

# DOSSIER DE PROJET

En vue de l'obtention du titre professionnel Développeur Web  
et Web Mobile  
Certification de niveau III

## Projet Picky



**Nina Petit**

Centre de formation: O'Clock  
Promo Robin 2021

# Sommaire

<b>I. Introduction</b>	<b>4</b>
<b>II. Remerciements</b>	<b>4</b>
<b>III. Liste des compétences couvertes par le projet</b>	<b>5</b>
A. Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité	5
B. Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité	5
<b>IV. Résumé du projet</b>	<b>7</b>
<b>V. Cahier des charges</b>	<b>8</b>
A. Conceptualisation de l'application	8
1. Minimum Viable Product	8
2. Évolutions envisagées	8
B. Mise en place de l'application	8
1. Wireframes	8
2. Charte graphique	9
3. Utilisateurs et user stories	10
4. Arborescence de l'application	12
<b>VI. Spécifications techniques</b>	<b>14</b>
A. Versionning	14
B. Workflow et technologies	14
C. Sécurité	15
D. Autres services	15
E. Base de données	16
<b>VII. Gestion du projet</b>	<b>17</b>
A. Présentation de l'équipe	17
B. Organisation du travail	17
1. Sprints	17
2. Outils collaboratifs	18

<b>VIII. Production</b>	<b>19</b>
A. Réalisations personnelles	19
1. Sprint 0: Barre de recherche	19
2. Sprint 1: Composant Card et Picky Find	23
3. Sprint 2: Résultats de Picky Mood	31
4. Sprint 3: Picky Wish (watchlist)	37
B. Jeu d'essai	45
<b>IX. Recherche à partir d'un site anglophone, extrait et traduction</b>	<b>48</b>
<b>X. Veille sur la sécurité</b>	<b>50</b>
<b>XI. Difficultés rencontrées</b>	<b>54</b>
<b>XII. Conclusion</b>	<b>56</b>
<b>XIII. Annexes</b>	<b>57</b>
A. Wireframes mobiles	57
B. Wireframes desktop	60

## I. Introduction

Grâce à la formation O'Clock en développement web fullstack JavaScript, j'ai acquis beaucoup de nouvelles connaissances et compétences sur le métier de développeur.se, dans un premier temps durant le socle, qui nous a formé aussi bien sur le back que sur le front, pendant 4 mois, puis en suivant la spécialisation front React pendant un mois. J'ai enfin suivi la spécialisation back Data en replay après la fin de la formation pour compléter mes connaissances.

Ce parcours passionnant m'a permis de développer après le socle et la spécialisation React, durant un mois, avec une équipe de quatre autres personnes, une application web, que je vais ici vous présenter.

Ce premier projet en équipe nous a permis d'approfondir les connaissances que nous avons acquises durant la formation, d'apprendre de nouvelles choses par nous-même, mais aussi et surtout nous a mis en situation professionnelle et, en nous faisant utiliser nos nouvelles compétences sur un projet réel, a démarré notre transition du statut d'apprenant.e à celui de développeur.se web.

## II. Remerciements

Je tiens tout d'abord à remercier toute l'équipe d'O'Clock, pour leur implication et leur pédagogie, qui ont fait de cette formation le meilleur choix que j'aurais pu faire pour découvrir et apprendre le métier de développeur.se web.

Je remercie également toute la promotion Robin pour l'entraide et la bonne humeur qui ont rendu ces plusieurs mois de formation plus simples et agréables.

Je remercie aussi mes coéquipiers et coéquipières avec qui j'ai eu un grand plaisir à travailler et à échanger durant ce mois de projet.

Ces six mois à O'Clock marquent un tournant dans ma vie et m'ont fait découvrir le métier qui me passionne, et cette formation n'aurait pas été la même sans toutes ces personnes.

### III. Liste des compétences couvertes par le projet

#### A. Développer la partie front-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Maquetter une application:

Pour créer les **wireframes**, en **responsive design**, chaque personne du groupe a d'abord fait quelques essais de son côté, puis on a choisi ensemble l'idée qui nous plaisait le plus, et on l'a approfondie tous ensemble, en y ajoutant et modifiant des choses, pour avoir des wireframes mobiles et desktop qui plisaient à tout le monde. Ensuite, nous avons créé les user stories tous ensemble pour se mettre d'accord sur le fonctionnement de l'application.

- Réaliser une interface utilisateur web statique et adaptable:

À partir des wireframes, chacun a commencé à réaliser les différentes pages du site web avec **React** et **Sass**, en prenant en compte le besoin de responsive. On y ajoutait les choix de charte graphique qu'on faisait au fil du temps pour les couleurs, les polices et tous les autres choix concernant le design. Chacun modifiait au besoin les choix faits premièrement dans les wireframes en consultant les autres membres du groupe.

