

AZAP

application de gestion de films

Dossier de projet

Projet réalisé dans la cadre de la présentation au **Titre Professionnel Développeur Web et Web Mobile**

Réalisé par
Maeva COURRIN
Promotion Sinbad - 2021

Organisme de formation
Ecole O'clock

Sommaire

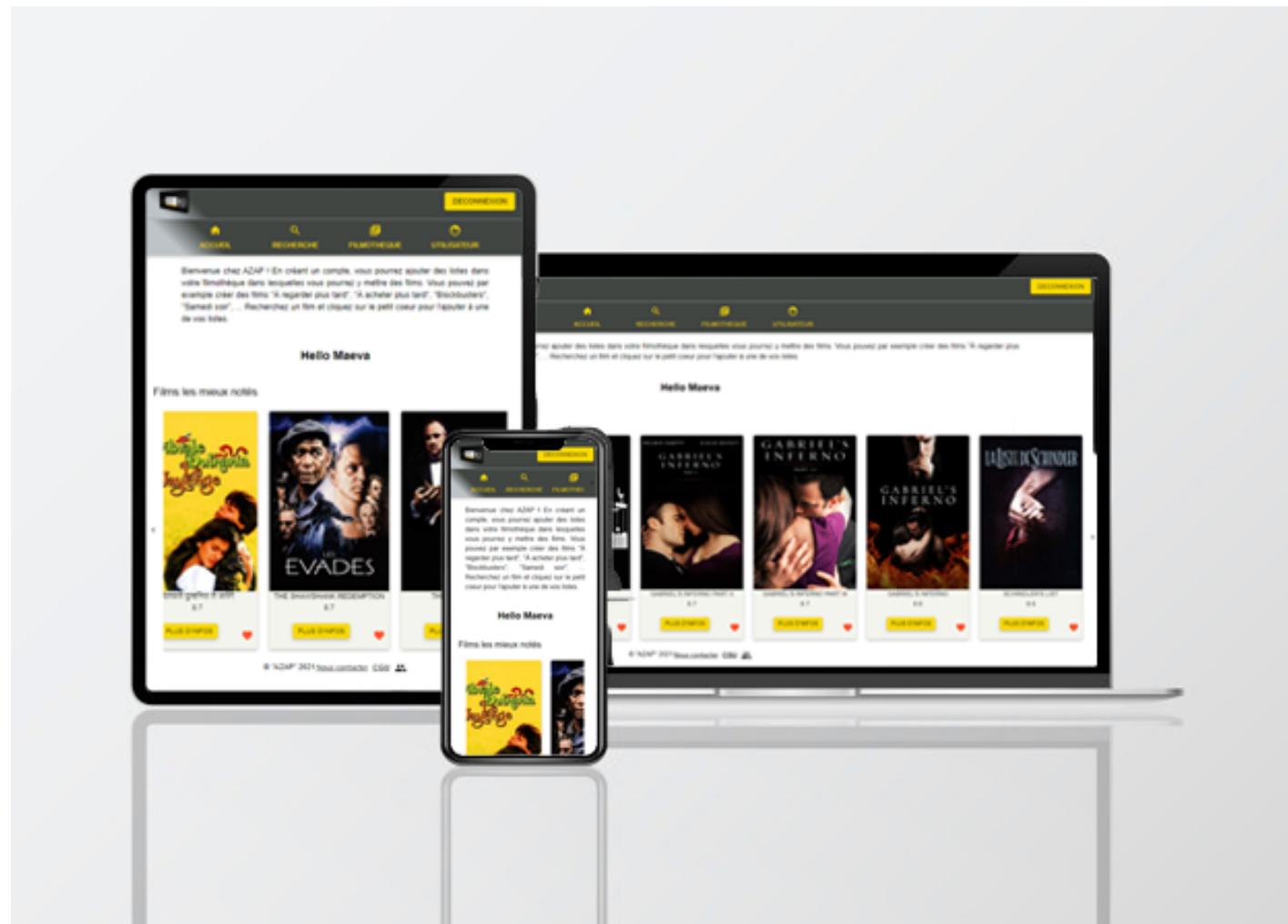
I - Introduction	p 3
II - Cahier des charges	p 5
Fonctionnalités du projet	p 5
Minimum Viable Product	p 7
Wireframes	p 8
Charte graphique	p 9
Rôles utilisateurs	p 10
User Stories	p 11
Modèle Conceptuel de Données	p 12
Modèle Logique de Données	p 13
Dictionnaire de données	p 14
Routes	p 15
III. Spécifications technique	p 16
Technologies du front-end	p 17
Technologies du back-end	p 18
Sécurité de l'application	p 19
IV. Organisation de l'équipe	p 20
V. Compétences du référentiel	p 24
CP1 : Maquetter une application	p 24
CP2 : Interface statique et adaptable	p 25
CP3 Interface web dynamique	p 27
CP5 : Créer une base de données	p 29
CP6: Composant d'accès aux données	p 32
CP7: Partie back-end d'une application web	p 37

VI. Réalisation personnelles	p 38
VII. Jeu d'essai	p 47
VIII. Veille sur les vulnérabilités de sécurités	p 49
XI. Résolution de problème: Recherche et traduction	p 54
X. Conclusion	p 60
XI. Annexes	p 61

Remerciement

Je voudrais tout d'abord remercier Estelle, sans laquelle ce projet n'aurait pas eu lieu, ainsi que mes autres camarades membres de l'équipe AZAP : Manuel, Yannick et Anthony; les membres et professeurs de ma promotion Simbad au sein de ces 6 mois intensifs chez O'Clock, où j'ai appris et continuerai d'apprendre et penser comme un développeur, mais aussi pour la qualité des cours.

I. Introduction à AZAP



Azap c'est quoi?

Vous est-il déjà arrivé de passer une heure entière à rechercher un film à regarder ce soir devant votre médiathèque HAVSTA Ikea ou sur les plateformes ?

Vous est-il déjà arrivé d'acheter deux fois le même film , alors qu'il trône tranquillement au-dessus de votre meuble télé ?

Nous vous avons compris !

- **Objectif**

L'objectif d'AZAP est de permettre à un utilisateur de **centraliser les informations** concernant ses films préférés mais pas seulement.

AZAP se veut une application de gestion de films, dans laquelle un utilisateur pourra créer ses propres listes, classer, trier et suivre les contenus qu'il souhaite voir et/ou acheter.

L'application est destinée à tous les publics, notamment pour les cinéphiles qui souhaitent pouvoir gérer efficacement les films qu'ils possèdent ou qu'ils aimeraient voir et avoir.

Nous avons pensé notre application en tant qu'**outil**, le but étant de créer **une interface simple et épurée**, afin de *favoriser son utilisation au quotidien*.

L'application a donc été conçue en mobile-first, dans le but d'être utilisée plusieurs fois par jour ou de façon ponctuelle.

- **Améliorations**

Dans une version plus avancée, il est prévu d'implémenter une solution de lecture de codes-barres.

Cette solution permettrait d'améliorer l'expérience utilisateur en proposant une application plus nomade dont l'expérience serait d'autant plus personnalisée et en adéquation avec les besoins de nos utilisateurs.

II.Cahier des charges

2.1. - Conceptualisation de l'application : Fonctionnalités du projet

Les spécifications fonctionnelles **décrivent** et/ou **spécifient** les **fonctionnalités** d'un site, d'une application ou encore d'un logiciel en vue de sa réalisation.

Elles ont été définies en équipe lors du premier sprint selon le cahier des charges de l'application.

Page principale (visiteur)

- Bouton de connexion utilisateur
- Accès à la fonctionnalité de recherche
- L'accès aux autres fonctionnalités renvoi vers une page de connexion
- Accès aux page "Contact", "Qui sommes-nous" et "CGU"

Page principale (utilisateur)

- Bouton de déconnexion
- Accès à la fonctionnalité de recherche
- Accès à la fonctionnalité "Filmothèque"
- Accès à l'espace personnel

Page de login

- Formulaire de connexion

Page de création de comptes

- Formulaire de création de comptes

Page contact

- Formulaire de contact pour assistance

Page "Qui sommes-nous ?"

- Présente les personnes qui ont contribué au projet

Profil utilisateur

- Visualisation de ses informations personnelles
- Possibilité de modifier ses informations
- Suppression du compte

Page recherche

- Permet d'effectuer une recherche
- Boutons radios pour permettre une recherche par filtres
- Ajouter un film à la liste de son choix

Page filmothèque

- Bouton de création de liste
- Affichage des listes de l'utilisateur
- Possibilité d'éditer et supprimer une liste
- Possibilité de supprimer un film d'une liste

Page d'un film

- Voir les informations d'un films
- Ajouter un film à la liste de son choix

Page CGU

- Mentions légales

2.2 - Conceptualisation de l'application : Minimum Viable Product

MVP où *Minimum Viable Product* est la version **la plus minimaliste** possible d'un produit. Les retours effectués permettront d'améliorer le MVP, de l'enrichir avec de nouvelles fonctionnalités, *par itération*, jusqu'à **devenir le produit fini**.

Nous avons défini en équipe le minimum requis pour la première release d'AZAP. Celle-ci est séparée selon si *l'utilisateur est connecté ou non*.

Disponible sans authentification

- Accéder à un formulaire de création de compte
- Accéder à un formulaire de connexion pour s'identifier
- Accéder à la page d'accueil
- Pouvoir faire une recherche par titre, acteurs, réalisateur (barre de recherche)
- Pouvoir faire une recherche par filtre
- Visualiser les résultats de la recherche
- Accéder à une page des détails d'un film
- Page « Qui sommes-nous ? »
- Page de contact
- Page de mentions légales

Disponible seulement après authentification

- Pouvoir se déconnecter
- Accéder au profil utilisateur avec la possibilité de modification des informations
- Accéder à la page filmothèque avec la possibilité de création et modification

Évolution

Nous n'avons pas eu le temps de mettre en place le **système de mailing** utilisé pour les *mots de passe oubliés*.

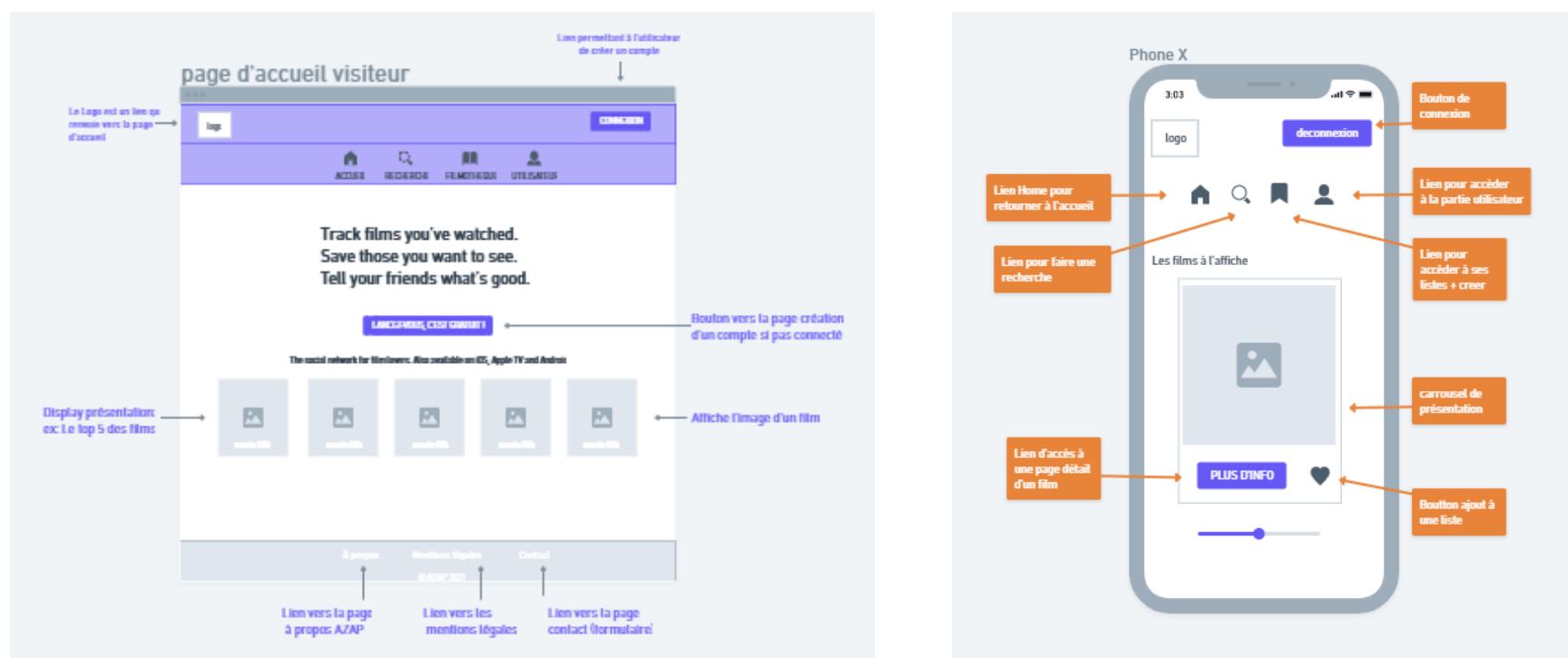
Nous avons également prévu l'implémentation d'un système de **lecture de code barre**, ainsi que l'ajout d'une partie "social" : **poster des commentaires** sous forme de threads, **créer des conversations** entre les membres, **attribuer une note aux films...**

2.3 - Mise en place de l'application : Wireframes

En tant que Product owner la création des **wireframes** m'a été confiée, ma production a toutefois été discutée, modifiée puis validée par l'ensemble de l'équipe.

Un **wireframes** ou maquette fonctionnelle est un schéma *utilisé* lors de la conception d'une interface utilisateur pour **définir les zones** et composants qu'elle doit contenir.

Nous avons utilisé l'outil web whimsical.com afin de partir sur la même base et obtenir des wireframes au rendu similaire.



2.4 - Mise en place de l'application : Charte graphique

Le logo a été réalisé par la fille d'Estelle. C'est un clin d'œil aux anciennes VHS .

Nous avons utilisé la police **Roboto** .

Les polices Roboto sont une famille de polices de caractères libre conçue par Christian Robertson pour Google notamment pour l'interface de la plateforme Android.

Depuis le 15 mai 2013, Roboto est la police d'écriture utilisée par *Google+* et *Google Maps*.

En termes de couleurs nous avons opté pour un thème **assez sombre**.

Nous avons cherché à évoquer le cinéma au travers du choix de notre palette.

Tout d'abord une couleur assez foncé pour rappeler les salles obscures puis le jaune en référence aux statuettes des oscars, où césars, et finalement une couleur claire pour apporter du contraste.

Concernant l'interface même du site, je me suis inspiré de l'application de *SensCritique*.



Roboto
SUNGASSES

Self-driving robot lollipop truck

Fudgesicles only 25¢

ICE CREAM

Marshmallows & almonds

#9876543210

Music around the block

Summer heat rising up from the boardwalk



2.5 - Utilisateurs et user stories : Rôles utilisateurs

Afin de définir aux mieux nos différents rôles utilisateurs nous nous sommes appuyés sur notre MVP, on retrouve donc deux types d'utilisateurs sur notre application:

- **Visiteur/Utilisateur non connecté :**

En tant que visiteur, l'utilisateur a accès à la **page d'accueil** du site décrivant l'application ainsi qu'à la fonction de **recherche** afin de découvrir les services et encourager la création d'un compte.

Les pages d'**inscription** et de **connexion**, ainsi que les pages de **contact**, **présentation de l'équipe du projet** et les **mentions légales**.

- **Utilisateur connecté :**

En tant qu'utilisateur connecté il a accès à **toutes les fonctionnalités** précédemment citées du site.

Il peut donc **créer des listes** depuis la page filmothèque et y **ajouter des films** depuis la fonction recherche.

Modifier où supprimer les informations à propos des listes, il peut également **modifier où supprimer** les informations de **son profil**.

2.6 - Utilisateurs et user stories : User Stories

Nous avons tous participé à la rédaction des **User Stories** tout en veillant à leur cohérence.

Une user story est une **explication** non formelle, générale d'une **fonctionnalité logicielle**, elle représente un **objectif final exprimé du point de vue de l'utilisateur**.

Cette formulation permet au Product Owner d'apporter une vision orientée client et d'identifier précisément la fonctionnalité et le bénéfice attendu.

