

Mission 3 : Ma filmothèque**Equipe:**

- Rajinie Vingadassamy
- Sébastien Texier
- Pires Baptiste

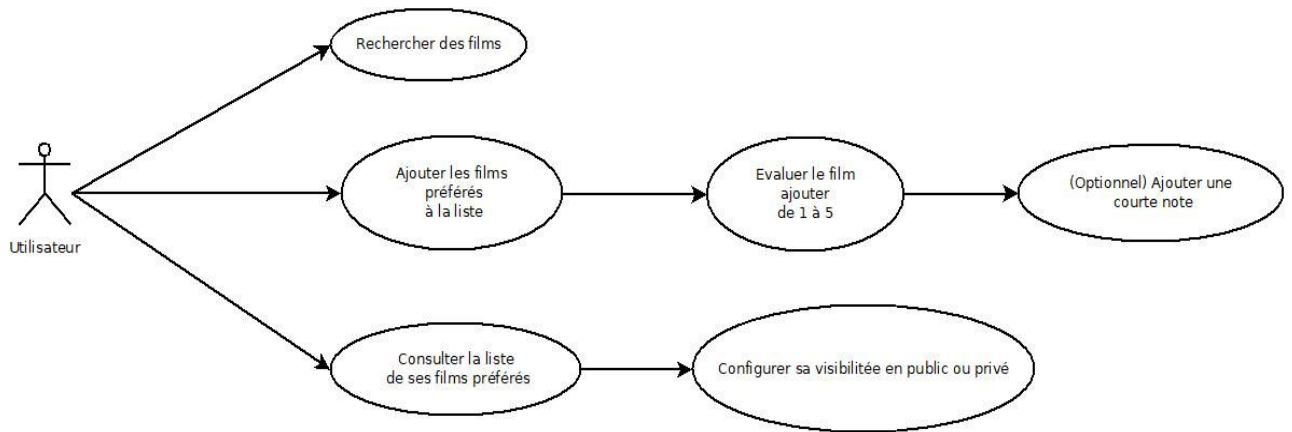
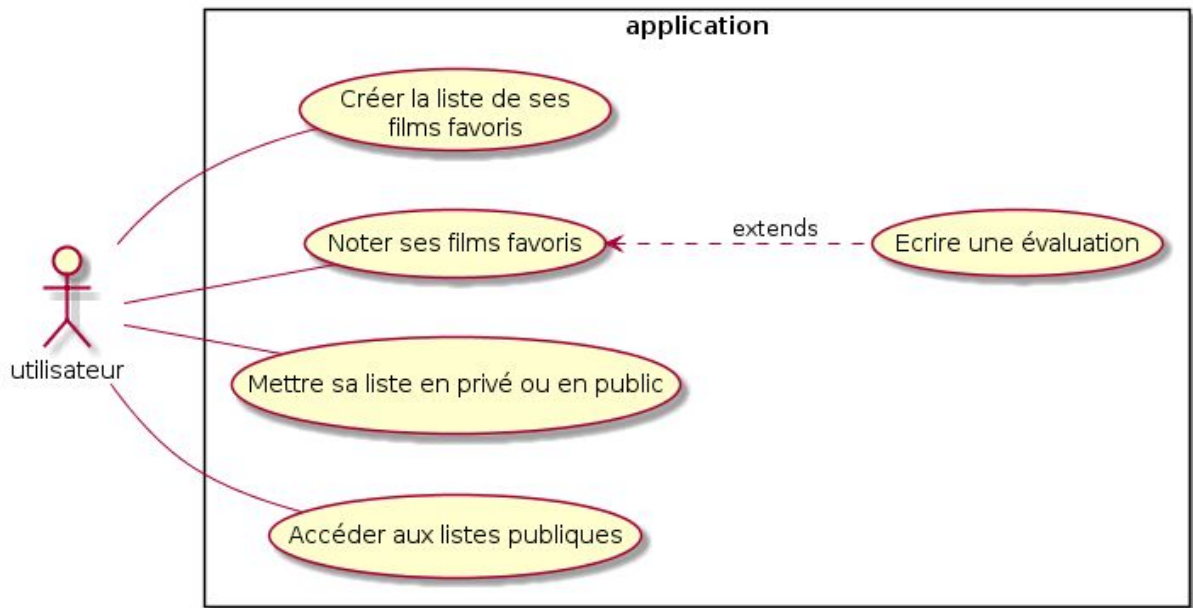
Packages :

