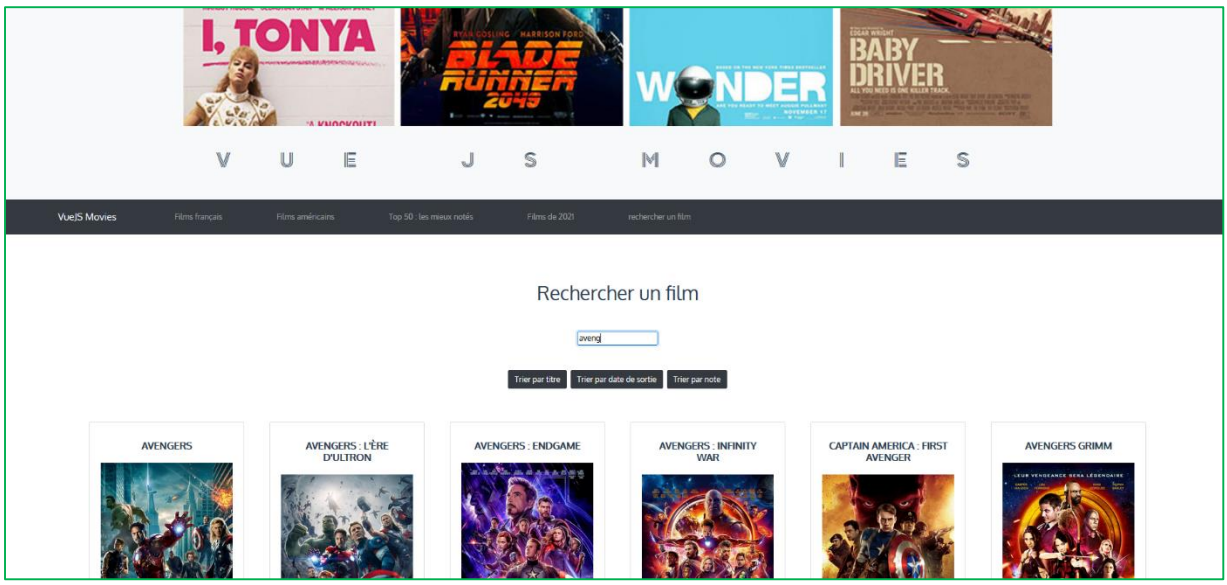
Accueil



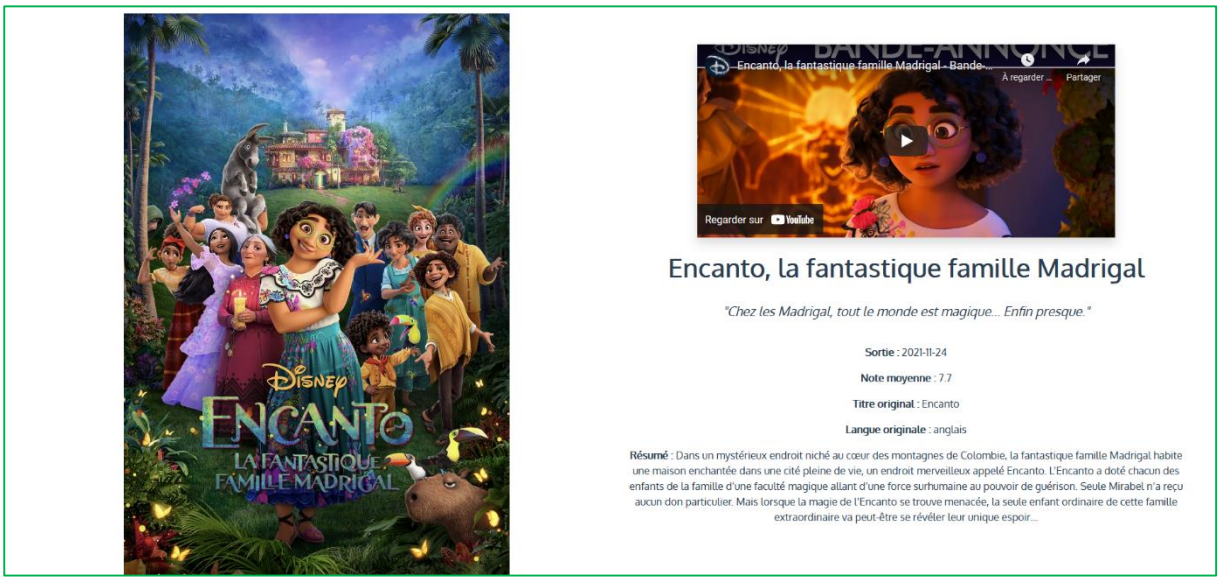
Films français



Rechercher un film



Détails d'un film



Liens utiles

Description	Adresse
Doc officielle	https://vuejs.org/guide/introduction.html
Installer Vue CLI (ligne de commande Vue)	https://cli.vuejs.org/guide/installation.html
Le modèle MVVM	https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel
Tuto : les bases (blog “jesuisundev”)	https://www.jesuisundev.com/comprendre-vuejs-en-5-minutes/
Présentation générale et exemples	https://www.numendo.com/blog/framework/vue-js-framework-javascript/
Créer des routes avec VueRouter	https://router.vuejs.org/
Autre site sur VueRouter	https://flaviocopes.com/vue-router/
Tour d’horizon de Vue 3	https://madewithvuejs.com/blog/vue-3-roundup

Bonus (facultatif)

Différences Vue 2 / Vue 3	https://medium.com/javascript-in-plain-english/differences-between-vue-2-and-vue-3-ee627e2c83a8