Extrait des users Stories pour la page d'accueil en tant que visiteur :

ID	Thème	En tant que	J'ai besoin de	Afin de	Domaine	Sprint
1	Accueil	Visiteur	Accéder à un formulaire de création de compte	-Connaître le but du site	Interface	1
2	Accueil	Visiteur	Pouvoir s'inscrire	-Découvrir les services	Interface	1
3	Accueil	Visiteur	Accéder à un formulaire de connexion pour s'identifier	-Découvrir les services	Interface	1
4	Accueil	Visiteur	Pouvoir s'identifier	-Découvrir les services	Interface	1

2.7 - Mise en place de l'application : MCD / Modèle Conceptuel de Données

Aussi appelé **Modèle Entité-Association**.

Cette modélisation des données permet de représenter de façon rigoureuse un système de données, ou système d'information, sous forme d'entités et des relations qui les lient.

Avant de réaliser le dictionnaire de données, nous pouvons :

- **Dessiner nos entités** représentées par un rectangle.
- **Répartir leurs attributs** : les données qui les concernent sont listées sous le nom de l'entité.
- **Définir un** (ou plusieurs) attribut qui identifie l'entité de manière unique : **le déterminant** (ou identifiant, discriminant) qui identifiera l'entité, et est souligné tout en haut.
- **Identifier les relations** entre les entités et les nommer par un verbe.
- **Définir les cardinalités** : les quantités minimum et maximum qui peuvent exister entre deux entités A et B.

Cas de notre MCD pour la relation utilisateur-bibliothèque

Combien de listes peut créer un utilisateur au minimum ?

=> 0 (si l'utilisateur n'a pas créé de liste).

Combien de listes peut créer un utilisateur au maximum ?

=> n (plusieurs).

Une liste appartient à combien d'utilisateurs au minimum ?

=> 1 (il faut un utilisateur pour créer une liste).

Une liste appartient à combien d'utilisateurs au maximum ?

=> 1 (pas de co-auteur).

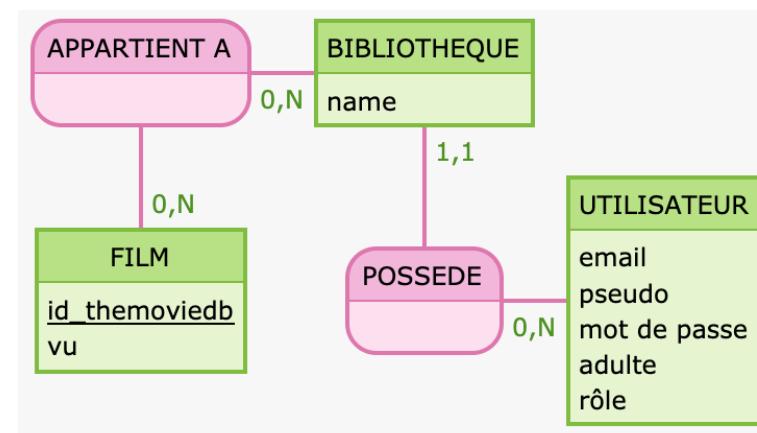


Schéma visuel du MCD généré via <http://mocodo.wingi.net>

2.8 - Mise en place de l'application : MLD/Modèle Logique de Données

Le Modèle Logique de Données est une étape vers le modèle physique final. Le MLD découle directement du MCD, en appliquant quelques règles - toute la réflexion a été faite au niveau du MCD.

- **Toute entité du MCD devient une table du MLD.** Les propriétés de ces entités deviennent les colonnes des tables. L'identifiant de l'entité devient la clé primaire de la table.
- **Si l'une des cardinalités max. vaut 1**, une clé étrangère est créée du côté de l'entité où se trouve le 1. Cette clé étrangère fera référence à l'identifiant dans la table associée. Exemple (Library est créé par 1 utilisateur), c'est donc la Library qui aura une clé étrangère vers l'Utilisateur.
- **Si les deux cardinalités max. sont n**, donc une relation «plusieurs à plusieurs» la relation devient une table à part entière en relation avec les deux entités. Cette table de liaison contient 2 clés étrangères vers les 2 tables à lier.

```
APPARTIENT A, ON BIBLIOTHEQUE, ON FILM
BIBLIOTHEQUE: _name
:
FILM: id_themoviedb, vu
POSSEDE, ON UTILISATEUR, 11 BIBLIOTHEQUE
UTILISATEUR: _email, pseudo, mot de passe, adulte, rôle
```

Description textuelle des entités et associations du mcd

```
BELONGS TO ( name, id_themoviedb )
LIBRARY ( id, name, #user_id )
MOVIE ( id, id_themoviedb, seen )
USER ( id, email, pseudo, mot de passe, adulte, rôle )
LIBRARY_HAS_MOVIE ( #id_themoviedb, #library_id )
```

MLD (modèle logique de données est la représentation des données d'un système d'information (SI)

2.9 - Mise en place de l'application : Dictionnaire de données

À partir du MLD, nous pouvons rédiger un **dictionnaire de données**.

Un dictionnaire des données est une **collection de métadonnées** ou de données de référence nécessaire à la conception d'une base de données relationnelle. Il revêt une importance stratégique particulière, car il est le vocabulaire commun de l'organisation.

- **Lister chacune des tables présentes dans le MLD**
- **Rapporter les relations entre les tables :**

Pour les relations dont les cardinalités sont (0,1) (1,1) (0,n) (1,n), il faut définir quelle table contient la **clé étrangère** faisant la liaison entre les deux tables.

Pour les relations dont les cardinalités sont de type (n,n), il faut créer une **table d'association** entre les deux tables.

Ensuite, pour chaque table, nous listons chaque information / champ qui la compose. Puis, pour chaque champ, nous allons :

- **Indiquer son type** (nombre, texte, booléen, calculé à partir d'autres informations...)
- **Définir ses spécificités ou contraintes** (clé primaire, auto-incrémenté, unique, non nul...)
- **Ajouter une description**
- **Ajouter un commentaire si besoin**

Nom symbolique	Description (rôle)	Domaine (Postgres)	Types (Postgres)	Commentaires	Contraintes, règles de calcul
<i>Library</i>					
id	Identifiant unique	N (Entier)	INT		Automatique
name	Nom de la library créer par l'utilisateur	Text	VARCHAR(50)		Obligatoire
user_id	L'identification à la relation entre la library et l'utilisateur	N (Entier)	INT		
<i>Library_has_Movie</i>					
id_themoviedb	identifiant unique	N (Entier)	INT	Identifiant du film dans the movie database	
library_id	identifiant unique	N (Entier)	INT	Identifiant de la library créer par l'utilisateur	

Extrait du dictionnaire de données. Voir annexe version complète

2.10 - Mise en place de l'application : Routes

Les routes représentent ici les **endpoints accessibles** de notre application.

HTTP Method : chaque requête effectuée devra décrire correctement l'action qu'elle souhaite effectuer auprès du serveur. Si ce n'est pas correct, le serveur vous renverra une erreur. Il y a 4 actions possibles (Get, POST, PUT & DELETE)

Chemin : endpoint, chemin d'accès à la route, l'url.

Description : action applicable

Retour : Contenu de la route

ID	Méthode	Chemin	Description	Retour
1	GET	/	Affiche la page d'accueil générale	JSON "Top 20 des films les plus likés"
2	GET	/login	Affiche la page pour se connecter	Redirect vers la page de connexion
3	POST	/login	Soumet le formulaire de connexion	Redirect vers Homepage JSON object {message: "Bienvenue + user_name"} ou Redirect vers /signin JSON object {message: "Une erreur s'est produite"}
4	GET	/signin	Affiche la page avec formulaire d'inscription	Dirige vers la page d'inscription
5	POST	/signin	Soumet le formulaire d'inscription	Redirect vers Homepage JSON object {message: "Bienvenue + user_name"} ou Redirect vers /signin JSON object {message: "Une erreur s'est produite"}
6	GET	/logout	Déconnecte l'utilisateur	Redirect vers Homepage JSON object {message: "A la prochaine + user_name !"}.

Extrait du document des routes.
(Voir annexe version complète)

III. Spécifications technique

3.1 - Versioning

Le versioning d'AZAP a été réalisé à l'aide de l'outil **Git**. Git est un **système de gestion de versions partagé**, chaque projet dispose d'un **répertoire central**, (appelé *repo*) la particularité de cet outil réside dans le fait que tous les utilisateurs *participants* téléchargeant une copie de ce répertoire sur leur appareil.

Chacune de ces copies constitue une sauvegarde à part entière ce qui implique qu'une connexion permanente au réseau n'est pas nécessaire. **Les modifications effectuées (les commits) peuvent être partagées** à tout moment avec tous les autres participants au réseau **et être reprises** dans le répertoire si elles sont pertinentes (*les merges*).

Github est un logiciel, ou plateforme son rôle est d'héberger les différents projets qui utilisent Git. Concrètement, ils proposent une **interface graphique** qui en simplifie l'utilisation.

Pourquoi utiliser Git ?

- L'aspect backup régulier du code, **le versioning**, est essentiel dans le développement web.
- L'aspect **collaboratif** qu'offre Git devient rapidement essentiel lorsqu'il y a plus d'un développeur dans le projet.
- Il permet de garder une trace des **versions antérieures** du site. Très utile en cas de debug.

Versioning projet

Nous avons choisi d'héberger la partie client et la partie serveur de notre application au sein d'un même repo dans leur dossier respectif .

Afin d'éviter tous conflits et séparer les concepts, **une branche front et une branche back** ont été créées depuis la branche principale (master). Chaque équipe se positionne sur sa branche directive, et nous avons créé une nouvelle branche au titre explicite (ex. depuis le terminal : `git checkout -b login/create_login_button`) à chaque ajout de fonctionnalité (= User Story, représenté par un ticket sur Trello).

Une fois la fonctionnalité finie on effectue une PR (**Pull Request**) de sa branche vers la branche directive (front/back).

Une fois la PR validée, et s'il n'y a pas de conflit avec la branche de destination on **merge** la PR : le contenu de la branche de fonctionnalité est intégré à celle de destination.

3.2 - Technologies du front-end

La partie front-end de notre application a été réalisée avec **ReactJS**.

ReactJS est une **bibliothèque (ou framework) JavaScript** libre développée par Facebook depuis 2013. Le but principal de cette bibliothèque est de faciliter la création d'application web monopage, via la création de composants dépendant d'un état et générant une page (ou portion) HTML à **chaque changement d'état**.

React est une bibliothèque qui ne gère que l'interface de l'application, considéré comme la vue dans le modèle MVC. La bibliothèque se démarque de ses concurrents par sa flexibilité et ses performances, en travaillant avec un **DOM virtuel** et en ne mettant à jour le rendu dans le navigateur qu'en cas de nécessité.

React permettant de faire ce que l'on appela une **SPA (Single Page Application)**, nous avons utilisé **React Router Dom** afin d'avoir des routes virtuelles (seuls les composants présents sur la route sont placés dans le DOM virtuel, le résultat est direct. L'utilisateur n'a pas l'impression d'avoir subi un chargement de page). Ces routes nous permettent d'accéder aux pages de recherche, filmothèque, profil, ...

La communication vers les données de l'API s'est faite via la librairie **Axios**.

La librairie Axios est une librairie **JavaScript** basée sur les **promises**, une fonctionnalité spécifique du langage JavaScript. La centralisation des données s'est faite à l'aide des **states de React** qui permettent un stockage local des données dans nos composants.

Concernant la partie graphique nous avons choisi d'utiliser **MaterialUI**.

MaterialUI est une librairie de composants React, conçue sur les principes du Material Design de Google en 2017 par une petite équipe de développeurs.

C'est une des bibliothèques de composants les plus populaires, il était donc intéressant pour nous de développer un projet avec.

3.3 - Technologies du back-end

La partie back-end de notre application a été réalisée avec **PostgreSQL** et le framework **Express JS** dans un environnement **NodeJS**.

NodeJS est un environnement d'exécution JavaScript côté serveur

Express JS est un Framework basé sur NodeJS,. Il gère toutes les fonctions comme le reroutage, le HTTPS et la gestion des erreurs.

PostgreSQL est une **base de données SQL** open-source qui stocke les données sous forme de tableaux et utilise des contraintes, des déclencheurs, des rôles, etc. pour les gérer. Il donne la possibilité de définir ses propres types de données et d'écrire le programme dans différents langages.

Pour versionner notre base de données, l'équipe back a eu recours à **Sqitch** : Sqitch est un système de migrations permettant de versionner une base de données.

Une **migration Sqitch** est l'équivalent d'un commit Git : elle recense les instructions SQL permettant de passer d'un état donné à l'état suivant (le **deploy**) ainsi que les instructions SQL pour revenir à l'état initial (le **revert**). Il est aussi possible de fournir un script de test pour confirmer la validité du déploiement (le **verify**).

Pour l'authentification et la sécurité des données utilisateur nous avons intégré les packages **json web token** ainsi que **bcrypt**.

JSON Web Token (JWT) est un standard ouvert défini dans la RFC 75191. Il permet **l'échange sécurisé de jetons** (tokens) entre plusieurs parties. Cette sécurité de l'échange se traduit par la *vérification de l'intégrité et de l'authenticité* des données.

bcrypt quant à lui est un *algorithme de hachage unidirectionnel*: Il utilise l'algorithme Eksblowfish pour hacher les mots de passe. Eksblowfish garantit que tout état ultérieur dépend à la fois du sel et de la clé pour nous il s'agit du mot de passe de l'utilisateur.

Pour authentifier et sécuriser **les requêtes vers notre api** nous avons également intégré le **package cors**

Enfin le package “**node-postgres**” qui sert d'interface avec la base de données.

3.4 - Sécurité de l'application

Côtés client:

-Il est nécessaire de **s'authentifier** en tant que membre pour avoir accès à des opérations sur la base de données. Un **token JWT** est créé et stocké dans le locale Storage , celui-ci est stocké à chaque requête faite par l'utilisateur vers l'API

- Au clic sur une action déclenchant un appel vers l'API, **le token JWT** est passé dans le header de la **requête HTTP**. Si celui-ci est valide, l'opération continuera et nous traiterons les données de réponse côté client. Le cas échéant, un message avertira l'utilisateur sur l'erreur même afin de l'aider à comprendre le refus/l'erreur dans la requête

Côtés server:

- **En production**, seules les requêtes provenant du serveur front sont autorisées dans les **CORS**.
- L'appel à l'API ne peut se faire qu'après une **authentification réussie**, générant un **token** propre à l'utilisateur connecté.
- Pour chaque requête effectuée vers l'API, on vérifie la présence du **token JWT** et sa **correspondance** avec celui émis par l'application au moment de l'authentification. Si le token ne correspond pas ou est expiré, la requête ne pourra être effectuée et on enverra une erreur en réponse.

IV.Organisation de l'équipe

4.1 - Présentation de l'équipe

Pour les besoins de ce projet, une équipe de 5 développeurs a été créée, la répartition des postes est la suivante :

Front:

- **Dev front:** SANCE Manuel
- **Lead dev front:** SEVERO Yannick
- **Product owner:** COURRIN Maeva

Back:

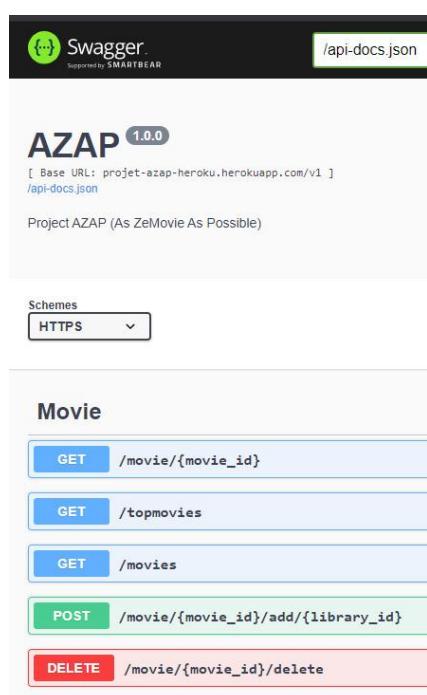
- **Lead dev back & Git master:** KOECHLER Estelle
- **Scrum master:** CAUDELI Anthony

4.2 - Organisation des tâches

Après validation de tous nos documents de projet en fin de sprint 0, nous avons représenté **chaque User Stories** dans un tableau Trello sous forme de tickets, labellisées selon si front/back et le composant associé: notre **backlog**.