- Bootstrap-vue
- axios
- VeeValidator
- vue-router
- vuex

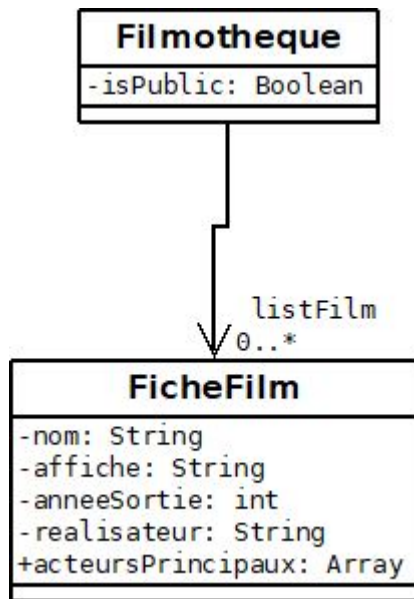
Répartition des tâches :

Ranjinie	Sébastien	Baptiste
Lien avec l'api TDMB	Ecriture des classes métiers	Méthode CRUD pour les favoris
Construction de la page d'accueil	Réalisation du modal pour les informations du film	Développement de la vue pour la gestion des favoris
	Tests unitaires pour la classe FicheFilm.spec.js	Tests unitaires pour la classe FavHandler.spec.js

I. Analyse de la demande**A. Diagramme des cas d'utilisation**



B. Diagramme de classe



C. Cahier des charges

Besoins:

- Application web de films préférés
- Voir la liste de tous les films
- Ajouter des films avec une évaluation notée de 1 à 5
- (Optionnel) Note courte allant avec le film écrite par l'utilisateur

Contraintes:

- Application en vue.js / Vuex / Validator
- Listes des films sauvegarder côté client via une solution de stockage
- Limiter le plus possible la taille mémoire des données de films
- Limiter le nombre de requêtes sur TMDB lors de la présentation de la liste des films

II - Gestion des favoris

a.1. L'ajout

L'ajout d'un favoris se passe au niveau de la page principale. En cliquant sur le coeur entouré en rouge sur la capture d'écran suivante :


Title	Description	Add
Inception	Dom Cobb est un voleur expérimenté, le meilleur dans l'art dangereux d...	 

Lors du clique, un modal s'affiche pour résumer les informations du film et permettre l'ajout en favoris :

a.2. Le modal

Réalisé avec la template de bootstrap, ainsi que modifier pour simplifier l'affichage, il permet d'afficher les informations nécessaire à la reconnaissance complète du film.

Voulez-vous ajouter ce film dans vos favoris ?



LE SEUL ESPOIR DE L'HUMANITÉ... N'EST PAS HUMAIN.

I, FRANKENSTEIN

Titre	Date De Sortie
I, Frankenstein	2014-01-22
Réalisateur	
Stuart Beattie	
Acteurs	
Aaron Eckhart	
Yvonne Strahovski	
Bill Nighy	
Jai Courtney	
Miranda Otto	

Annuler

Valider

La méthode fav(id) permet de récupérer les informations avec des appels à l'api et modifie le store:

```
fav(id) {
```

Ce début de fonction permet la récupération des données déjà acquises sur le film

```
  this.$store.commit('SET_FILM', id);
```

```
  this.$store.commit('SET_LE_FILM', id);
```

```
  this.infoFilm = this.$store.getters.unFilm;
```

```
  this.titleFilmFav = this.infoFilm.title;
```