- Développer une interface utilisateur web dynamique:

Les interfaces utilisateurs ont été réalisées avec les bibliothèques Javascript **React** et **Redux**. React gère les données de l'application (le **state**) de façon dynamique. Quand ce state change, les éléments du DOM virtuel de React sont recalculés et affichés. Cette bibliothèque répond aux besoins d'interactivité et de manipulation dynamique sans besoin de recharger les pages. React-Redux permet d'avoir un **store**, qui a pour responsabilité de détenir le state. On peut ensuite créer des composants ayant accès au store et donc au state, que ce soit pour le lire ou pour déclencher des évolutions. En pouvant accéder au state n'importe où dans les composants, on élimine les intermédiaires, et l'application devient beaucoup plus évolutive.

#### B. Développer la partie back-end d'une application web ou web mobile en intégrant les recommandations de sécurité

- Créer une base de données

Pour créer la base de données, les données ont d'abord été recensées dans un **dictionnaire de données**, puis nous avons créé le **MCD** (Modèle Conceptuel de Données), pour représenter la structure des données et ses relations, et le **MLD** (Modèle Logique de Données), qui décrit la structure de données de manière textuelle. Ensuite, la base de données a été créée en **SQL**, grâce au SGBD (Système de Gestion de Base de Données)

**PostgreSQL**, avec **Sqitch**, qui est un système de migrations permettant de versionner une base de données.

- Développer les composants d'accès aux données

Pour accéder et interagir avec la base de données, nous avons utilisé une connexion **pool**. On a ensuite créé les requêtes dans des **dataMappers**, dans des fonctions qui sont ensuite utilisées dans les **contrôleurs**, qui eux-mêmes sont utilisés dans le **routeur**. En plus des données venant de notre propre base de données, nous avons utilisé une **API tierce**, celle de **BetaSeries**. Il y a donc certains endroits dans les dataMappers où on fait des requêtes vers cette API grâce à **fetch**, venant du module node-fetch.

- Développer la partie back-end d'une application web ou web mobile

La partie back-end de l'application a été créée avec **Node.js**, environnement d'exécution qui permet de créer des applications côté serveur en JavaScript, et utilise le framework **Express**.

Les requêtes vers la base de données ou vers l'API tierce BetaSeries sont initiées par un appel **axios** venant du front puis relayées par le système de routing d'express, pour ensuite créer des requêtes vers la base de données de Picky en utilisant PostgreSQL ou d'autres appels en fetch vers l'API tierce.

#### **IV. Résumé du projet**

Depuis maintenant quelques années, l'explosion du streaming est indéniable et les plateformes pour regarder nos programmes préférés se multiplient. Ce nouveau phénomène, par la quantité de possibilités qu'il crée, complique paradoxalement le fait de choisir ou de situer un programme.

Plusieurs problématiques émergent de ce phénomène: parfois, on veut regarder un programme précis, déjà défini, mais on ne se souvient plus, ou on ne sait pas, sur quelle(s) plateforme(s) on peut le trouver, ni s'il se trouve sur une des plateformes auxquelles on est abonné.e.

Il arrive également que l'on ne sache pas exactement ce que l'on veut regarder, mais qu'on veuille trouver un film ou une série qui est sur une de nos plateformes et qui corresponde à notre envie du moment (film, série, comédie, thriller...).

Enfin, il arrive que l'on n'ait même pas d'idée approximative du style du programme que l'on recherche, ou que l'on ait envie d'être surpris.e, en visionnant un des programmes auxquels nous avons accès sur nos plateformes.

C'est ici que Picky entre en jeu. Picky peut trouver la/les plateforme(s) qui diffusent le programme que vous voulez regarder, ou vous aide à trouver un programme disponible sur une de vos plateformes et qui corresponde à vos envies du moment, ou encore vous propose un programme aléatoire.

## V. Cahier des charges

### A. Conceptualisation de l'application

#### 1. Minimum Valuable Product:

Profil:

- Création d'un compte
- Modification des informations du compte
- Connexion via e-mail/mot de passe
- Suppression du compte

Picky Find (recherche par titre): Barre de recherche qui affiche les résultats des programmes qui correspondent à la recherche

Picky Mood (recherche par critères): Recherche d'un programme par critères:

- Type (film, séries)
- Genre (comédie, romantique, horreur, etc)
- Plateforme (Netflix, Amazon prime, Disney +, etc)

Picky Wish (watchlist): Ajout de programmes à une watchlist pour les retrouver plus tard

#### 2. Evolutions envisagées

Picky Mood:

- Seul ou accompagné
- Avec la famille/avec les amis

Picky Wish (watchlist):

- Pouvoir gérer le visionnage d'un programme (cocher pour "vu")

Autres:

- Programmes aléatoires (fait)
- Voir les détails d'un programme (acteurs, année, trailer...) (fait)
- Mode sombre (fait)
- Compteur de J'aime sur un programme
- Associer un profil utilisateur à des plateformes
- Connexion via les réseaux sociaux
- Envoi de mail de vérification

### B. Mise en place de l'application

#### 1. Wireframes

Nous avons créé les wireframes en **responsive design**, dans une approche **mobile first**, puis en travaillant sur l'affichage desktop. Il était important pour nous que les utilisateurs du site puissent profiter des fonctionnalités de Picky sur

un ordinateur, mais d'autant plus sur un téléphone. En effet, Picky est un outil utilisé de manière ponctuelle pour trouver un programme, il est donc nécessaire que son utilisation puisse se faire grâce à quelques clics sur un téléphone qui serait ensuite mis de côté.

(Voir annexes pour les wireframes)

## 2. Charte graphique

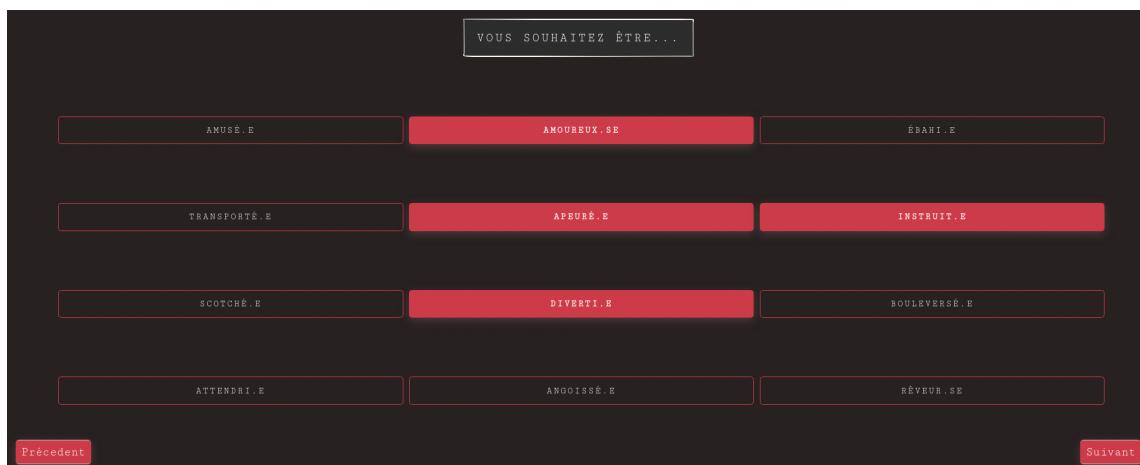
Les décisions concernant la charte graphique (couleurs, logos, typographie, etc) se sont faites au fur et à mesure, et en essayant de plaire à chacun.

Pour les couleurs du site, nous nous sommes décidés sur deux couleurs principales: rouge et jaune. Nous avons utilisé le jaune sur les principaux éléments, notamment les boutons, comme par exemple ici sur Picky Mood:



Nous avons utilisé le rouge pour le mode sombre, et d'autres éléments du site, comme la barre de recherche principale.

Mode sombre de Picky Mood:



Pour le logo, nous voulions quelque chose qui rappelle le cinéma, et le fait de regarder des films ou séries. Nous avions donc en tête une image de popcorn, et nous voulions que le nom de notre application, Picky, apparaisse dessus. Nous voulions également que nos couleurs principales, rouge et jaune, ressortent. Nous avons eu la chance de pouvoir faire appel à quelqu'un pour le réaliser:

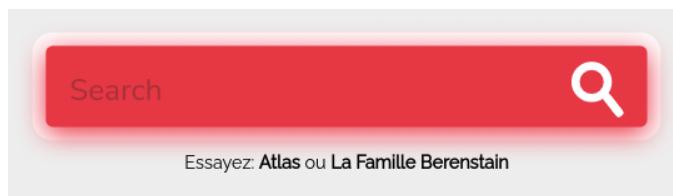


Pour la typologie, nous avons choisi deux polices différentes:

- une police principale, Cutive Mono:



- une autre police pour certains éléments, Raleway:



### 3. Utilisateurs et user stories

#### Public visé et rôles:

Cette application s'adresse à tous les sériephiles et cinéphiles francophones qui sont abonné.e.s à une ou plusieurs plateformes de streaming.