Lors des différents sprints, chacun travaille sur un ticket sans oublier de les déplacer dans la liste suivante, ceci afin de permettre à tous d'avoir une vue d'ensemble de l'avancement du projet.

Pour suivre l'avancement des endpoint l'équipe back a mis en place une doc-api via swagger, cela nous a été très utile pour tester les end-points et avoir une mise à jour en temps réel.



The screenshot shows the Swagger UI for the AZAP API. At the top, it displays the base URL and the JSON documentation file. Below, the 'Movie' section is expanded, showing five API endpoints: a GET endpoint for a specific movie, a GET endpoint for top movies, a GET endpoint for all movies, a POST endpoint for adding a movie to a library, and a DELETE endpoint for removing a movie from a library.

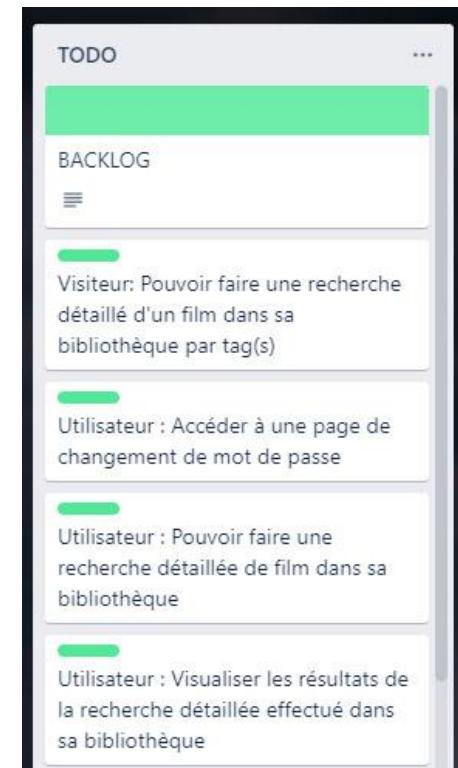
Meeting

Tous les matins nous commençons la journée par une réunion. Chacun évoque le travail effectué la veille, le travail prévu pour la journée, mais également et surtout les difficultés rencontrées afin de les résoudre aux plus soit via les connaissances partagées.

En fonction des besoins, les équipes étaient ensuite séparées, front ou back, également en pair programming.

Ces réunions ont toutes été retranscrites dans le journal de bord de l'équipe.

Ces réunions étaient aussi l'occasion de faire le point sur les branches et pull requests disponibles.



The screenshot shows a Trello board with a 'TODO' list. The backlog section contains four user stories: a visitor can perform a detailed search for a movie in their library by tag, a user can access a password change page, a user can perform a detailed movie search in their library, and a user can view the results of a detailed search performed in their library.

4.4 - Programme des sprints

Nous avons travaillé en sprint grâce à la méthode **AGILE SCRUM**. Chaque sprint a eu une durée d'une semaine, nous avions régulièrement, une fois par semaine, une réunion avec nos référents de promotion afin de faire un point sur l'avancement du projet, les sujets bloquants et de répondre à toutes les questions rassemblées au cours de la semaine.

- **Sprint 0**

Le premier sprint a été consacré à **l'élaboration du cahier des charges**, et des différents documents tels que MCD, MLD, wireframes, user stories ... mais aussi à la définition des rôles de chacun.

- **Sprint 1**

C'est le moment où nous avons commencé à coder.

Côté front: création du dossier de projet React dans la repo, **installation des packages**. *Répartition de l'intégration statique* pour toutes les pages, et on prépare la dynamisation avec les données de l'api TMDB.

Côté back : création du dossier de projet Postgres et **installation des packages** requis. On met en place la gestion de l'authentification et du token JWT tout en avançant sur l'API. Mise en place de la BDD, des entités, CORS et token JW tout en avançant sur les routes (movies et libraries)

- **Sprint 2**

On ajoute et améliore les fonctionnalités tout en corrigeant les bugs.

Côté front: Mise en place de la **connexion à l'API**. **Dynamisation** des pages via les endpoints, corrections des bugs.

Côté back : **Avancement** des sur les endpoint search. **Déploiement**.

- **Sprint 3**

On finit les fonctionnalités actuelles, **vérifie en équipe toutes les fonctionnalités** fonctionnelles ou non et **améliorations/corrections** des problèmes front/back rencontrés en vue de la présentation

V. Compétences du référentiel

Partie front-end : CP1 - Maquetter une application

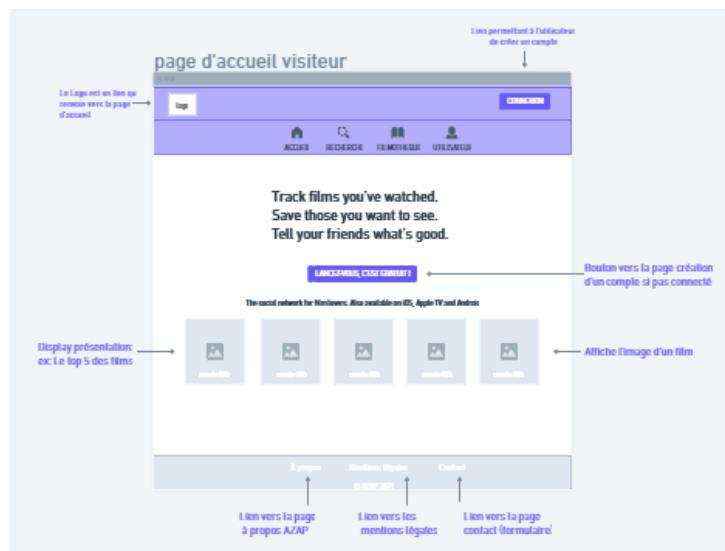
Pour la réalisation de la page d'accueil, nous nous sommes basés sur les **User Stories** de la page d'accueil (voir annexe).

A l'aide de l'application en ligne **whimsical**, nous avons commencé à réaliser le wireframe pour un **affichage mobile**, en utilisant comme référence dans ce cas un iPhone X (375x812px).

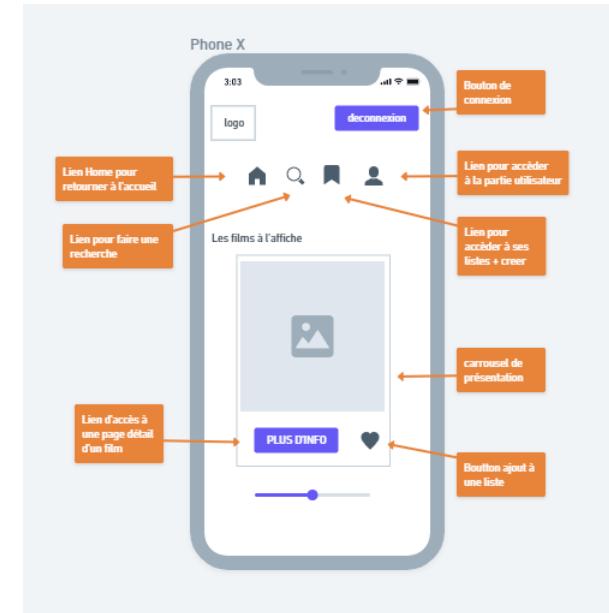
On commence par mettre en place le header avec le logo à gauche et le bouton de connexion à droite.

Puis la barre de navigation avec les icônes correspondantes aux composants.

Puis un composant carrousel qui présente une sélection de film, à l'intérieur de ce composant on ajoute des boutons call-to-action, si l'utilisateur n'est pas connecté il sera redirigé vers la page de connexion, s'il est connecté l'action sera exécutée.



wireframe Accueil, version desktop visiteur



wireframe Accueil, version mobile connecté

Après la création de la page en version mobile, nous avons pu décliner l'affichage pour un écran de PC (1920x1080px).

Dans cette version, le carrousel prend toute la vue de la fenêtre.

Partie front-end : CP2 - Interface statique et adaptable

L'application ayant été composée en **mobile-first**, il a fallu adopter son **affichage selon les supports utilisés** (téléphone, ordinateur...) afin d'optimiser l'affichage.

MaterialUI propose un système de custom très poussé, basé sur *les classes CSS, Grid CSS, media queries*.

Par défaut, MaterialUI implémente un thème avec des breakpoints et des méthodes pour utiliser ces breakpoints.

Default breakpoints

Each breakpoint (a key) matches with a *fixed* screen width (a value):

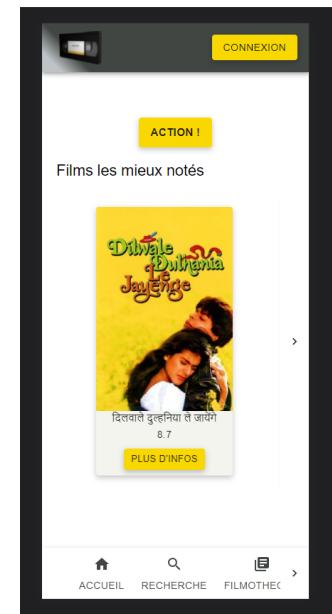
- **xs**, extra-small: 0px
- **sm**, small: 600px
- **md**, medium: 900px
- **lg**, large: 1200px
- **xl**, extra-large: 1536px

These values can be [customized](#).

CSS Media Queries

CSS media queries are the idiomatic approach to make your UI responsive. The theme provides four styles helpers to do so:

- [theme.breakpoints.up\(key\)](#)
- [theme.breakpoints.down\(key\)](#)
- [theme.breakpoints.only\(key\)](#)
- [theme.breakpoints.between\(start, end\)](#)



Dans cet exemple je vais détailler l'affichage de la page Home et de ses composants, *dans une version qui n'a pas été retenue par l'équipe*.

L'idée de départ était qu'en version mobile la navBar soit affichée en bas de l'écran, pour une meilleure ergonomie .



Ici on souhaite que sous une certaine taille de viewport la **navBar** soit positionnée **en bas de l'écran-affichage**

Pour cela on utilise la **méthode** fournis par **MaterialUI** :
theme.breakpoints.down("sm") ,

on va *créer un objet* avec cette méthode, et l'ajouter à notre classe CSS . (ligne 73)

On ajoute notre objet à la **classe root** de notre composant (ligne 74 - 76)

Pour remédier à un problème d'affichage, (le composant navBar se positionne sous le composant display) on ajoute un z-index: 1 pour donner une priorité de rendu à ce composant.

Enfin dans le composant **Home**, on *ajuste le rendu du composant display*.

De la même manière que précédemment, dans ce composant on souhaite ajuster la taille du display, on ajuste la à width 70%, enfin on supprime le texte de présentation pour rationaliser notre affichage en version mobile.

```
61 // == Css styles
62 const useStyles = makeStyles((theme) => ({
63   root: {
64     flexGrow: 1,
65     width: "100%",
66     //background: "Linear-gradient(45deg, #bdcc3c 10%, #424642 20%)",
67     //backgroundColor: "#424642",
68     display: "flex",
69     justifyContent: "center",
70     marginTop: 1,
71     boxShadow: "rgba(0, 0, 0, 0.24) 0px 3px 8px",
72   },
73   [theme.breakpoints.down("sm")]: {
74     position: "absolute",
75     bottom: "0",
76     zIndex: "1",
77     backgroundColor: "#FFFFFF",
78   },
79 },
80 icon: {
81   color: "#F8D800 !important",
82   opacity: 1,
83 },
84 [theme.breakpoints.down("sm")]: {
85   backgroundColor: "#FFFFFF",
86   color: "#424642 !important",
87 },
88 },
89 },
90 },
91 });
92 });
93 }
```

```
107   return (
108     <div className={classes.root}>
109       <Tabs
110         value={value}
111         onChange={handleChange}
112         variant="scrollable"
113         scrollButtons="on"
114         TabIndicatorProps={{
115           style: {
116             display: "none",
117           },
118         }}
119         aria-label="scrollable force tabs example">
120           <Tab
121             label="Accueil"
122             component={Link}
123             to="/"
124             icon={<HomeIcon />}
125             {...allyProps()}
126             className={classes.icon}
127           />
128         
```

Partie front-end : CP3 - Interface web dynamique

Le composant Library illustre le **dynamisme d'une page web** ainsi que le rafraîchissement du **DOM** offert par **React**. Sur cette page l'utilisateur à la possibilité de *créer, supprimer, et modifier* une liste.

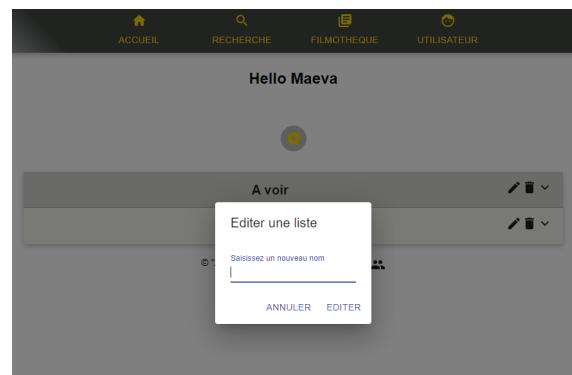
-Au click sur le bouton “+” une modal avec un formulaire s'ouvre pour permettre à l'utilisateur de créer une nouvelle liste. Cette donnée est stockée dans un state puis passée dans le corps de la requête HTTP pour être envoyée vers la BDD.

L'utilisation du hook `UseEffect` sur la fonction permettant de récupérer les listes permet un re-rendu du composant à la création ou à la modification de ce state, sans temps de chargement.

-Au click sur l'icône “crayon” une modal s'ouvre et permet à l'utilisateur de modifier la donnée “nom” d'une liste.

Le processus est similaire à l'ajout sauf que l'ID de la liste est passé en argument de la fonction appelée pour la mise à jour dans la BDD.

-Au click sur l'icône “corbeille”, la liste est supprimée, l'Id est passé en argument dans la fonction appelée pour la suppression dans la BDD.



- **Editer une liste**

Pour éditer une liste on clique sur le bouton prévu à cet effet, celui-ci déclenche l'ouverture d'une **modal superposée à l'interface utilisateur**. Cette modal comporte **un formulaire**, l'input permet d'enregistrer la valeur.



La fonction `onChange` de l'input déclenche la fonction, `handleChange` (ligne 96), celle-ci permet de **stocker la valeur** dans un state (ligne 65) grâce à la fonction `useState` de react..

La fonction `handleEditList` (ligne 74) est déclenchée à la **soumission du formulaire**.(ligne 134)

En paramètre de cette fonction **l'ID de la liste** (ligne 134) celui-ci est passé en `props` depuis le composant supérieur *Library* (ligne 62)

Cette fonction est **asynchrone** ce qui signifie que son traitement va s'opérer dans une boucle d'événement en utilisant une **promise**.

On prépare nos données pour exécuter notre requête vers la BDD avec **axios**.

Dans le **header** de notre requête on place notre **token** que l'on récupère depuis le local storage (ligne 79), puis on prépare la **donnée** à être envoyé vers notre BDD. (ligne 82)

On utilise ici une **méthode PUT** pour réaliser une **update** dans la BDD (ligne 85).

En paramètre de cette méthode on ajoute nos **objets** data et config.

Enfin on déclenche la fonction `getFilmsListed` (ligne 90).

Cette fonction est, elle aussi, passée en `props` depuis le composant supérieur *Library* (ligne 62).

Le déclenchement de cette fonction va entraîner un **re-rendu du composant** .

La modification du nom de la liste va apparaître immédiatement sans temps de chargement.

```
61
62  const ModalEditList = ({ liste, getFilmsListed }) => {
63    const classes = useStyles();
64    const [open, setOpen] = React.useState(false);
65    const [name, setName] = useState();
66
67
68  const handleEditList = async (listeId) => {
69    handleClose();
70    try {
71      const config = {
72        headers: {
73          Authorization: localStorage.getItem("token"),
74        },
75      };
76      const data = {
77        name: name,
78      };
79      await axios.put(
80        `https://projet-azap-heroku.herokuapp.com/v1/libraries/${listeId}/edit`,
81        data,
82        config
83      );
84      getFilmsListed();
85    } catch (e) {
86      console.log(e)
87    }
88  };
89
90
91
92
93
94
95
96  const handleChange = (e) => {
97    setName(e.target.value);
98  };
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130  <Button
131    type="submit"
132    onClick={(e) => {
133      e.stopPropagation();
134      handleEditList(liste.id);
135    }}
136    color="primary">
137    Editer
138  </Button>
```

Partie back-end : CP5 - Créer une base de données

La création de la base de données s'est faite selon les documents du cahier des charges. Nous avons choisi d'utiliser une base de données PostgreSQL.