Ensuite nous allons récupérer les membres du film qui ont permis sa réalisation, composé des acteurs et de l'équipe de tournage

```
  axios.get('https://api.themoviedb.org/3/movie/${id.id}/credits?api_key=c4183e64dc74d13d605f6815173449f3&language=fr-FR')
```

```

        .then((response) => {
            if (response) {
                this.$store.commit('SET_CASTFILM', response.data.cast);
                this.$store.commit('SET_CREWFILM', response.data.crew);
                this.castFilm = response.data.cast;
                this.crewFilm = response.data.crew;
                this.realisateur = this.crewFilm[0].name;
            }
        });
    }
}

```

Enfin nous allons récupérer l'affiche du film

```

axios.get('https://api.themoviedb.org/3/movie/${id.id}?api_key=c4183e64dc74d13d605f6815173449f3
&language=fr-FR')
    .then((response) => {
        console.log("test : " + response.data);
    });
}

```

Et nous devons la reconstituer avec une base d'URL qui ne change jamais et la partie qui permet de l'identifier

```

        if (response) {
            this.$store.commit('SET_AFFICHEFILM', response.data.poster_path);
            this.codeAffiche = this.baseAffiche + response.data.poster_path;
        }
    });
}

```

Lorsque l'on clique sur le bouton *Valider* cela appelle la méthode `add_favoris()` :

```

add_favoris() {
    let fiche_film = new FicheFilm(this.infoFilm["title"],
    this.$store.getters.lAfficheFilm, this.infoFilm["release_date"],
    this.realisateur,
    this.castFilm)
    FavHandler.add_favoris(fiche_film)
}

```

Cette méthode crée un objet `FicheFilm` pour ensuite appeler la méthode `FavHandler.add_favoris()` qui ajoutera la fiche au film au `LocalStorage`. La méthode est la suivante :

```

/**
 * Cette methode permet d'ajouter un favoris au LocalStorage
 */
static add_favoris(fiche_film) {

    // On recuperer Les favoris
    let favoris = this.get_favoris();
}

```

```

// On ajoute Le nouveau favoris
if(favoris.indexOf(fiche_film) == -1){
    favoris.push(fiche_film);
    // On sauvegarde Les favoris
    this.save_fav_array(favoris);
}
}



```

Dans un premier temps, on récupère les favoris puis on vérifie que la la FicheFilm n'y est pas déjà, si elle n'y est pas alors on l'ajoute et on sauvegarde le nouveau tableau de favoris.

b. L’Affichage

Maintenant que l'on peut stocker les Favoris, il faut les afficher. Une nouvelle vue est donc créée, *Favorie.vue*.

Vos favoris

Nom	Realisateur	Anne Sortie	Acteur Principaux	Rating	Memo	Affiche	Supprimer
Isn't It Romantic	Todd Strauss- Schulson	2019- 02-13	Rebel Wilson Adam Devine Liam Hemsworth Priyanka Chopra Betty Gilpin Brandon Scott Jones	2/5 Modifier	Pas de memo Modifier		

Voici ce qui est affiché lorsqu'un film est en favoris. Nous avons décidé d'afficher seulement les premiers acteurs principaux du film.

c. La suppression

La suppression d'un favoris se fait en cliquant sur l'icone poubelle dans la dernière colonne en partant de la gauche du tableau. Lorsque l'on clique, cela appelle la méthode suivante :

```
/**
 * Cette methode prend en argument un tableau de FicheFilm et un objet
 * FicheFilm. Elle supprime l'objet FicheFilm si il est présent dans le
 * tableau, sinon elle lance une exception (@todo)
 * @param {Array[FicheFilm]} favs_array
 * @param {FicheFilm} item
 */
static del_fav(favs_array, item) {

    // On recupere l'index de l'objet
    let item_i = favs_array.indexOf(item);
    // On le supprime si il est trouve
    if (item_i != -1) favs_array.splice(item_i, 1);
    // On sauvegarde dans le LocalStorage le nouveau tableau de favoris
    this.save_fav_array(favs_array);
    // On retourne le tableau
    return favs_array;
}
```