Dans ce projet, nous pouvons définir 2 types d'usagers:

- le visiteur (non inscrit)
- l'utilisateur (inscrit)

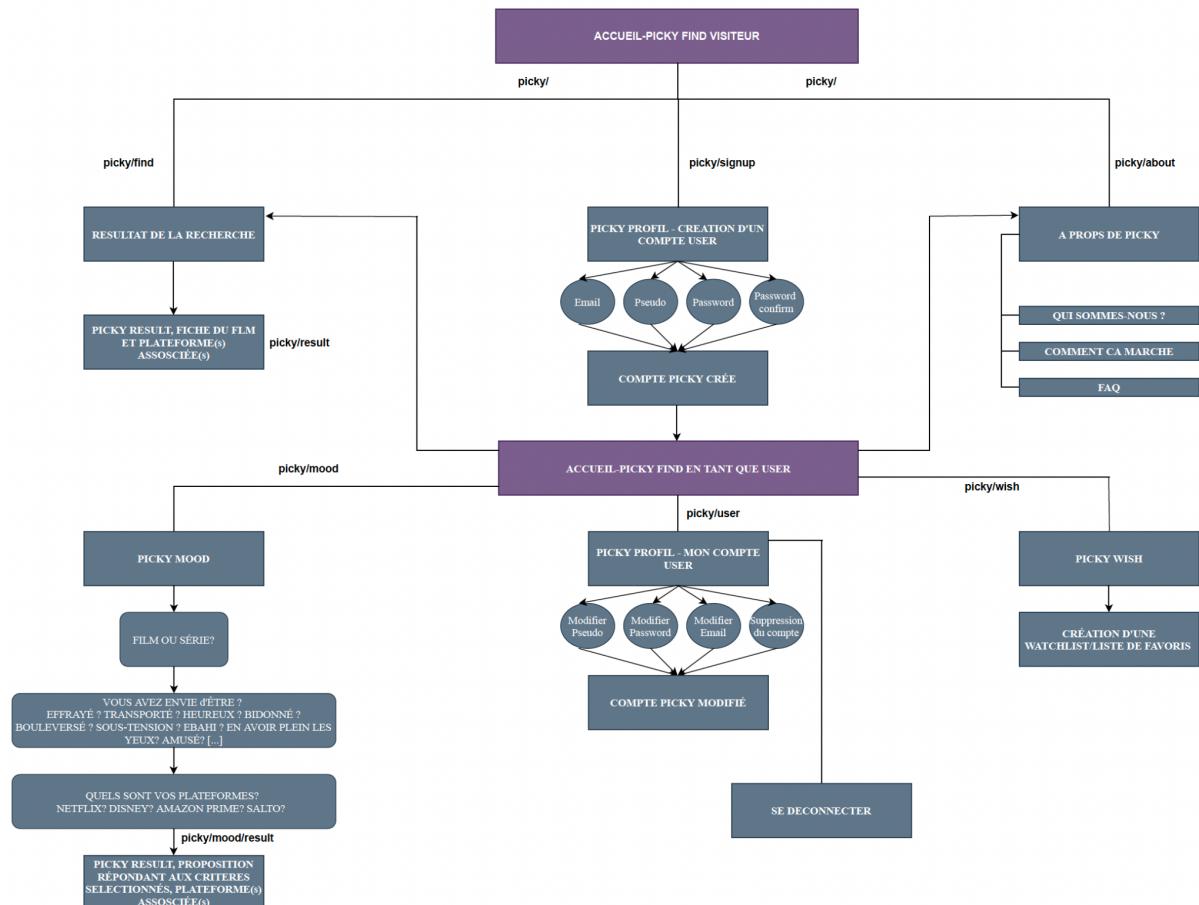
User Stories:

En tant que...	Je veux...	Afin de...	Version
visiteur	utiliser la barre de recherche (Picky Find)	trouver le programme que je recherche et pouvoir consulter ses informations	MVP
visiteur	accéder à la page à propos	en apprendre plus sur le site et sa création	MVP
visiteur	voir les détails d'un programme	en apprendre plus dessus, notamment pour pouvoir choisir et voir sur quelle(s) plateforme(s) il se trouve	2
visiteur	accéder aux réseaux sociaux de Picky	suivre Picky en dehors du site et voir son évolution, en apprendre plus dessus	2
visiteur	activer/désactiver le mode sombre	avoir le visuel qui me convient le mieux	2
visiteur	utiliser les boutons aléatoires (Picky Lucky)	trouver un programme au hasard	2
visiteur	créer un compte	avoir un compte et accéder à toutes les fonctionnalités	MVP
visiteur	me connecter	utiliser toutes les fonctionnalités	MVP
utilisateur	me déconnecter	mettre un terme à ma session	MVP
utilisateur	supprimer mon compte	faire respecter mes droits à l'oubli	MVP

utilisateur	modifier mon profil	mettre à jour mes informations	MVP
utilisateur	rechercher un programme par critère (Picky Mood)	trouver un programme qui correspond à mes attentes	MVP
utilisateur	ajouter un programme à ma watchlist	le retrouver plus tard	MVP
utilisateur	supprimer un programme de ma watchlist		MVP

#### 4. Arborescence de l'application

Avant de nous lancer dans la programmation de l'application, nous avons créé visuellement l'arborescence du site pour savoir dans quelle direction nous allions:



Nous avons ensuite commencé à programmer l'application, et petit à petit, certains changements ont été faits, notamment car il y a eu de nouvelles fonctionnalités et pages correspondantes ajoutées, et d'autres modifications que nous trouvions plus judicieuses. Voilà l'arborescence finale du site:

#### Header:

- Bouton pour ouvrir le menu burger
- Logo de Picky Mood (renvoie à la page d'accueil)
- Lien vers la page de connexion
- Lien vers la page d'inscription

#### Menu Burger:

- Lien vers Picky Find (page d'accueil)
- Lien vers Picky Mood
- Lien vers Picky Wish (watchlist)
- Lien vers Picky Lucky
- Lien vers la page à propos
- Bouton pour activer/désactiver le mode sombre
- Liens vers les réseaux sociaux (Facebook, Twitter, Instagram)

#### Page d'accueil

- Barre de recherche pour Picky Find
  - > Résultats
- Suggestions (un film et une série)
- Footer: Lien menant à Picky Mood/Lien menant à Picky Lucky

#### Picky Mood (si connecté):

Film ou série? -> Humeur? -> Plateformes? -> Résultats

#### Picky Lucky:

- Bouton "Be Picky" (série au hasard)
- Bouton "Be Lucky" (film au hasard)
  - > Page de résultats
    - Résultat
    - Footer: Lien pour revenir à "l'accueil" de Picky Lucky/Lien pour afficher un nouveau résultat

#### Picky Wish (watchlist) (si connecté):

- Cartes avec les programmes
- Footer: Lien vers Picky Find/Lien vers Picky Mood

#### Page à propos (modifiée):

- À propos de nous
- À propos de Picky
- Technologies utilisées
- Mentions légales

#### Page d'inscription:

- Formulaire d'inscription (pseudo, email, mot de passe, confirmation du mot de passe), bouton de validation
- Lien vers la page de connexion

Page de connexion:

- Formulaire de connexion (email, mot de passe), bouton de validation
- Lien vers la page d'inscription

Page profil (si connecté):

- Modification pseudo
- Modification email
- Modification mot de passe (avec confirmation)
- Suppression du compte

De plus, pour chaque carte de résultat, si on clique dessus, une modale s'ouvre avec les détails sur le programme.

Les liens vers les pages qui nécessitent d'être connecté renvoient à la page d'inscription si le visiteur n'est pas connecté.

## VI. Spécifications techniques

### A. Versionning

Nous avons travaillé sur un seul et même repo pour le front et le back, dans deux dossiers différents. Pour organiser le travail, nous avions différentes branches.

Premièrement, pour chaque fonctionnalité, chacun créait une nouvelle branche et codait dessus. Ensuite, nous venions merger les branches avec ces fonctionnalités sur une branche develop. Enfin, nous avions la branche main, sur laquelle on n'a mergé notre travail que vers la fin, quand c'était nécessaire, notamment pour mettre l'API du back en ligne.

### B. Workflow et technologies

Frontend:

- **React**: bibliothèque JavaScript pour la création d'application web monopage
- **Redux**: bibliothèque JavaScript de gestion d'état, pour centraliser le state React et pouvoir l'utiliser directement partout
- **React-router-dom**: package npm pour créer des routes dynamiques dans l'application
- **Axios**: package npm pour faire des requêtes (ici, au back)
- **Sass**: langage de script pour le CSS

Backend:

- **Node**: plateforme logicielle permettant l'exécution de JavaScript côté serveur
- **Express**: framework pour Node.js
- **Joi**: langage de description de schéma et validateur de données pour JavaScript
- **Sqitch**: système de migrations permettant de versionner une base de données
- **PostgreSQL**: système de gestion de base de données relationnelles
- **Cors**: package express pour activer CORS (Cross-Origin Resource Sharing) avec différentes options
- **Redis**: système de gestion de base de données, notamment utilisé pour la mise en cache
- **Swagger**: langage de description d'interface pour décrire des API RESTful
- **Bcrypt**: fonction de hachage pour les mots de passe
- **Node-fetch**: module pour faire des requêtes (ici, à l'API tierce Betaseries)

### C. Sécurité

Pour garantir la sécurité de l'application Picky et de la base de données, nous avons utilisé différentes technologies et techniques: **Joi**, pour vérifier les données avant d'aller les insérer dans la base de données ou d'y faire des changements, **Bcrypt**, pour hacher les mots de passes avant de les stocker, ou encore l'authentification avec **JSON Web Token** pour les parties du site qui nécessitent d'être connecté. Je reviens sur l'utilisation de ces technologies dans la partie Veille sur la sécurité.