PostgreSQL est un **système de gestion de bases de données relationnel** robuste et puissant, aux fonctionnalités riches et avancées, capable de manipuler en toute fiabilité de gros volumes de données, même dans des situations critiques.

Pour garantir que notre base de données puisse être maintenue selon les évolutions de notre projet, l'équipe back a eu recours à l'outil de versioning Sqitch.

Sqitch est un **système de migrations** permettant de versionner une base de données.

Une migration Sqitch est l'équivalent d'un commit Git : elle recense les **instructions SQL** permettant de passer d'un état donné à l'état suivant (*le deploy*) ainsi que les instructions SQL pour revenir à l'état initial (*le revert*). Il est aussi possible de fournir un script de test pour confirmer la validité du déploiement (*le verify*).

- **Création de la base de données**

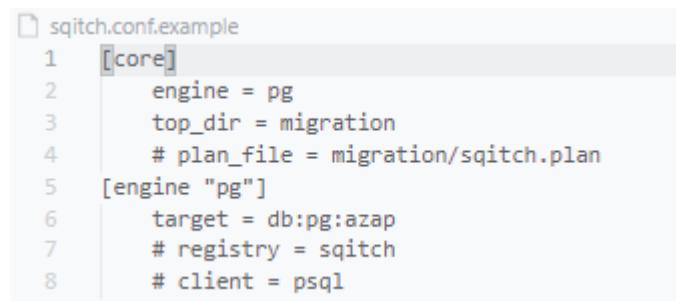
On commence par créer une base de données avec notre utilisateur postgres directement depuis le terminal, ici notre BDD se nomme : azap

Puis avec la commande: **sqitch init azap --engine pg --top-dir migrations --target db:pg:azap**, j'initialise Sqitch dans le projet. Grâce à cette commande mes fichiers de migrations et de scripts vont être créés par Sqitch.

- **Une migration de Sqitch consiste en 2 fichiers :**

sqitch.conf qui détaille la configuration du projet (SGBD utilisé et nom de la base de données, principalement)

sqitch.plan qui recense l'ordre des migrations du projet (chaque nouvelle migration ajoute une ligne à ce fichier)



```
sqitch.conf.example
1 [core]
2   engine = pg
3   top_dir = migration
4   # plan_file = migration/sqitch.plan
5 [engine "pg"]
6   target = db:pg:azap
7   # registry = sqitch
8   # client = psql
```

- **3 dossiers contenant les scripts de migration:**

-Dans **dossier deploy** on ajoute nos **requêtes DDL** pour créer nos tables

Le langage DDL (Data Definition Language) permet la création, modification et suppression des objets.

-Dans le **dossier revert**, on ajoute les commandes pour supprimer nos tables afin de pouvoir revenir à un état initial et apporter des corrections si nécessaires au script deploy.

-Dans le **dossier verify**, on ajoute un script de vérification pour s'assurer que nos tables et entités ont bien été créés

- **Déploiement:**

Une fois le script prêt, on peut le déployer dans notre base de données avec la commande **sqitch deploy**.

Dans notre cas Sqitch va être en mesure de trouver le chemin de notre BDD grâce au fichier **sqitch.conf**

```

  initsql  x
migration > deploy > initsql
  1  -- Deploy azap:init to pg
  2
  3  BEGIN;
  4
  5  DROP TABLE IF EXISTS "user";
  6  DROP TABLE IF EXISTS library;
  7  DROP TABLE IF EXISTS movie;
  8  DROP TABLE IF EXISTS library_has_movie;
  9
 10 -- Create domain email
 11 CREATE EXTENSION citext;
 12 CREATE DOMAIN email AS citext
 13   CHECK(VALUE ~ '^[a-zA-Z0-9.!#$%&'*+/=?_-{}~]+@[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,
 14   61}[a-zA-Z0-9])?(?:\.[a-zA-Z0-9](?:[a-zA-Z0-9-]{0,61}[a-zA-Z0-9])?)*$');
 15
 16 -- Create user
 17 CREATE TABLE "user" (
 18   id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
 19   email EMAIL NOT NULL UNIQUE,
 20   username VARCHAR(50) NOT NULL UNIQUE,
 21   password VARCHAR(100) NOT NULL,
 22   adult BOOLEAN DEFAULT false,
 23   role VARCHAR(50) DEFAULT 'member'
 24 );
 25
 26 -- Create Library
 27 CREATE TABLE library (
 28   id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
 29   name VARCHAR(50) NOT NULL,
 30   user_id int REFERENCES "user"(id)
 31 );
 32
 33 -- Create movie
 34 CREATE TABLE movie (
 35   id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
 36   id_themoviedb INT NOT NULL,
 37   seen BOOLEAN DEFAULT false
 38 );
 39
 40 -- Create Library_has_movie
 41 CREATE TABLE library_has_movie (
 42   movie_id INT REFERENCES movie(id) ON DELETE CASCADE,
 43   library_id INT REFERENCES library(id) ON DELETE CASCADE
 44 );
 45
 46 COMMIT;
 47

```

Une fois notre BDD créée et notre script exécuter, on va créer notre **module de connexion à la base de données**.

*Dans un premier temps on va définir nos **variables d'environnements** grâce au package **dotenv**.*

Ces variables vont permettre à l'application de **se connecter** à la base de données.

Dans un second temps on va créer notre client de connexion

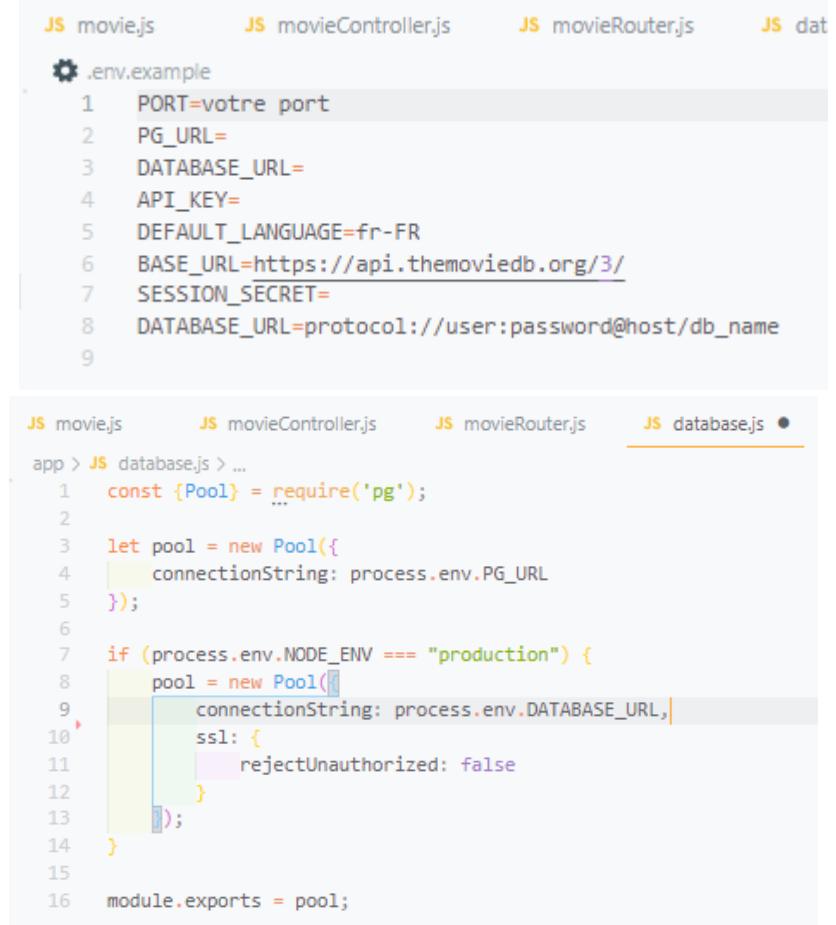
On commence par extraire la class **Pool** de **pg** (ligne 1)

On crée une **nouvelle instance** de Pool, puis on connecte cette instance grâce à nos **variables d'environnements**.

L'API ayant été déployée sur Heroku, on ajoute un use case en fonction de la variable d'environnement **NODE_ENV** (ligne 7).

Dans le cas où cette variable indique "**production**" l'API n'étant pas encore déployée la connexion se fera depuis la database de l'hôte.

Enfin, on ajoute une option (ligne 10 -11) pour éviter qu' Heroku ne rejette les connexions non https.



```
JS movie.js      JS movieController.js      JS movieRouter.js      JS database.js
.env.example
1 PORT=votre port
2 PG_URL=
3 DATABASE_URL=
4 API_KEY=
5 DEFAULT_LANGUAGE=fr-FR
6 BASE_URL=https://api.themoviedb.org/3/
7 SESSION_SECRET=
8 DATABASE_URL=protocol://user:password@host/db_name
9

JS movie.js      JS movieController.js      JS movieRouter.js      JS database.js
app > JS database.js > ...
1 const {Pool} = require('pg');
2
3 let pool = new Pool({
4   connectionString: process.env.PG_URL
5 });
6
7 if (process.env.NODE_ENV === "production") {
8   pool = new Pool({
9     connectionString: process.env.DATABASE_URL,
10    ssl: {
11      rejectUnauthorized: false
12    }
13  });
14
15 module.exports = pool;
```

Partie back-end : CP6 - Composants d'accès aux données

L'application a été conçue selon le pattern **MVC** pour Model View Controller, la vue se faisant via React. Pour plus de clarté et simplifier la maintenance de notre code, on applique le concept de **SOC (separation of concerns)**.

Router : On crée donc **plusieurs instances de Router**, selon nos endpoints définies dans le cahier des charges et leur logiques métiers.

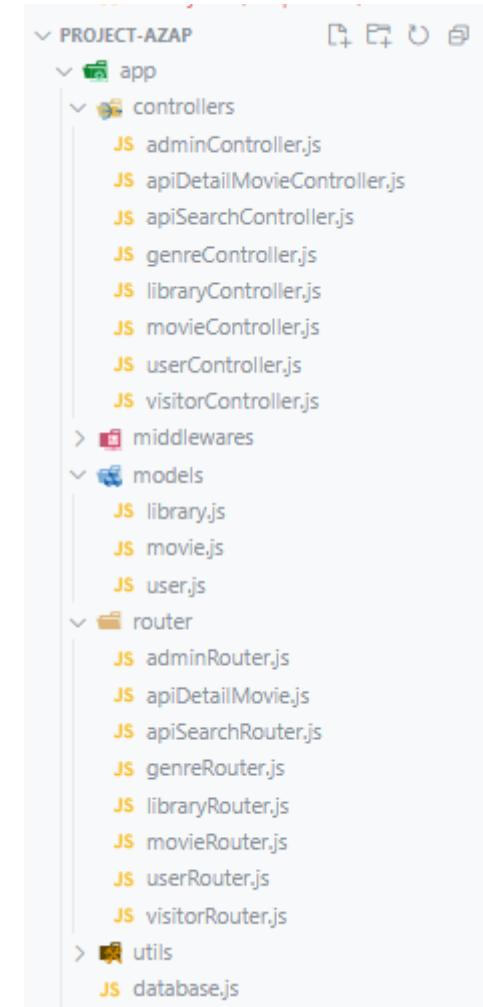
Pour chaque endpoints on va désigner **la route**, **la méthode HTTP** utilisée, et le **controller** requis.

Controllers : La couche Controller gère les requêtes des utilisateurs. Elle est responsable de **retourner une réponse** avec l'aide des couches Model et Vue.

Là encore on applique le concept de **SOC**, un controller est créé pour chaque entité de notre BDD.

Models : La couche Model représente la partie de l'application qui *exécute la logique métier*. Cela signifie qu'elle est responsable de **récupérer les données**, de les convertir selon les besoins. Chaque models dispose de ses propres accès CRUD à la BDD.

Là encore on applique le concept de **SOC** , et on crée un model par entité de la BDD



- Ajouter un film à une liste : Model

Dans notre **model** movie.js on commence par importer notre **client de connexion à la BDD**. (ligne 1)

On déclare la **class Movie** (ligne 16) et on prévoit un **constructor** (ligne 19 - 23).

La méthode *findAllMovies* est une méthode **asynchrone static** la communication se fait dans le sens BDD vers le client exclusivement, et en utilisant une promise.

En **paramètre** de notre méthode on place **l'id** de la library recherchée (ligne 33)

On déclare une **propriété rows**, cette propriété est extraite du résultat renvoyé par notre BDD. (ligne 35)

Cette propriété va contenir le **résultat** de notre **requête SQL** (ligne 35),

Notre **requête SQL** utilise une **fonction SQL**. Cette fonction nous permet de réaliser une **jointure** entre les tables, Library_Has_Movie et Movie.

Dans cette fonction **l'id** de la library recherchée a une **valeurs de paramètres**, référencées dans la commande avec **\$1** dans le WHERE, ceci afin d'éviter les injections SQL.

Le résultat obtenu est un **tableau d'objets**.

On va donc **mapper** sur ce résultat, (ligne 42) et pour chaque objet de ce tableau on va retourner une **nouvelle instance** de la **classe movie**.(ligne 42)

Donc tous les films contenus dans la liste.

The image shows a code editor with two tabs: 'movie.js' and 'find_all_movies.sql'.

movie.js:

```

JS movie.js  X
app > models > JS movie.js > ...
1  const db = require("../database");
2
3
16  class Movie {
17    static NoMovieError = NoMovieError;
18
19    constructor(data = {}) {
20      for (const prop in data) {
21        this[prop] = data[prop];
22      }
23    }
24
25
26    /*
27    static async findAllMovies(libraryId) {
28      try {
29        const { rows } = await db.query("SELECT * from find_all_movies($1)", [
30          libraryId,
31        ]);
32
33        if (!rows) {
34          throw new NoMovieError();
35        } else {
36          return rows.map((row) => new Movie(row));
37        }
38      } catch (error) {
39        if (error.detail) {
40          throw new Error(error.detail);
41        } else {
42          throw error;
43        }
44      }
45    }
46
47
48
49
50
51  }

```

find_all_movies.sql:

```

JS movie.js  ●  find_all_movies.sql  X  JS movieController.js  ●
migration > deploy > functions > find_all_movies.sql
1  -- Deploy azap:functions/find_all_Movies to pg
2
3  BEGIN;
4
5  -- Create function to find all movies in a Library of a user
6  CREATE FUNCTION find_all_movies(int) RETURNS TABLE(id int, id_themoviedb int, seen boolean) AS $$
7    SELECT id, movie.id_themoviedb, seen
8    FROM library_has_movie
9    JOIN movie ON library_has_movie.movie_id = movie.id
10   WHERE library_id=$1
11 $$ LANGUAGE SQL STRICT;
12
13 COMMIT;
14

```

- Ajouter un film à une liste : Controller

On commence par **importer** notre **model Movie** (ligne 2), puis on crée notre objet **movieController** (ligne 4)

On crée une méthode **addMovie** asynchrone (ligne 6)

Dans cette méthode on récupère depuis les **params** de l'url : *l'id du film* et *l'id de la liste* (ligne 8 et 9)

Dans un premier temps on va vérifier si le film est déjà présent dans la liste

On interroge le model à l'aide de la méthode **findAllMovies**, en paramètre de cette méthode l'id de la library. (ligne 11)

Puis on teste à l'aide d'une méthode **filter()** le tableau de résultat.(ligne 12)

Si un des id des objets obtenus dans les résultats correspond à l'id du film que l'on souhaite ajouter, on renvoie un status 409.(ligne 17)

Sinon, on utilise les données pour créer un objet à transmettre à la création de la **nouvelle instance de Movie**. (ligne 21 - 22)

On interroge à nouveau le model Movie mais cette fois avec une méthode **addMovie** et l'objet précédemment créé, en paramètre de cette fonction on passe l'id de la library.

```

JS movie.js          JS find_all_movies.sql      JS movieController.js •
app > controllers > JS movieController.js > movieController > deleteMovie
1  const { findAllByUser } = require("../models/library");
2  const Movie = require("../models/movie");
3
4  const movieController = {
5
6    addMovie: async (req, res) => {
7      try {
8        const id_themoviedb = parseInt(req.params.movie_id, 10);
9        const library_id = parseInt(req.params.library_id, 10);
10
11        const movies = await Movie.findAllMovies(library_id);
12        const isPresent = movies.filter(
13          (elem) => elem.id_themoviedb === id_themoviedb
14        );
15
16        if (isPresent.length) {
17          res.status(409).end();
18          return;
19        }
20
21        const data = { id_themoviedb: id_themoviedb, seen: false };
22        const movie = new Movie(data);
23
24
25        const newMovie = await movie.addMovie(library_id);
26
27        res.status(201).json(newMovie);
28      } catch (error) {
29        res.status(500).json(error);
30      }
31    },
32  },

```

- Ajouter un film à une liste : Model

La méthode **addMovie** est asynchrone et prend en paramètre l'id de la library.