Elle prend en paramètre le tableau de favoris et le favoris à supprimer. Si le favoris est bien présent dans le tableau alors il sera supprimé et la base de données locale LocalStorage sera mise à jour en sauvegardant le nouveau tableau grâce à la méthode

`this.save_fav_array(favs_array);`.

d. La modification

La modification concerne seulement deux paramètres : La note que l'utilisateur attribue au film ainsi que le memo qui est limité à 30 caractères. Nous allons dans un premier temps regarder ce qu'il se passe lorsque l'on souhaite mettre à jour la note que l'on donne au film.

Voici l'interface de mise à jour de la note :

Modifier votre note :



Le bouton en vert correspond à la note que le film a actuellement.

Lors du clic sur un des autres boutons, cela appelle la méthode

`FavHandler.update_rating()` qui effectue le changement. Voici son corps :

```
/**
 * Cette methode permet de modifier la note d'un film
 * return : Retourne la fiche modifier (ou pas si ! 0 >= rating >= 5)
 * @param {FicheFilm} fiche_film
 * @param {int} rating
 */
static update_rating(fiche_film, rating) {
    // On verifie que la note est entre 0 et 5
    if(rating >=0 && rating <= 5) fiche_film.rating = rating;

    // On retourne la fiche
    return fiche_film;
}
```

Elle prend en paramètre un objet `FicheFilm` qui représente le film pour lequel on souhaite modifier la note ainsi qu'un entier, *rating* qui représente la nouvelle note.

Si la note est comprise entre 0 et 5 alors une nouvelle note sera attribuée.

Pour la mise à jour du memo, le fonctionnement est assez similaire, voici l'interface de modification :

Modification du mémo :

Maximum 30 caractères



Modifier

Si le mémo du film est vide le champ de texte ne sera pas pré-rempli mais si il y en a déjà un alors il sera pré-rempli. La modification est effective lorsque l'utilisateur clique sur le bouton

Modifier. Lors du clique, la méthode *FavHandler.update_memo()* est appelé. Voici son corps :

```
/**
 * Cette methode permet de modifier Le memo d'un film
 * return : Retourne la fiche modifier (ou pas si ! 0 >= memo.lenght >= 30)
 * @param {FicheFilm} fiche_film fiche a modifier
 * @param {String} memo nouveau memo
 */
static update_memo(fiche_film, memo) {
    if(memo.length >= 0 && memo.length <= 30) fiche_film.memo = memo;

    return fiche_film;
}
```

Cette méthode est vraiment semblable à celle qui permet la modification de la note. Elle prend en paramètre *fiche_film* qui est un objet *FicheFilm* et *memo* qui est la nouvelle valeur potentielle du mémo du film. Si la longueur du mémo est compris entre 0 et 30, alors le nouveau mémo est adopté.

III - Les tests unitaires

a. Les tests pour les méthodes de la classe FavHandler

Pour la classe **FavHandler**, deux tests ont été écrit.

Le premier est un test permettant de vérifier l'ajout d'un favoris dans le LocalStorage. Voici son code :

```
test("Test d'ajout de favoris", () => {
    // On recupere Les favoris avant l'ajout pour avoir le nombre de favoris
    let favs = FavHandler.get_favoris();

    // On ajoute le favoris
    let fiche = new FicheFilm("Nom du film", "/blabla.jpg", "20919", "real",
["act1", "act2"], 0, "");
    FavHandler.add_favoris(fiche);

    // On recupere le Local storage après l'ajout
    let favoris_after_add = FavHandler.get_favoris();

    // Tests
    try {

        expect(favoris_after_add.length).toEqual(favs.length + 1);
    } catch (e) {
        console.log(e)
    } finally {
        // Peu importe ce qu'il se passe on vide le LocalStorage
        window.localStorage.clear();
    }
}
```

```
    }  
  });
```

Pour ce test, dans un premier temps on récupère les favoris qui sont déjà stockés, puis on crée et ajoute une FicheFilm aux favoris. On récupère ensuite une nouvelle liste de favoris qui correspond à l'ancienne plus le nouveau que l'on vient d'ajouter.