### D. Autres services

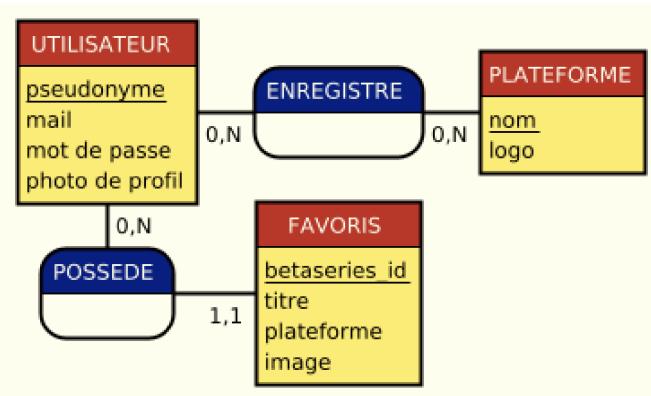
Pour créer Picky, nous avions besoin d'une base de données avec un grand nombre de programmes diffusés sur les plateformes de streaming en France, qui ait le plus d'informations possible sur chaque programme, et notamment la liste des plateformes sur lesquelles le programme est diffusé.

Pour cela, au lieu d'ajouter les programmes et toutes leurs informations dans notre base de données nous même, ce qui se serait certainement avéré beaucoup trop long, surtout vu la durée d'un mois que nous avions pour créer le projet, et insuffisant (nous n'aurions pas pu recenser plus que quelques dizaines, voire centaines, de programmes), nous avons eu recours à une API externe, qui est celle de **Betaseries**.

Grâce à l'API de Betaseries, nous avons pu faire des requêtes à chaque fois qu'un utilisateur cherchait un/des programmes via les différentes fonctionnalités de notre application, et ainsi offrir les meilleurs résultats possibles.

## E. Base de données

Modèle Conceptuel de Données (MCD):



Le **MCD** représente la structure des données et ses relations. Celui de Picky est assez petit car nous utilisons l'API tierce Betaseries pour récupérer tous les programmes et leurs informations, il n'y a donc pas besoin de stocker toutes ces informations dans notre base de données.

Il y a donc ici les utilisateurs, les programmes qu'ils ont mis en favori, et les plateformes qu'ils possèdent (en ce qui concerne les plateformes, on ne les utilise pas encore dans le fonctionnement de Picky, mais on a voulu les mettre pour faciliter l'évolution par la suite).

Modèle Logique de Données (MLD):

Le **MLD** découle du MCD. Il décrit la structure de données utilisée sans faire référence à un langage de programmation.

```

    UTILISATEUR (pseudonyme, mail, mot de passe, photo de profil);
    PLATEFORME (nom, logo);
    FAVORI (betaseries_id, titre, plateforme, image, utilisateur_id);
    UTILISATEUR_POSSEDE_PLATEFORME (utilisateur_id, plateforme_id);
  
```

## VII. Gestion du projet

### A. Présentation de l'équipe

L'équipe du projet Picky était composée de quatre développeurs et développeuses ayant suivi le même socle de formation de développeur web fullstack, puis un mois de spécialisation (React ou Data):

- Maeva, qui avait amené l'idée de Picky, et qui était donc **Product Owner**. Elle était là pour diriger la vision et la conception de notre site web. Elle a suivie la spécialisation Data.
- Sofiane, qui avait également fait la spécialisation Data, et qui a donc naturellement été assigné au rôle de **Lead dev Back**. Il devait prendre les décisions concernant le back.
- Délia, venant de la spécialisation React, qui avait le rôle de **Scrum Master**, et qui s'assurait donc du bon déroulement des réunions et de l'implication de chaque membre du groupe.
- Clément, qui était **référent technique**, et qui s'occupait alors de faire des recherches et de lire les documentations sur les technologies et outils que nous utilisions quand cela était nécessaire. Il a suivie la spécialisation React.
- Moi (Nina), ayant également suivi la spécialisation React, qui ai pris le rôle de **Lead dev Front**, et qui devais donc prendre les décisions et trancher sur ce qui concernait le front.

En dehors de ces rôles, nous avons partagé le rôle de **Git Master**, puisque nous ne voulions pas laisser la responsabilité de gérer Git à une seule personne, outil qu'aucun d'entre nous ne maîtrisait totalement. Ainsi, nous avons tous pu améliorer nos connaissances et compétences dans son utilisation.

Enfin, malgré les rôles qui ont été distribués pour prendre les décisions et améliorer la gestion de l'équipe, chacun participait à toutes les discussions sur la direction que nous voulions donner au projet, et tout le monde a pris part à toutes les phases de développement.

### B. Organisation du travail

#### 1. Sprints

Pour organiser le travail, nous nous sommes inspirés du cadre de développement **Scrum**, qui se base sur les principes agiles. Nous avons découpé notre projet en **sprints**. Un sprint durait une semaine. Dans cette méthode, avant chaque sprint, une réunion a lieu pour déterminer les tâches à faire durant le sprint, selon ce qui est prioritaire et faisable.