Dans un premier temps on va ajouter le nouvel objet movie dans la table movie.

Pour se faire, on utilise ici une **requête préparée** (ligne 122). Dans cette requête l'id du film est passé en paramètre grâce à l'objet courant **this** , et est référencé dans la requête avec **\$1** on retourne toutes les données créées avec la commande *****

On déclare une **propriété rows**, qui va contenir le **résultat** de notre première **requête préparée (ligne 127)** : précisément **on cible l'id** de la première valeur du tableau d'objet retourné

Donc l'id du film que l'on vient d'ajouter à la table movie.
(ligne 128)

Puis on utilise cette l'id ainsi que celui de la library dans une **seconde requête préparée** (ligne 132)

Cette requête ajoute le film dans la table library_has_movie.

On retourne l'objet créé à notre **controller** (ligne 137)

```
118
119  async addMovie(library_id) {
120    try {
121      const preparedQueryMovie = {
122        text: `INSERT INTO movie(id_themoviedb) VALUES($1) RETURNING *`,
123        values: [this.id_themoviedb],
124      };
125
126      const { rows } = await db.query(preparedQueryMovie);
127      const movie_id = rows[0].id;
128
129      const preparedQueryLiaison = {
130        text: `INSERT INTO library_has_movie(movie_id, library_id) VALUES($1, $2)`,
131        values: [movie_id, library_id],
132      };
133
134      await db.query(preparedQueryLiaison);
135
136      return this;
137
138    } catch (error) {
139      console.log(error);
140      if (error.detail) {
141        throw new Error(error.detail);
142      } else {
143        throw error;
144      }
145    }
146  }
147}
```

- Ajouter un film à une liste : Router

Dans le fichier '**movieRouter**' on commence par **instancier** une **classe Router** depuis Express. (ligne 1 - 3)

On require le controller qui sera utilisé: *movieController* (ligne 5)

On crée notre route avec *la méthode HTTP* utilisée (ligne 27) puis **le endpoint** (ligne 28) enfin le **controller** et la **méthode** utilisée (ligne 30).

Sur nos endpoint les paramètres `movie_id` et `library_id` seront transmis depuis la partie front de notre application, on ajoute une **expression régulière**.

```
JS movie.js      JS movieController.js      JS movieRouter.js X
app > router > JS movieRouter.js > ...
1  const { Router } = require("express");
2
3  const router = Router();
4
5  const movieController = require("../controllers/movieController");
6  const checkConnected = require("../middlewares/checkConnected");
7

18 /**
19  * Add a new movie in database (when a user adds a movie to one of his libraries)
20  * Test Regex... | Test Regex...
21  * @route POST /movie/{movie_id}/add/{library_id}
22  * @group Movie
23  * @param {number} movie_id.path.required The movie_id of the movie to add
24  * @param {number} library_id.path.required The Library_id of the user to add movie
25  * @returns {Movie} 201 - The newly added movie
26  * @returns {string} 500 - An error message
27 */
28 router.post(
29   "/movie/:movie_id(\d+)/add/:library_id(\d+)",
30   checkConnected,
31   movieController.addMovie
32 );
```

Partie back-end : CP7 - Partie back-end d'une application web

On retrouve dans AZAP différents concepts et bonnes pratiques dans la mise en place de la partie back-end.

Utilisation de la POO (Programmation Orientée Objet)

La POO permet ici de réutiliser le code dans différents projets. Cela permet également d'avoir un code clair et organisé en identifiant chaque éléments présents dans l'application comme un objet ayant son contexte, ses propriétés et les actions qui lui sont propres.

Sécurité

- **CORS (Cross-Origin Resource Sharing)**

Nous avons utilisé le package “cors” qui nous a permis de définir les règles CORS. Ce package permet de définir les domaines qui auront accès à notre API REST. Ceux-ci étaient désactivés lors de l'utilisation du serveur en mode développement, mais activés en production avec uniquement acceptation des requêtes provenant du serveur front. Cette opération interdit le chargement à partir d'autres serveurs.

- **Variable d'environnement**

L'utilisation d'une variable d'environnement avec le fichier .env à la racine du projet permet de définir en un point un accès, ici la BDD et les informations de connexion. Ce fichier n'est pas publié sur le repo du projet pour éviter de le dévoiler. Seuls les développeurs participants à l'application ont accès à ces données.

- **Token JWT**

Le package JWT Authentication Bundle nous a offert la possibilité d'utiliser les JSON Web Tokens afin de protéger les ressources de notre API REST.

Lorsque l'utilisateur s'authentifie correctement, un token lui est retourné. Ce token est ensuite transmis côté front afin de permettre à l'utilisateur de lire ou d'écrire les ressources protégées à chaque requête.

VI. Réalisations personnelles

6.1 - Wireframes page filmothèque

En tant que Product owner la création des **wireframes** de la version mobile m'a été confiée.

Pour cela j'ai utilisé l'outil en ligne **whimsical** qui permet de réaliser rapidement et efficacement des wireframes grâce à ses nombreux outils et options.

J'ai ensuite consulté les **User Stories** pour la page en question ainsi que le dictionnaire des données pour connaître les informations présentes et les features à intégrer.

Tout d'abord on retrouve notre **header** : celui-ci contient notre *logo* et *un bouton de déconnexion*.

En dessous un **sélecteur**, pour choisir *la liste que l'on souhaite afficher*, même si *cette solution n'a pas été retenue dans la version finale*.

Vient ensuite **les options d'affichage** de la liste :

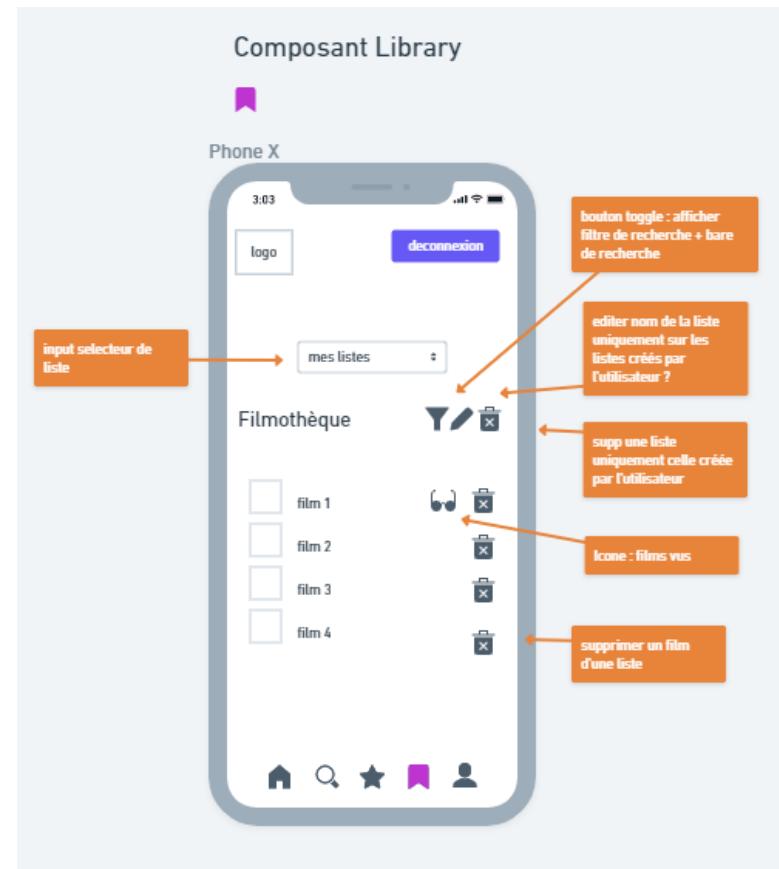
- les filtres de recherche interne à une liste
- L'édition d'une liste
- La suppression d'une liste

Les filtres de ne sont pas présents dans le MVP présenté, ils ont été néanmoins intégrés dans cette maquette.

On retrouve ensuite **les films** contenus dans une liste:

Chaque film dispose d'un **bouton de suppression**, ainsi que de l'**option "film vu"** *la aussi cette feature n'est pas présente dans notre MVP mais a aussi été intégrée.*

Finalement on retrouve notre **barre de navigation**, qui dans cette version se situe en bas de l'écran d'affichage là aussi *cette option n'a pas été retenue*.



Wireframe mobil composant Library

6.2 - Intégration de la page filmothèque

- **Intégration statique**

Après la création d'une **branche dédiée** à la création de cette page dans Git, j'ai commencé par réaliser **l'intégration statique**.

J'ai créé un nouveau dossier '**UserLibrary**' commençant par une majuscule pour indiquer que ceci était un composant,

J'ai créé un **composant fonctionnel UserLibrary**, (ligne 92) pour se faire MaterialUI offre une variété de composants customisables, après quelques recherches et en accord avec les autres membres de l'équipe j'ai choisi un composant qui diffère légèrement de notre wireframe d'origine.

J'ai choisi ce composant *pour son ergonomie et son rendu graphique*.

- **Appel à l'API**

Pour importer les data de nos utilisateurs, j'initialise **la requête vers notre API avec Axios**.

Dans le headers de la requête je récupère le token, pour ne pas que la connexion à l'API soit rejetée par le middleware.(ligne 108 - 122)

La fonction getFilmListed va me *retourner toutes les listes d'un utilisateur, et les films qu'elles contiennent*.

Cette requête **requiert l'ID unique de notre utilisateur**.

Pour ce faire , **je déstructure l'objet "user" du LocalStorage**. (ligne 106) *Cet objet user est envoyé dans la localStorage à la création d'un compte où, au loggin d'un utilisateur.*

Puis j'initialise un **state de react** pour **les données récupérées** (ligne 95)



```
JS index.js ...\\UserLibrary M ● JS index.js ...\\ModalEditList
src > components > UserLibrary > JS index.js > UserLibrary
91
92 const UserLibrary = () => [
93   const classes = useStyles();
94   const [isExpanded, setIsExpanded] = React.useState(false);
95   const [listes, setListes] = useState([]);
96
97   const handleChange = (panel) => (event, isExpanded) => {
98     setIsExpanded(isExpanded ? panel : false);
99   };

```



```
100
101 useEffect(() => {
102   getFilmsListed();
103   return () => setListes([]);
104 }, [ ]);
105
106 let { id, username } = JSON.parse(localStorage.getItem("user"));
107
108 const getFilmsListed = async () => {
109   try {
110     const config = {
111       headers: {
112         Authorization: localStorage.getItem("token"),
113       },
114     };
115     const response = await axios.get(
116       `https://projet-azap-heroku.herokuapp.com/v1/libraries/${id}/movies`,
117       config
118     );
119     setListes(response.data);
120   } catch (e) {
121   }
122 };
123

```

• Intégration dynamique

Une fois le lien avec l'API établie, j'ai pu **importer** directement les données **dans le corps de mon composant**.

Tout d'abord je récupère les listes, pour se faire j'utilise la fonction **map()** sur les données de mon state précédemment créé. (ligne 169)

```
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
```

```
return (
  <div className="user-library">
    <NavBar />
    <div className="create_button">
      <Typography component="h1" variant="h5" className={classes.typo}>
        Hello {username}
      </Typography>
      <ModalCreateList getFilmsListed={getFilmsListed} />
    </div>
    <div className={classes.root}>
      {listes.map((liste) => (
        <Accordion key={liste.id} className={classes.container}>
          <AccordionSummary id="panel1bh-header">
            <Typography variant="h6" className={classes.title}>
              {liste.name}
            </Typography>
          </AccordionSummary>
          <AccordionDetails>
            <div>
              <List>
                {liste.moviesDetails.map((filmListed) => (
                  <ListItem key={filmListed.id}>
                    <div className={classes.moviesliste}>
                      <CardContent
                        className={classes.cardContent}
                        component={Link}
                        to={`/movie/${filmListed.id}`}>
                        <CardMedia
                          className={classes.poster}
                          component="img"
                          alt="poster"
                          image={`https://image.tmdb.org/t/p/original/${filmListed.poster_path}`}
                          title="poster film"
                        />
                        <Typography className={classes.movieTitle}>
                          {filmListed.title}
                        </Typography>
                        <Typography />
                      </CardContent>
                      <DeleteIcon
                        className={classes.iconDeleteFilm}
                        onClick={(e) => [
                          e.stopPropagation(),
                          handleDeleteFilm(filmListed.id),
                        ]}
                      />
                    </div>
                  </ListItem>
                ))}
              </List>
            </div>
          </AccordionDetails>
        </Accordion>
      ))}
    </div>
  </div>
)
```

Une fois mes listes en place, je dois récupérer **les films qu'elles contiennent**.

Pour ce faire je vais également utiliser la fonction **map()** cette fois-ci je vais mapper sur l'objet "moviesDetails" renvoyé par notre API. (ligne 195)

Cet objet contient les données des films contenus dans une liste.

```
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
```

```
  </List>
  </div>
  <List>
    {listes.map((liste) => (
      <ListItem key={liste.id}>
        <div className={classes.moviesliste}>
          <CardContent
            className={classes.cardContent}
            component={Link}
            to={`/movie/${filmListed.id}`}>
            <CardMedia
              className={classes.poster}
              component="img"
              alt="poster"
              image={`https://image.tmdb.org/t/p/original/${filmListed.poster_path}`}
              title="poster film"
            />
            <Typography className={classes.movieTitle}>
              {filmListed.title}
            </Typography>
            <Typography />
          </CardContent>
          <DeleteIcon
            className={classes.iconDeleteFilm}
            onClick={(e) => [
              e.stopPropagation(),
              handleDeleteFilm(filmListed.id),
            ]}
          />
        </div>
      </ListItem>
    ))}
  </List>
)
```

- **Interaction côté client**

Une fois les données intégrées à mon composant, j'ai pu commencer à ajouter **les interactions côté client** nécessaires

Pour cette page, l'on souhaite que l'utilisateur puisse *supprimer un film, ajouter, supprimer, et éditer une liste*.

Le fonctionnement des modals pour la création et l'édition d'une liste sont décrite dans la partie précédente «**Partie front-end : CP3 - Interface web dynamique**».

- **Supprimer un film**

Je commence par **intégrer une icône delete** depuis MaterialUI (ligne 213)

Au clique sur cette icône, ma **fonction asynchrone handleDeleteFilm** sera déclenchée, *en paramètre de cette fonction l'id du film* en question. (ligne 217)

Cet id est utilisé dans la requête AXIOS permettant de supprimer un film dans une liste depuis notre API.(ligne 149)

Dans le headers de la requête je récupère le token, pour ne pas que la connexion à l'API soit rejetée par le middleware.(ligne 145)

Une fois la requête effectuée, je déclenche la fonction **getFilmsListed()**. (ligne 155)

Cette fonction est déclenchée lors du chargement de notre composant, (ligne 108) et permet une mise à jour de notre state initial grâce au **hook de react useEffect**. (ligne 101)

```
213     <DeleteIcon
214       className={classes.iconDeleteFilm}
215       onClick={(e) => [
216         e.stopPropagation();
217         handleDeleteFilm(filmlisted.id);
218       ]}
219     />
220   </div>
221 </ListItem>
```

```
140
141   const handleDeleteFilm = async (filmId) => {
142     try {
143       const config = {
144         headers: {
145           Authorization: localStorage.getItem("token"),
146         },
147       };
148       await axios.delete(
149         `https://projet-azap-heroku.herokuapp.com/v1/movie/${filmId}/delete`,
150         config
151       );
152     } catch (e) {
153       console.log(e.message);
154     }
155     getFilmsListed();
156   };
157
```

```
100
101   useEffect(() => {
102     getFilmsListed();
103     return () => setListes([]);
104   }, []);
105
106   let { id, username } = JSON.parse(localStorage.getItem("user"));
107
108   const getFilmsListed = async () => {
109     try {
110       const config = {
111         headers: {
112           Authorization: localStorage.getItem("token"),
113         },
114       };
115       const response = await axios.get(
116         `https://projet-azap-heroku.herokuapp.com/v1/libraries/${id}/movies`,
117         config
118       );
119       setListes(response.data);
120     } catch (e) {
121     }
122   };
123
```

6.3 - Composant réutilisable

Nous avons plusieurs **composants récurrents** dans l'application: *modal, formulaire, card*.