Pour finalement tester si la longueur du nouveau tableau de favoris correspond à la longueur du tableau initial + 1. Ce test est entouré d'une déclaration try/catch/finally pour être sûr que le LocalStorage soit vidé à la fin.

Le second est un test qui vérifie la suppression d'un favori. Voici également son code :

```
test("Test de suppression d'un favoris", () => {  
  // On créer un tableau de favoris avec un seul favoris  
  let fiche = new FicheFilm("Nom du film", "/blabla.jpg", "20919", "real",  
["act1", "act2"], 0, "");  
  let favs = [fiche];  
  
  // On supprime le favoris  
  favs = FavHandler.del_fav(favs, fiche);  
  
  // On vérifie qu'il a bien été supprimé  
  expect(favs).toEqual([]);  
});
```

Ce test est plus simple mais il permet de tester si la suppression d'un favori est effective.

b. Les tests pour les méthodes de la classe FicheFilm

Pour la classe FicheFilm 3 tests ont été écrits.

Pour le premier test, on regarde si la création d'une fiche de film se réalise correctement:

```
test('Exemple test cas général de FicheFilm', () => {
```

```
  //création du film
```

```
    const film = new FicheFilm("avatar", "/1sd5g3g486gs1.png",  
"2019-01-01", "Steven Spielberg", "Bruce");
```

```
  //vérification des bonnes attribution des variables
```

```
    expect(film.nom).toMatch('avatar');
```

```
    expect(film.affiche).toMatch('/1sd5g3g486gs1.png');
```

```
    expect(film.anneeSortie).toMatch('2019-01-01');
```

```
    expect(film.realisateur).toMatch('Steven Spielberg');
```

```
    expect(film.acteurPrincipaux).toMatch('Bruce');
```

```
});
```

Pour le second test on vérifie que peut importe quel champ soit vide, il ne posera pas problème à la création de sa fiche film:

```
test('Exemple où la FicheFilm est vide', () => {  
  const film = new FicheFilm("", "", "", "", "");  
  expect(film.nom).toMatch('');  
  expect(film.affiche).toMatch('');  
  expect(film.anneSortie).toMatch('');  
  expect(film.realisateur).toMatch('');  
  expect(film.acteurPrincipaux).toMatch('');  
});
```

Enfin pour le troisième test on vérifie que plusieurs fiches de film peuvent être créés sans poser problème:

```
test('Exemple avec plusieurs FicheFilm', () => {  
  const film1 = new FicheFilm("avatar", "/1sd5g3g486gs1.png",  
    "2019-01-01", "Steven Spielberg", "Bruce");  
  const film2 = new FicheFilm("Koh-Lanta", "/1515s6gfs56h.png",  
    "2014-01-01", "Denis Brogniart", "Patrick");  
  expect(film1.nom).toMatch('avatar');  
  expect(film1.affiche).toMatch('/1sd5g3g486gs1.png');  
  expect(film1.anneSortie).toMatch('2019-01-01');  
  expect(film1.realisateur).toMatch('Steven Spielberg');  
  expect(film1.acteurPrincipaux).toMatch('Bruce');  
  expect(film2.nom).toMatch('Koh-Lanta');  
  expect(film2.affiche).toMatch('/1515s6gfs56h.png');  
  expect(film2.anneSortie).toMatch('2014-01-01');  
  expect(film2.realisateur).toMatch('Denis Brogniart');  
  expect(film2.acteurPrincipaux).toMatch('Patrick');  
});
```