Ceci afin de conserver **une seule source de vérité** dans les composants, limitant ainsi la duplication du code et donc des erreurs, et optimiser la maintenance des composants et éviter les différences de rendu dans l'interface.

- **CardFilm**

Pour réaliser ce composant j'ai créé un **dossier CardFilm** dans le dossier "components", à l'intérieur le fichier index.js.

En étudiant la documentation de MaterialUI j'ai pu *intégrer les composants de cette bibliothèque pour créer un composant sur mesure pour notre application*.

MaterialUI fournit une multitude de composants, ces composants ont tous une utilité propre.

Pour mon composant CardFilm :

J'ai commencé par un **container Card**.

Puis un **container CardActionArea** pour la *zone cliquable du composant*, ici au clique sur l'affiche et sa bordure, l'utilisateur sera redirigé vers la page du film en question.

Suivi d'un composant **CardMedia** pour *intégrer l'image*.

Puis un composant **CardContent** qui va contenir du texte.

Enfin un container **CardActions** pour les *boutons cliquables*, le bouton "plus d'infos" et une modal.

```
44
45 const CardFilm = ({  
46   original_title: title,  
47   poster_path: poster,  
48   vote_average: vote,  
49   id,  
50 }) => [  
51   const classes = useStyles();  
52  
53   // je déstructure les infos de l'user du localStorage  
54   let username = JSON.parse(localStorage.getItem("user"));  
55  
56   return (  
57     <Card className={classes.root}>  
58       <CardActionArea component={Link} to={`/movie/${id}`}>  
59         <CardMedia  
60           component="img"  
61           alt="Display films"  
62           image="https://image.tmdb.org/t/p/w300/${poster}"  
63           title="Display films"  
64         />  
65         <CardContent className={classes.title}>  
66           <Typography variant="subtitle2" component="h2">  
67             | {title}  
68           </Typography>  
69           {vote}  
70         </CardContent>  
71       </CardActionArea>  
72       <CardActions className={classes.cardActions}>  
73         <Button  
74           component={Link}  
75           to={`/movie/${id}`}  
76           variant="contained"  
77           size="small"  
78           className={classes.button}>  
79           Plus d'infos  
80         </Button>  
81         {username ? <AddFilm id={id} /> : null}  
82       </CardActions>  
83     </Card>  
84   );  
85  
86  
87   export default CardFilm;
```

composant *CardFilm*

Pour styliser mon composant j'ai utilisé la fonction **makeStyles** de *MaterialUI*.

Cette fonction nous est utile pour **créer des objets JavaScript** (ex: ligne 19) ces objets sont ensuite passer en paramètre de nos composants grâce à la **props classes** de *MaterialUI* (ligne 65).

Enfin la variable **UseStyle** est utilisée comme un *hook* de *React*.

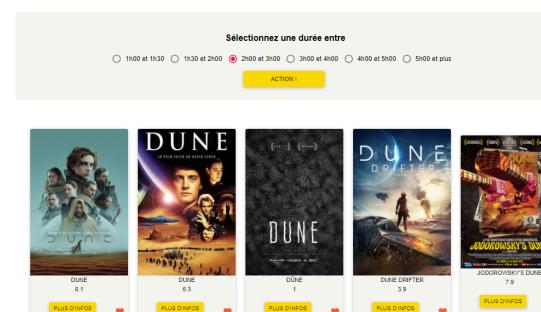
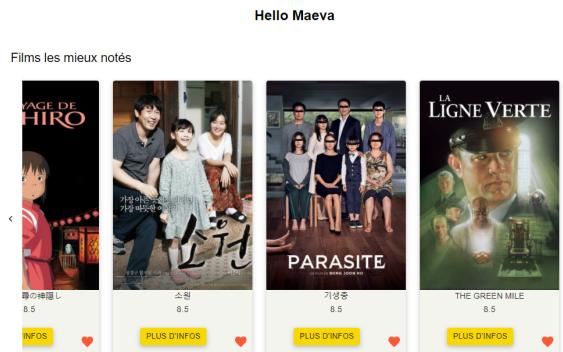
Une fois mon composant créé , celui-ci **peut être importé** dans les composants qui le nécessitent.

```
188      })
189      aria-label="scrollable force tabs example"
190      >
191        {trending.map((film) =>
192          <Tab
193            key={film.id}
194            icon={<CardFilm {...film} />}
195            {...allyProps(0)}
196          />
197        )));
198      </Tabs>
```

composant *cardFilm* dans *Home*

```
85      })
86      aria-label="scrollable force tabs example"
87      >
88        {films.map((film) =>
89          // <Tab icon={<CardFilm key={film.id} {...film} />} />
90          <Tab key={film.id} icon={<CardFilm {...film} />} />
91        )));
92      </Tabs>
93    );
94  );
95  >
```

composant *cardFilm* dans *Seacrh*



```
JS index.js          JS CardFilm.js X
src > components > CardFilm > JS CardFilm.js > CardFilm
1  // == Import NPM
2  import React from "react";
3  import { Link } from "react-router-dom";
4  // == Import library @material-ui
5  import { makeStyles } from "@material-ui/core/styles";
6  import {
7    Card,
8    CardActionArea,
9    CardActions,
10   CardContent,
11   CardMedia,
12   Button,
13   Typography,
14 } from "@material-ui/core";
15 // == Import Components
16 import AddFilm from "../AddFilm";
17 // == Css styles
18 const useStyles = makeStyles({
19   root: {
20     backgroundColor: "#F3F4ED",
21     maxWidth: 310,
22     boxShadow: "rgba(0, 0, 0, 0.24) 0px 3px 8px",
23     transition: "transform 0.15s ease-in-out",
24     "&:hover": {
25       transform: "scale3d(1.05, 1.05, 1)",
26     },
27   },
28   button: {
29     backgroundColor: "#F8D800",
30     margin: "auto",
31     display: "flex",
32     alignItems: "baseline",
33     boxShadow: "rgba(0, 0, 0, 0.24) 0px 3px 8px",
34   },
35   title: {
36     margin: 0,
37     padding: 0,
38   },
39   cardActions: {
40     display: "flex",
41     flexWrap: "wrap",
42   },
43 });
44
```

6.4 - Router

Le routage est la capacité pour notre application **d'afficher différentes pages**.

Cela signifie que l'on va donner à notre utilisateur l'illusion qu'il navigue entre plusieurs pages, avec différentes urls, alors que React va s'occuper de charger et rendre les composants correspondants.

Par défaut React ne possède pas de routage, il est donc nécessaire dans notre cas d'installer le **package react-router-dom**.

Puis on va réaliser nos import de composants depuis **react-router-dom**.

Le **BrowserRouter/Router**, Il crée ce qu'on appelle *un Contexte*, un moyen de diffuser une information à d'autres composants,

la **Route** conditionne *le rendu de nos éléments* en fonction de l'url active.

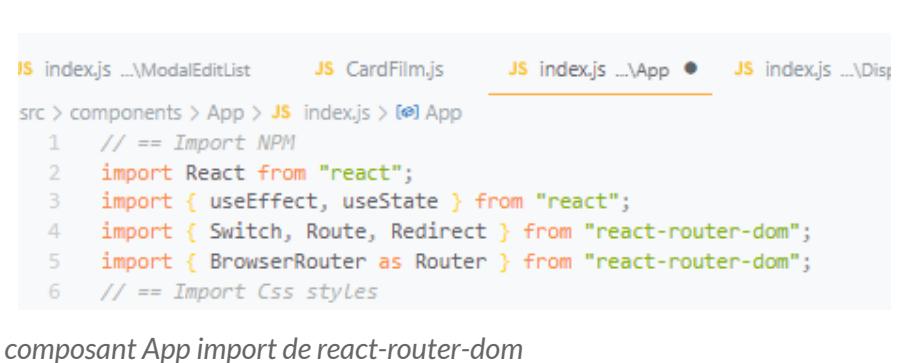
Le **switch** assure que *seule la première route à matcher sera affichée*.

Le **redirect** va nous permettre de *faire des redirections*

Certaines pages de notre application ne sont accessibles qu'aux **utilisateurs connectés**.

Pour se faire, je commence par **créer un state pour l'état de connexion**. (ligne 24)

Dans le hook **useEffect** je paramètre *une fonction chargée de récupérer l'objet "token"* dans le localStorage, en fonction du résultat, on va mettre à jour **notre state**.(ligne 28 et 30)



```
1 // == Import NPM
2 import React from "react";
3 import { useEffect, useState } from "react";
4 import { Switch, Route, Redirect } from "react-router-dom";
5 import { BrowserRouter as Router } from "react-router-dom";
6 // == Import Css styles
```

composant App import de react-router-dom



```
22
23 const App = () => [
24   let [isLoggedIn, setIsLoggedIn] = useState(false);
25
26   useEffect(() => {
27     const hasToken = !!localStorage.getItem("token");
28     if (hasToken) setIsLoggedIn(true);
29
30     return () => setIsLoggedIn(false);
31   }, []);
32 ]
```

UseEffect dans App

- **Routes avec Redirect**

Dans notre MVP certains composants **en fonction de l'état de connexion de l'utilisateur, redirigent vers un autre composant.**

ici sur l'url “/library” j'utilise le paramètre de react-router “exact” pour m'assurer que ce composant ne pourra être rendu ailleurs que sur cette url.

Puis je paramètre un affichage conditionnel avec l'opérateur ternaire “condition ? true : false”.

Grâce à cet opérateur **je teste la valeur de mon state.**

En fonction du résultat de cette opération l'utilisateur sera redirigé grâce au composant Redirect. (ligne 80 - 86).

Dans le cas où l'utilisateur n'est pas connecté (ligne 84) en props de ce composant je passe l'état de mon state ainsi que la fonction permettant d'agir sur ce state.

Ces props seront utilisés dans le composant.

```
/* Routes avec redirect vers la page connexion si utilisateur NON connecté */
<Route exact path="/library">
  {isLoggedIn ? (
    <Redirect to="/library" />
  ) : (
    <SignIn setIsLoggedIn={setIsLoggedIn} isLoggedIn={isLoggedIn} />
  )}
</Route>
<Route exact path="/my-profile">
  {isLoggedIn ? (
    <Redirect to="/my-profile" />
  ) : (
    <SignIn setIsLoggedIn={setIsLoggedIn} isLoggedIn={isLoggedIn} />
  )}
</Route>
/* Renvoie vers une page d'erreur 404 si aucunes des routes ne corresponds */
<Route>
  <Error />
</Route>
</Switch>
<Footer />
</Router>
```

Composant App, routes avec Redirect

- **Routes utilisateur connecté/déconnecté**

Une fois ces redirections mises en place j'ai pu mettre en place les autres routes de notre router.

Là aussi je vais utiliser **un affichage conditionnel grâce à l'opérateur logique ‘&&** je teste la valeur du state. (ligne 41)

En fonction du résultat de cette opération l'utilisateur pourra accéder aux routes définis dans notre MVP, pour son état de connexion.

Une fois les routes mises en place, j'ajoute les composants qui seront rendus sur chaque page de notre application .

J'ajoute notre composant Header (ligne 37) sans l'inclure dans la switch du router et je ne défini pas de route pour le rendu de ce composant.

Je fait de même avec le composant Footer (ligne 90)

Finalement, je paramètre le composant Error (ligne 87).

A l'intérieur de notre Switch et en dernière position dans notre router ceci afin d'éviter qu'il remplace les autres routes dans notre arborescence.

Si aucune des autres routes ne peut être rendu par react, c'est ce composant qui le sera.

```

34
35   return (
36     <div className="App">
37       <Router>
38         <Header isLoggedIn={isLoggedIn} setIsLoggedIn={setIsLoggedIn}>
39           <Switch>
40             /* Routes utilisateur connecté */

41             {isLoggedIn && (
42               <>
43                 <Route path="/my-profile" exact >
44                   <User setIsLoggedIn={setIsLoggedIn} />
45                 </Route>
46                 <Route path="/library" exact component={UserLibrary} />
47                 <Route path="/" exact component={Home} />
48                 <Route path="/sign-in" exact component={SignIn} />
49                 <Route path="/sign-up" exact component={SignUp} />
50                 <Route path="/movie/:id" exact component={MovieDetail} />
51                 <Route path="/search" exact component={Search} />
52                 <Route path="/reset-password" exact component={ResetPassword} />
53                 <Route path="/team" exact component={Team} />
54                 <Route path="/general-conditions" exact component={GeneralConditions} />
55               </>
56             )
57           /* Routes pour utilisateurs NON connecté */
58           <Route path="/" exact component={Home} />
59           <Route path="/sign-in" exact>
60             <SignIn setIsLoggedIn={setIsLoggedIn} isLoggedIn={isLoggedIn} />
61           </Route>
62           <Route path="/sign-up" exact component={SignUp} />
63           <Route path="/movie/:id" exact component={MovieDetail} />
64           <Route path="/search" exact component={Search} />
65           <Route path="/team" exact component={Team} />
66           <Route path="/general-conditions" exact component={GeneralConditions} />
67         </Switch>
68       </Router>
69     </div>
70   );
71
72   export default App;
73

```

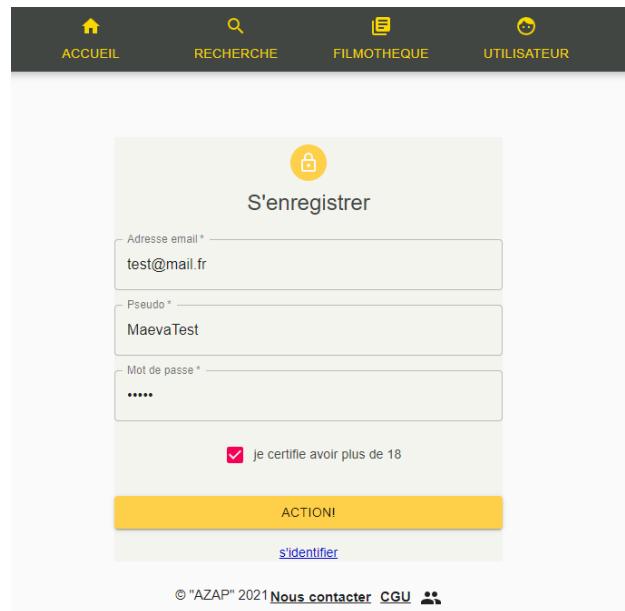
Composant App - Composant Error et routes utilisateur connecté et non-connecté

VII. Jeu d'essai :

Pour réaliser mon jeu d'essai j'ai choisi de tester la fonction SignIn de notre application.

1-Données en entrée : C'est l'action réalisée par l'utilisateur.

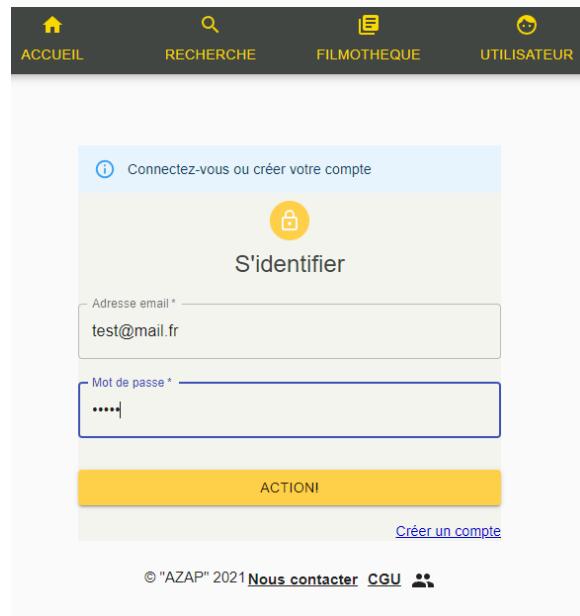
Ici notre utilisateur va créer un compte au moyen du composant SignIn



The screenshot shows the registration form. It has fields for 'Adresse email *' (test@mail.fr), 'Pseudo *' (MaevaTest), and 'Mot de passe *' (****). There is a checkbox for 'je certifie avoir plus de 18' (I certify being over 18) which is checked. At the bottom is a yellow 'ACTION!' button.

2-Données attendues : C'est comment l'application est censée réagir à l'action utilisateur.

Ici à la soumission du formulaire l'utilisateur est bien redirigé vers le composant SignUp afin de se logger sur l'application.

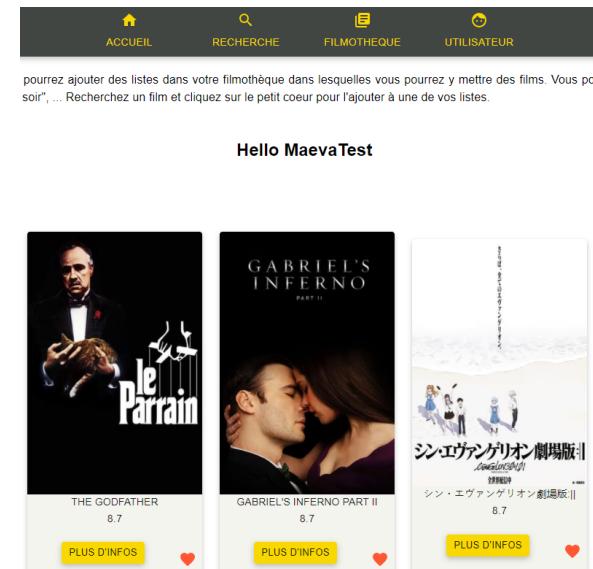


The screenshot shows the login form. It has fields for 'Adresse email *' (test@mail.fr) and 'Mot de passe *' (****). The password field is highlighted with a blue border. At the bottom is a yellow 'ACTION!' button. Below the button is a link 'Créer un compte'.

3-Données obtenues : C'est la réaction réelle de l'application

Ici, une fois notre utilisateur loggé, celui-ci est redirigé vers la page Home de notre application.

Sur notre page Home un affichage conditionnel permet de personnaliser le rendu de notre application pour l'utilisateur.



4-Résultats : Si la donnée attendue est identique à la donnée obtenue, alors le test est réussi.

Pour le jeu d'essai réalisé ici, un token unique a bien été créé pour notre utilisateur "MaevaTest".

Grâce à ce token l'application à les moyens de garantir une authentification sécurisé et unique.

Application	
Key	Value
token	eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJ1c2Vyljp7lmldjo4MCwiZW1haWwiOiJ0ZXN0QG1ha...
user	{"id":80,"email":"test@mail.fr","username":"MaevaTest","adult":true,"role":"member"}

VIII. Veille sur les vulnérabilités de sécurités

Cas n°1 : Cookies vs Token JWT

Lors de la réalisation du cahier des charges, il a déjà fallu se renseigner sur le moyen utilisé pour l'authentification d'un utilisateur, avec deux solutions pour le stockage du token JWT : dans un cookie ou dans le navigateur (localStorage).

Tous deux possèdent des avantages et inconvénients qu'il a fallu départager pour procéder à l'implémentation de l'un des deux.

Avantages/inconvénients

• Cookies - avantages

- L'utilisation de cookies permet à notre application d'être stateful.
- les cookies sont plus petits et donc plus facile à stocker dans le navigateur.
- les cookies peuvent "HTTP-only" ce qui signifie qu'ils peuvent être impossible à lire côté client, cela améliore la protection contre les attaques XSS.

• Cookies - inconvénients

- le cookie étant envoyé automatiquement à chaque requête cela le rend vulnérable aux attaques de type CSRF
- Les Cookies ne sont pas appropriés pour les API, il n'est pas possible d'envoyer un cookie depuis un domaine vers un autre.
- Le client devra envoyer un cookie à chaque requête vers le serveur, même pour des accès non sécurisé.

• Token - avantages

- L'authentification via token est stateless c'est-à-dire que le serveur n'a pas besoin de stocker le token. Le serveur crée, signe et vérifie le token.
- L'authentification via token est plus approprié pour les API avec différents domaines
- Les données contenues dans le token peuvent être de toute sorte, ce qui rend les tokens plus flexibles

• Token - inconvénients

- Les tokens sont plus vulnérables aux attaques XSS côté clients.
- Le stockage des tokens côté client est plus problématique:
Si le token est stocké dans le cookie et que celui-ci a une taille importante, le cookie sera moins performant.
Stocké dans le sessionStorage le token sera effacé à la fermeture du navigateur, et dans le localStorage le token sera lié à un domaine spécifique.

Verdict

Nous avons choisi de partir sur l'implémentation du token dans le localStorage car suffisant pour la sécurité de notre application au vu de sa simplicité.

Nous avons eu également moins de mal à mettre en place le token JWT que les cookies.

Idéalement, il faudrait deux tokens : un token d'accès et un token de rafraîchissement afin de ne stocker ces tokens que le temps de la session, et permettre l'accès à celui de rafraîchissement uniquement par le serveur.

Cas n°2 : Sensibilité et chiffrement du mot de passe

- **Sensibilisation**

Un mot de passe doit avoir un niveau minimum de complexité pour ne pas être facilement deviné à des fins malveillantes. Pour se faire, on avertit généralement l'utilisateur si la complexité n'est pas suffisante selon les bonnes pratiques Opquast.

Dans notre application côté front, nous avons créé une **Regex**, celle-ci est utilisée dans notre **composant SignIn**.

Ce composant est notre formulaire d'inscription, pour tester les règles de sécurité requise à savoir :

- **Huits caractères minimum,**
- **au moins une lettre majuscule**
- **une lettre minuscule**
- **un chiffre**

Si la condition n'est pas validée, alors une erreur sera affichée à la soumission du formulaire



```
JS index.js M JS regex.js U X
src > components > SignUp > JS regex.js > ...
1  export const validPassword = new RegExp('^(?=.*?[A-Za-z])(?=.*?[0-9]).{6,}$');

81 const handleSubmit = async (e) => {
82   e.preventDefault();
83
84   if(regex.test(validatedPassword)){
85     const { email, username, password } = e.target.elements;
86
87     let details = {
88       email: email.value,
89       username: username.value,
90       password: password.value,
91     };
92
93     if (checked) {
94       details.adult = true;
95     } else {
96       details.adult = false;
97     }
98
99     try {
100       const response = await axios.post(
101         "https://projet-azap-heroku.herokuapp.com/v1/signup",
102         details
103       );
104
105       if (response.status === 201)
106         setMessageStatus("L'utilisateur a bien été créé !");
107         history.push('/sign-in',{ message: messageStatus });
108     } catch (error) {
109       if (error.response.status === 409)
110         setMessageWarning("Cet email existe déjà !");
111
112       if (error.response.status === 422)
113         setMessageWarning("Le format d'email n'est pas valide !");
114
115       /* Affichage du message warning durant 7 secondes */
116       setTimeout(() => {
117         setMessageWarning()
118       }, 7000);
119     }
120   } else{
121     setMessageWarning("Le mot de passe doit contenir: Huit caractères minimum, au moins une lettre majuscule, une lettre minuscule et un chiffre");
122   }
}
```

- Chiffrement

Coté back on va **chiffrer les mots de passe** grâce au package **bcrypt**.

Hasher les mots de passe représente la meilleure méthode puisque **non-réversible**, cette méthode permet de créer une empreinte de la chaîne fournie (le mot de passe) et de stocker cette empreinte en base, cette méthode est donc la plus recommandée.

```
26 // 3. hashage du mot de passe
27 let salt = 10
28 const encryptedPassword = bcrypt.hashSync(password, salt);
29 req.body.password = encryptedPassword;
30
31
```

XI. Résolution de problème: Recherche et traduction

A titre personnel, lors de l'élaboration de ce projet et notamment lors de l'intégration de la page Filmothèque j'ai pu me rendre compte que je n'avait pas complètement saisi le concept d'effet de bord, en particulier avec le Hook useEffect de React.

Chose problématique étant donné l'importance de ce hook dans ce composant.

J'ai commencé par faire une recherche en explicitant ma problématique en anglais : react render component when state changes with useEffect

Je suis alors tombé sur le blog d'un développeur senior/coach à destination des développeurs, et d'un article concernant ma problématique.

A Simple Explanation of React.useEffect()

them right into your inbox.

Enter your email

Subscribe

Join 4569 other subscribers.

react **hook** **useeffect**

45 Comments

I am impressed by the expressiveness of React hooks. You can do so much by writing so little.

But the brevity of hooks has a price — they're relatively difficult to get started. Especially useEffect() — the hook that manages side-effects in functional React components.

In this post, you'll learn how and when to use useEffect() hook.

Table of Contents

- useEffect() is for side-effects
- Dependencies argument
- Component lifecycle
 - Component did mount
 - Component did update
- Side-effect cleanup
- useEffect() in practice
 - Fetching data
- Conclusion

About Dmitri Pavlutin

Software developer, tech writer and coach. My daily routine consists of (but not limited to) drinking coffee, coding, writing, coaching, overcoming boredom 😊.

[Email](#) [Twitter](#) [Facebook](#) [LinkedIn](#)

Contact me

1. ***useEffect() is for side-effect***

A functional React component uses props and/or state to calculate the output. If the functional component makes calculations that don't target the output value, then these calculations are named side-effects.

Examples of side-effects are fetch requests, manipulating DOM directly, using timer functions like setTimeout(), and more.

The component rendering and side-effect logic are independent. It would be a mistake to perform side-effects directly in the body of the component, which is primarily used to compute the output.

How often the component renders isn't something you can control — if React wants to render the component, you cannot stop it.

How to decouple rendering from the side-effect? Welcome useEffect() — the hook that runs side-effects independently of rendering.

1. ***useEffect() est utilisé pour les effets de bords***

Un composant fonction utilise des props, et/ou un state pour calculer un résultat. Si ces calculs ne concernent pas la valeur de sortie, ces calculs sont des effets de bords.

Exemples d'effets de bord, récupérer des requêtes, manipuler le DOM directement, utiliser des fonctions timer comme setTimeout() ...

Le rendu du composant et les effets de bord sont indépendants.

Ce serait une erreur d'effectuer des effets de bords directement dans le corps du composant, celui-ci est utilisé pour calculer le résultat.

La fréquence des rendu d'un composant n'est pas quelque chose que l'on peut contrôler, si React veut rendre le composant on ne peut pas l'arrêter.

Comment découpler le rendu du composant et les effets de bords? Bienvenue au hook useEffect qui permet d'exécuter les effets de bords indépendamment du rendu du composant.

useEffect() hook accepts 2 arguments: useEffect(callback[, dependencies]);

-Callback is the function containing the side-effect logic. callback is executed right after changes were being pushed to DOM.

-Dependencies is an optional array of dependencies.

useEffect() executes callback only if the dependencies have changed between renderings.

Put your side-effect logic into the callback function, then use the dependencies argument to control when you want the side-effect to run. That's the sole purpose of useEffect().

Le Hook useEffect accepte deux arguments, un callback et une dépendance.

-Le callback est la fonction qui contient l'effet de bord. Le callback est exécuté directement après que les changements aient été fait dans le DOM.

-La dépendance, est un tableau de dépendance optionnel.

useEffect() exécute le callback seulement si le tableau de dépendance a été modifié entre les rendus du composant.

Placer la logique de l'effet de bord dans le callback, puis utiliser le tableau de dépendance en argument pour contrôler l'exécution de l'effet de bord, c'est le but de useEffect.

2. Dependencies argument

dependencies argument of useEffect(callback, dependencies) lets you control when the side-effect runs. When dependencies are:

A) Not provided: the side-effect runs after every rendering.

B) An empty array []: the side-effect runs once after the initial rendering.

C) Has props or state values [prop1, prop2, ..., state1, state2]: the side-effect runs only when any dependency value changes.

Let's detail into the cases B) and C) since they're used often.

2. Dépendances

Les dépendances de useEffect (le callback et le tableau de dépendance) permettent de contrôler quand les effets de bords seront exécutés. Lorsque le tableau de dépendance est:

- A) Non fournis : L'effet de bord est exécuté après chaque rendu.
- B) Un tableau vide : l'effet de bord est exécuté une fois lors du rendu initial.
- C) A des props ou un state : l'effet de bord est exécuté seulement lorsque les valeurs changent.

Nous allons détailler les cas B et C, car ce sont les plus utilisés.

3. Component lifecycle

3.1 Component did mount

Use an empty dependencies array to invoke a side-effect once after component mounting:

useEffect(..., []) was supplied with an empty array as the dependencies argument. When configured in such a way, the useEffect() executes the callback just once, after initial mounting.

3. Cycle de vie des composants

3.1 Montage du composant

Utiliser un tableau de dépendance vide permet d'invoquer un effet de bord lors du montage du composant:

Lorsque useEffect() est configuré avec un tableau vide en dépendance; le callback est exécuté une fois après le montage initial du composant.

3.2 Component did update

Each time the side-effect uses props or state values, you must indicate these values as dependencies:

The useEffect(callback, [prop, state]) invokes the callback after the changes are being committed to DOM and if and only if any value in the dependencies array [prop, state] has changed.

Using the dependencies argument of useEffect() you control when to invoke the side-effect, independently from the rendering cycles of the component. Again, that's the essence of useEffect() hook.

3.2 Mise à jour du composant

A chaque fois qu'un effet de bord utilise des props, ou les valeurs d'un state, vous devez indiquer ses valeurs en tant que dépendances. useEffect utilise son callback après que les changement aient été transmis au DOM, si et seulement si les valeurs utilisées en tant que dépendance ont été modifiés.

Utiliser les dépendances de useEffect permet de contrôler quand les effets de bords seront exécutés, indépendamment de la méthode des cycles de vie de React.

C'est le principe du hook useEffect().

4. Side-effect cleanup

Some side-effects need cleanup: close a socket, clear timers.

If the callback of useEffect(callback, deps) returns a function, then useEffect() considers this as an effect cleanup:

Cleanup works the following way:

- A) After initial rendering, useEffect() invokes the callback having the side-effect. cleanup function is not invoked.*
- B) On later renderings, before invoking the next side-effect callback, useEffect() invokes the cleanup function from the previous side-effect execution (to clean up everything after the previous side-effect), then runs the current side-effect.*
- C) Finally, after unmounting the component, useEffect() invokes the cleanup function from the latest side-effect.*

4. Nettoyer les effets de bords

Certains effets de bords nécessitent d'être nettoyer: fermer une connexion, effacer un timer.

Si le callback de useEffect retourne une fonction, alors useEffect considère ce callback en tant qu'effet de nettoyage.

Voici la procédure:

- A) Après le rendu initial, useEffect appelle le callback pour produire l'effet de bord. La fonction de nettoyage n'est pas appelée.
- B) Dans les rendus de composant suivants, avant d'appeler le prochain effet de bord useEffect appelle la fonction de nettoyage a partir de l'effet de bord précédent et avant d'exécuter le suivant.
- C) Finalement, une fois le composant démonté useEffect() appelle la fonction de nettoyage.

5. Conclusion

useEffect(callback, dependencies) is the hook that manages the side-effects in functional components. callback argument is a function to put the side-effect logic. dependencies is a list of dependencies of your side-effect: being props or state values. useEffect(callback, dependencies) invokes the callback after initial mounting, and on later renderings, if any value inside dependencies has changed.

5. Conclusion

useEffect() est un hook qui permet de gérer les effets de bords dans les composants fonctions de React. La fonction callback de useEffect permet d'évaluer l'effet de bord. Les dépendances sont une liste des valeurs dont dépendent les effets de bords.

useEffect() appelle la fonction callback après au rendu initial du composant, et plus tard si les valeurs contenu dans les dépendances changent.

X. Conclusion

Pour conclure, je dirais que ce projet m'a fait prendre conscience des responsabilités que l'on peut avoir dans la réalisation d'un projet en équipe, ainsi que les difficultés à déceler les anomalies et surtout à appréhender les non-conformités qui en découlent.

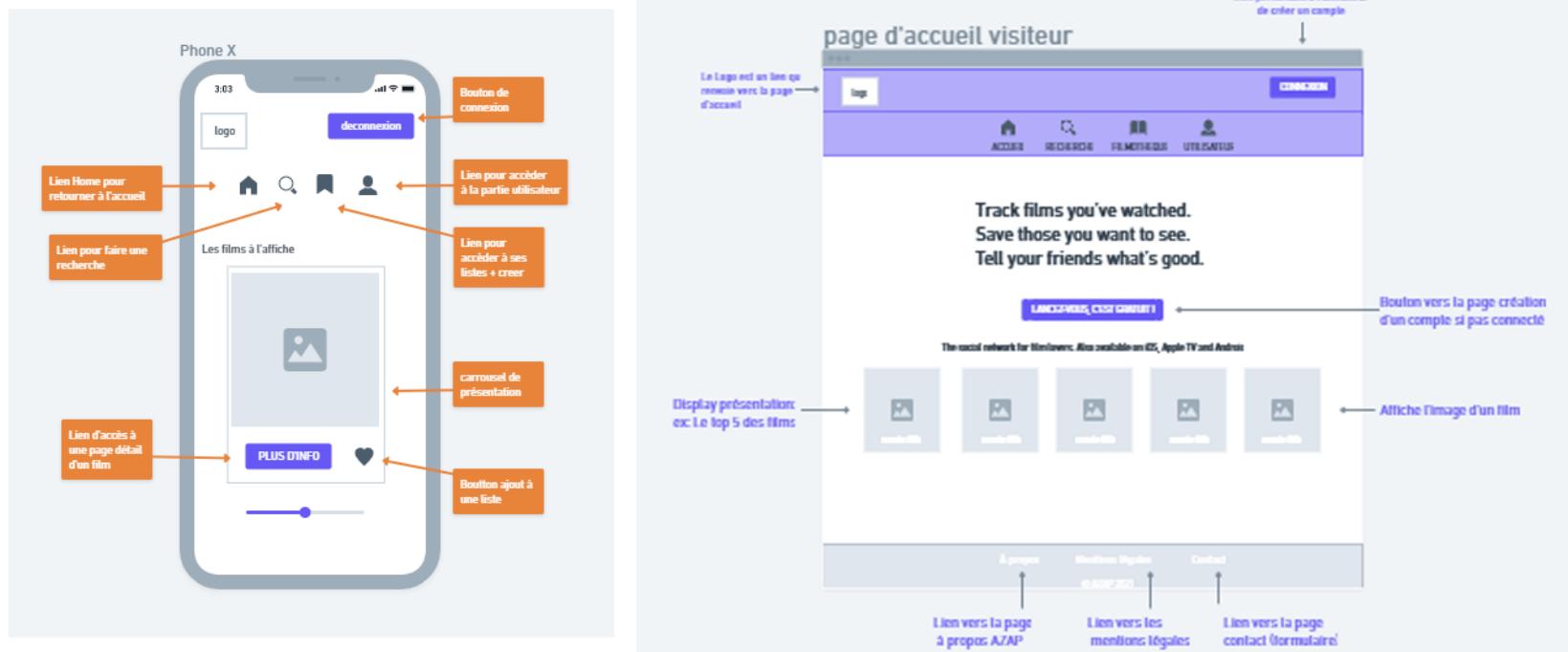
L'expérience acquise au fil de ma formation pour devenir développeur, mon implication personnelle et la recherche continue de connaissance m'ont été bénéfiques, et je pense avoir pris les bonnes décisions au cours des tâches qui m'étaient attribuées.

XI. Annexes

User Stories:

Users Stories								
ID	Thème	En tant que	J'ai besoin de	Afin de	Domaine	Sprint	Reste à faire au 26/07	Priorité du Sprint 2
1	Accueil	Visiteur	Accéder à un formulaire de création de compte	-Connaitre le but du site	Interface	1	ok	
2	Accueil	Visiteur	Pouvoir s'inscrire	-Découvrir les services	Interface	1	ok	
3	Accueil	Visiteur	Accéder à un formulaire de connexion pour s'identifier	-Découvrir les services	Interface	1	ok	
4	Accueil	Visiteur	Pouvoir s'identifier	-Découvrir les services	Interface	1	ok	
5	Page d'un film	Visiteur	Accéder à une page de récupération de mot de passe	-Découvrir les services	Interface	2		
6	Profil d'utilisateur	Visiteur	Accéder à la page d'accueil top 20 films	-Découvrir les services	Interface	1	ok	
7	Profil d'utilisateur	Visiteur	Pouvoir faire une recherche par titre, acteurs, réalisateur (barre de recherche)	-Découvrir les services	Interface	1	ok	
8	Profil d'utilisateur	Visiteur	Pouvoir faire une recherche détaillée d'un film	-Découvrir les services	Interface	1	ok	
9	Profil d'utilisateur	Visiteur	Visualiser les résultats de la recherche détaillée	-Découvrir les services	Interface	1	ok	
10	Profil d'utilisateur	Visiteur	Accéder à une page des détails d'un film	-Découvrir les services	Interface	1	ok	
11	Accueil	Visiteur	Consulter la liste des autres membres qui sont en public	-Découvrir les services	Sécurité	X		
12	Accueil	Utilisateur (membre)	Pouvoir me déconnecter	-Me déconnecter	Interface	2	ok	
13	Accueil	Utilisateur (membre)	Accéder à une page profil	-Gérer mes informations personnels	Interface	2	ok	
14	Profil d'utilisateur	Utilisateur (membre)	Pouvoir modifier mon mot de passe	-Gérer mes informations personnels	Sécurité	2	ok	
15	Profil d'utilisateur	Utilisateur (membre)	Accéder à une page de changement de mot de passe	-Gérer mes informations personnels	Sécurité	2	ok	
16	Profil d'utilisateur	Utilisateur (membre)	Pouvoir modifier mon adresse mail	-Gérer mes informations personnels	Sécurité	2	ok	
17	Profil d'utilisateur	Utilisateur (membre)	Pouvoir modifier mon user name	-Gérer mes informations personnels	Sécurité	2	ok	
18	Profil utilisateur	Utilisateur (membre)	Demander la récupération de mes données personnelles (RGPD) (cf US 43)	-Gérer mes informations personnels	Sécurité	2	ok	
19	Profil utilisateur	Utilisateur (membre)	Pouvoir supprimer mon compte	-Gérer mes informations personnels	Sécurité	2	ok	
20	Profil d'utilisateur	Utilisateur (membre)	Pouvoir faire une recherche détaillée de film dans sa bibliothèque (genre + durée, acteurs)	-Utiliser les services	Interface	X		
21	Profil d'utilisateur	Utilisateur (membre)	Visualiser les résultats de la recherche détaillée effectué dans sa bibliothèque	-Utiliser les services	Interface	X		
22	Profil d'utilisateur	Utilisateur (membre)	Créer, modifier, supprimer une liste	-Utiliser les services	Interface	2	ok	2
23	Profil d'utilisateur	Utilisateur (membre)	Visualiser mes bibliothèques	-Utiliser les services	Interface	2	ok - a tester	
24	Profil d'utilisateur	Utilisateur (membre)	Visualiser les films d'une bibliothèque	-Utiliser les services	Interface	2	ok - a tester	
25	Profil utilisateur	Utilisateur (membre)	Ajouter, supprimer un film à une ou plusieurs liste(s)	-Utiliser les services	Interface	2	ok	2
26	Profil utilisateur	Utilisateur (membre)	Sélectionner aléatoirement un film dans mes bibliothèque	-Utiliser les services	Interface	2		
27	Profil d'utilisateur	Utilisateur (membre)	Accéder une page des détails d'un film contenu dans ma bibliothèque	-Découvrir les services	Interface	2	ok - a tester	
28	Profil d'utilisateur	Utilisateur (membre)	Identifier visuellement les films que j'ai déjà vu	-Découvrir les services	Interface	X		
29	Profil utilisateur	Utilisateur (membre)	Noter ou dénoter un film	-Utiliser les services	Interface	X		
30	Profil utilisateur	Utilisateur (membre)	Mettre ma liste de film en public	-Utiliser les services	Interface	X		

WireFrames : page Home



WireFrames : page Filmothèque

Composant Library

page bibliothèques

WireFrames : page Film

The image displays two wireframe prototypes for a movie page, labeled 'Phone X' and 'page affichage résultat d'un film'.

Phone X Wireframe:

- Header:** A search icon (magnifying glass) and a placeholder 'placeholder'.
- Top Bar:** 'Phone X' (device name), a logo, a 'deconnexion' button, and a 'Dark mode' toggle switch.
- Content Area:**
 - Rating:** A 5-star rating icon with the text 'Noter un film'.
 - Title:** 'Titre'.
 - Summary:** A Latin quote: 'Si ergo illa tantum fastidium compescere contra naturalem usum habili, quem habelis vestra potestate, non aliud quam aversantur incurare.'
 - Buttons:** Home icon, search icon, star icon, bookmark icon, and user icon.
 - Callouts:**
 - 'les films que j'aimerais voir: Au clique : Ajouter à la wishlist'
 - 'les films que je possèdent: Au clique : Ajouter à la liste des films que je possèdent'

page affichage résultat d'un film:

- Header:** 'AZAP' (with a toggle switch), 'DECONNEXION', 'FILMOTHÉQUE', 'MÉMORIE', and a 'placeholder' button.
- Search Bar:** 'PARCOURIR PAR' (Year, Rating, Popularity, Genre, Duration), 'TROUVER UN FILM' (Search bar with placeholder 'Titre du film'), and a 'TROUVER' button.
- Result Section:** 'Résultat de la recherche pour "Toto et les oiseaux"'. It shows the movie poster for 'Toto et les oiseaux' (2019, directed by Aligorus).
- Movie Details:**
 - Title:** 'Toto et les oiseaux'.
 - Add to Favorites:** 'Ajouter à ma liste de favoris' (with a heart icon).
 - Description:** 'TOUT CE QUE VOUS AVEZ BESOIN EST D'UNE PISTE TUÉE.' (Summary: After being forced to work for a criminal mastermind, Toto, a young chauffeur in love, finds himself caught up in a robbery gone wrong.)
 - Member Review:** 'Avis de Karstlin Oppquat' (Rating: 4.5 stars, 456 reviews).
 - Comments:** '3456 likes' (with a heart icon), '456k', '105k', and '329k' (shares, comments, and likes).
 - Footer:** 'AZAP', 'Mentions légales', and 'Contact'.

Annotations:

- Titre du film:** Points to the movie title 'Toto et les oiseaux'.
- Ajout du film au favoris:** Points to the 'Ajouter à ma liste de favoris' button.
- Descriptif d'un film:** Points to the movie description.
- Avis d'un membre:** Points to the member review section.
- Les jaimes du commentaire membre:** Points to the '3456 likes' section.
- Nombre de vues du film:** Points to the '456k' views count.
- Bouton partager avec le nombre de partages du film:** Points to the '105k' shares count.
- Les jaimes du film affiché:** Points to the '329k' likes count.
- À propos:** Points to the 'À propos' link in the footer.

Routes (endpoints)

Liste des routes				
ID	Méthode	Chemin	Description	Retour
1	GET	/topmovies	Affiche la page d'accueil générale	200 : JSON liste des films top20 500 : erreur serveur
3	POST	/login	Soumet le formulaire de connexion	ok : JSON {message: "Bienvenue <username> !"} error : status 500
5	POST	/signup	Soumet le formulaire d'inscription	ok : JSON object {newUser} status 200 error : status 500
6	GET	/logout	Déconnecte l'utilisateur	Redirect vers Homepage JSON object {message: "A la prochaine + user_name !"} JSON object {list of users}
8	GET	/admin/users	Affiche la liste des tous les utilisateurs	JSON object {message: "Le compte a bien été supprimé"}
9	DELETE	/admin/delete/{user_id}	Supprime le compte d'un user	JSON object {message: "Votre mot de passe a bien été modifié"} ou Redirect vers page reset password JSON object {message: "Une erreur s'est produite"}
11	POST	/password/forgot	Soumet le formulaire avec mon adresse mail	Envoyer un lien temporaire avec token vers email renseigné Redirect vers Homepage JSON object {message: "Un email a été à votre adresse"}
12	GET	/password/reset/:token	Affiche la page pour reset le mot de passe	JSON object {token} ou JSON object {message: "Le token n'est pas valide ou expiré"} Redirect page reset password
13	PUT	/password/reset/:token	Soumet le formulaire avec nouveau mot de passe	Redirect vers Homepage JSON object {message: "Votre mot de passe a bien été modifié"} ou Redirect vers page reset password JSON object {message: "Une erreur s'est produite"}
15	PUT	/account/password/reset	Soumet le formulaire avec nouveau mot de passe	Redirect vers Homepage JSON object {message: "Votre mot de passe a bien été modifié"} ou Redirect vers page reset password JSON object {message: "Une erreur s'est produite"}
16	POST	/search/{search}	Soumet le formulaire de recherche avec filtre	200 : JSON liste de films 500 : erreur serveur
17	POST	/search	Soumet le formulaire de recherche depuis la barre de recherche	JSON object {list of movies} Redirect vers une page contenant les résultats de la recherche
18	GET	/movie/{movie_id}	Affiche les détails d'un film	200 : JSON liste de film avec leur détails 500 : erreur serveur
19	GET	/account	Affiche du profil utilisateur	JSON object {user}
20	DELETE	/account/delete	Supprime le compte utilisateur	JSON object {message: "Votre compte a bien été supprimé, et vos informations personnelles effacées"} Redirect vers page d'accueil
22	PUT	/account/email/reset	Soumet le formulaire avec nouvel email	Redirect vers Account JSON object
22c	PUT	/account/username/reset	Soumet le formulaire avec nouveau pseudo	Redirect vers Account JSON object
25	GET	/libraries	Affiche les librairies de l'utilisateur	JSON object {user's librairies} Redirect to /libraries
27	POST	/libraries/add	Soumet le formulaire ajout de la bibliothèque	JSON object {user's librairies} Redirect to /libraries
28	PUT	/libraries/{library_id}/edit	Soumet le formulaire de modification d'une bibliothèque	JSON object {user's librairies} Redirect to /libraries
29	DELETE	/libraries/{library_id}/delete	Soumet le formulaire de suppression d'une bibliothèque	JSON object {user's librairies} Redirect to /libraries
30	GET	/libraries/{library_id}/search	Affiche une bibliothèque	JSON object {library}
32	POST	/libraries/{library_id}/search	Soumet le formulaire de recherche avec filtre (parmi la bibliothèque)	JSON object {list of movies} Redirect vers une page contenant les résultats de la recherche
33	PUT	/movie/{movie_id}/add/{library_id}	Ajoute un film à une bibliothèque	JSON object {message: "Le film a bien été ajouté à <name library>"}
34	DELETE	/libraries/{library_id}/{movie_id}/delete	Supprime un film d'une bibliothèque	JSON object {list of movie in library}
35	GET	/libraries/{library_id}/random	Affiche une page avec le résultat de la recherche random	JSON object {movie}
36	GET	/libraries/{library_id}/{movie_id}/details	Affiche les détails d'un film contenu dans une bibliothèque	JSON object {movie}
37	PUT	/movies/{movie_id}/seen	Ajoute un indicateur visuel - le film a été vu par l'utilisateur	JSON object {list of movies}
38	GET	/libraries/{library_id}	Affiche le contenu d'une bibliothèque	JSON object {list of movies}
39	GET	/genres	Affiche la liste des genres de films disponibles	JSON object {list of movie genres}

Dictionnaire de données :

Nom symbolique	Description (rôle)	Domaine (Postgres)	Types (Postgres)	Commentaires	Contraintes, règles de calcul
Library					
id	Identifiant unique	N (Entier)	INT		Automatique
name	Nom de la library créer par l'utilisateur	Text	VARCHAR(50)		Obligatoire
user_id	L'identification à la relation entre la library et l'utilisateur	N (Entier)	INT		
Library_has_Movie					
id_themoviedb	identifiant unique	N (Entier)	INT	Identifiant du film dans the movie database	
library_id	identifiant unique	N (Entier)	INT	Identifiant de la library créer par l'utilisateur	
Movie					
id_moviethedb	Identifiant unique propre à l'API "themoviedb"	N (Entier)	INT		
seen	Est-ce que j'ai vu ce film	Boolean	BOOLEAN		
User					
id	Identifiant unique	N (Entier)	INT		Automatique
email	Adresse email de l'utilisateur	Create Domain email ?	email (extension citext) ?	Format: contient @	Obligatoire
pseudo	Identifiant unique de l'utilisateur	Text	VARCHAR(50)		Obligatoire
password	Mot de passe du compte de l'utilisateur	Hash	VARCHAR(100)	Mot de passe sera hashé	Obligatoire
adult	Est-ce que l'utilisateur est majeur	Boolean	BOOLEAN		Obligatoire
role	Visiteur, membre, admin	Text	VARCHAR(50)		Automatique