Pendant chaque sprint, nous faisions un **daily scrum**, c'est-à-dire une réunion quotidienne au début de la journée, durant 10 à 15 minutes. Cette réunion est menée par le Scrum Master (Délia), et sert à vérifier la progression de l'équipe, à ajuster les prochaines étapes, et à relever d'éventuels points bloquants, pour en trouver une solution après la réunion.

A la fin de chaque sprint, nous faisions un point sur le résultat du sprint, chacun présentait ce qu'il avait réalisé, et nous pouvions ensuite recommencer un nouveau sprint.

Au total, nous avons effectué 4 sprints:

- Sprint 0: conception du projet. Élaboration du cahier des charges, wireframes, user stories, conception de la base de données (dictionnaire de données, Modèle Conceptuel de Données, Modèle Logique de Données).
- Sprint 1: Développement du Minimum Viable Product
- Sprint 2: Continuité du développement du MVP, et début du développement de la version 2
- Sprint 3: Dernières améliorations et retouches, tests.

## 2. Outils collaboratifs

Pour se répartir les tâches que nous listions au début de chaque sprint, nous utilisions **Cithub Projects**. Grâce à ce kanban, nous pouvions séparer les tâches dans différentes colonnes: À faire (front), À faire (back), En cours, et Fait. Une fois les tâches énumérées, chacun pouvait se positionner sur l'une d'elles en écrivant son nom dessus, l'effectuer, puis passer à une autre tâche.

Pour échanger sur le projet, nous avons utilisé **Discord** et **Slack**. Sur Discord, nous faisions les réunions quotidiennes, puis nous restions toute la journée connectés. De cette manière, si quelqu'un avait une question à n'importe quel moment ou avait besoin d'aide, il suffisait d'activer son micro et de la poser. Il arrivait aussi parfois que nous nous séparions en plusieurs groupes pour pouvoir discuter de certaines parties du projet sans déranger les autres ou encore de coder à plusieurs lorsqu'il y avait des difficultés. Sur Discord, il est également possible de partager son écran, ce qui était très pratique pour voir le code de chacun. On utilisait également Slack car on peut plus facilement y envoyer des liens utiles, du code, ou des informations qu'on ne veut pas voir disparaître. Slack était aussi utile pour communiquer des informations aux personnes absentes de Discord à un moment donné.

Enfin, on utilisait **Google Drive** pour noter et retrouver toutes les informations importantes et les travaux réalisés notamment sur le cahier des charges, la charte graphique, les carnets de bord, etc.

Bien que les tâches étaient ainsi réparties entre chaque membre du groupe, chacun pouvait à tout moment demander de l'aide aux autres, et il arrivait alors que l'on travaille pendant un moment à deux ou trois sur une tâche, jusqu'à ce que la situation se débloque.

## VIII. Production

### A. Réalisations personnelles

#### 1. Sprint 0

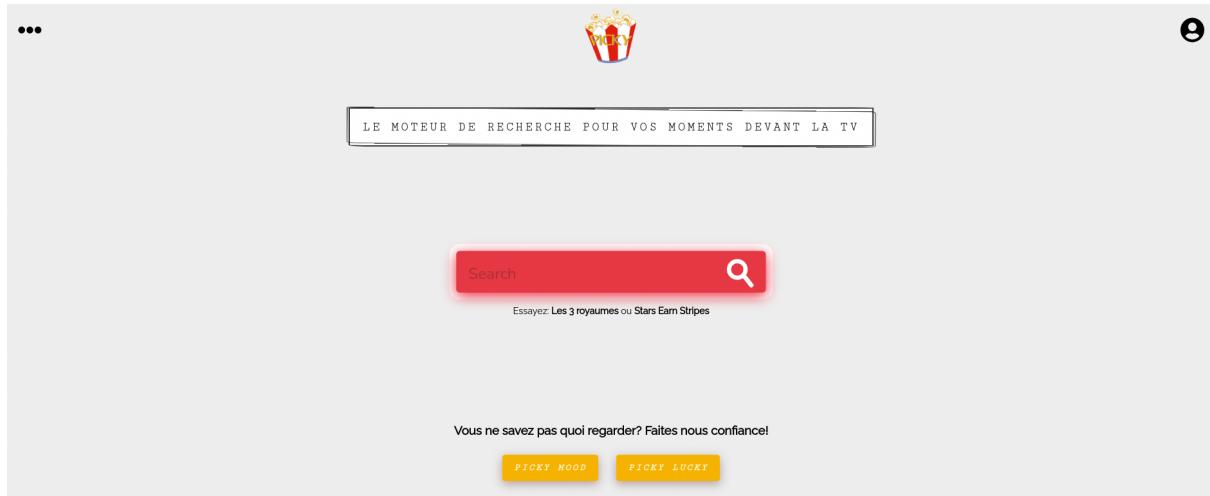
Mes principales réalisations:

04/06/2021	<ul style="list-style-type: none"><li>- Aide à la rédaction du cahier des charges</li><li>- Brainstorming sur le projet</li></ul>
05/06/2021	<ul style="list-style-type: none"><li>- Création d'un moodboard</li><li>- Début de wireframe version mobile</li></ul>
07/06/2021	<ul style="list-style-type: none"><li>- Création d'un wireframe commun version mobile</li><li>- Aide à la création du MCD, MLD, et dictionnaire de données</li></ul>
08/06/2021	<ul style="list-style-type: none"><li>- Aide à la mise en place des "To Do" du projet</li><li>- Aide à la rectification des routes</li><li>- Début de création de maquette</li></ul>
09/06/2021	<ul style="list-style-type: none"><li>- Mise en place de React</li><li>- Création de la search bar</li></ul>
10/06/2021	<ul style="list-style-type: none"><li>- Fin du composant search bar<ul style="list-style-type: none"><li>- Changement de CSS quand l'utilisateur écrit dans la searchbar et retour à la normale quand il efface</li><li>- Un peu de CSS et responsive</li></ul></li><li>- Composant Home:<ul style="list-style-type: none"><li>- Création du composant + container</li><li>- Ajout du header avec la sidebar, de la searchbar et de texte sur le composant</li><li>- Quand l'utilisateur écrit dans la searchbar, les éléments autour disparaissent</li></ul></li></ul>

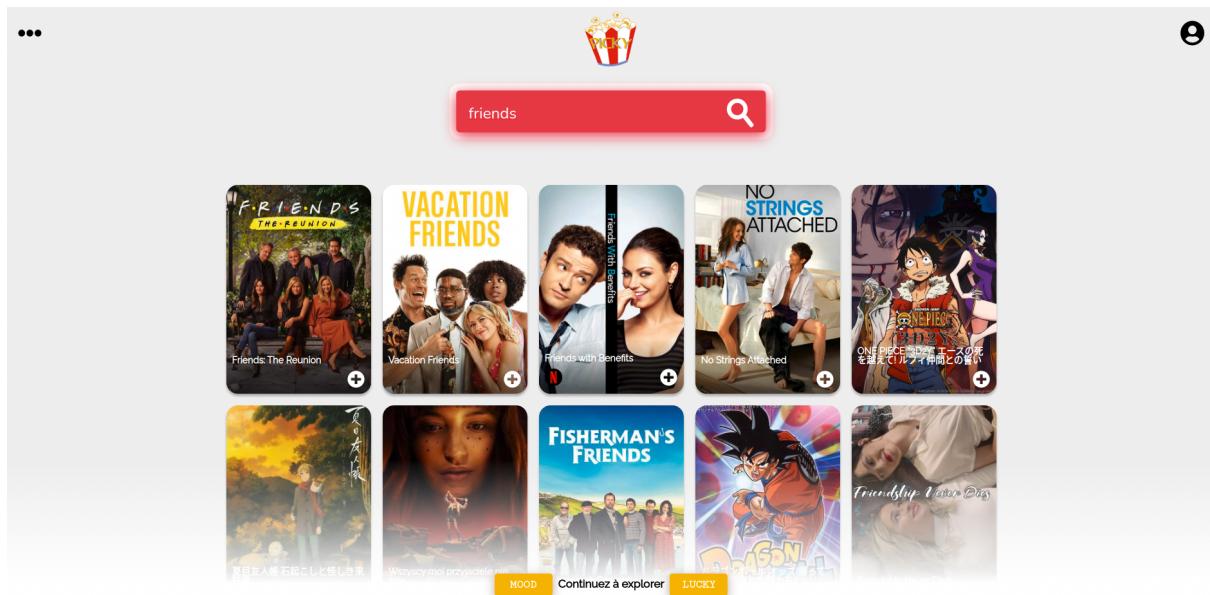
## Barre de recherche:

Sur la page d'accueil, une barre de recherche permet de trouver les programmes dont le titre correspond à ce qu'on y écrit. Lorsque l'utilisateur écrit dedans puis appuie sur entrée, la barre de recherche remonte vers le haut de la page.

Etat de base:



Etat lors d'une recherche:



Lorsque l'utilisateur arrive sur la page, il y a un focus automatique sur cette barre de recherche pour qu'il puisse directement écrire dedans. Dans le composant SearchBar:

```
const inputRef = useRef(null);

useEffect(() => {
  inputRef.current.focus();
}, []);
```

On crée une référence, qui n'est pour l'instant associée à aucun élément.

On crée le focus dessus, en utilisant le hook d'effet `useEffect`. En lui donnant un tableau vide en second argument, le focus se déclenche ici dès qu'on arrive sur la page.

```
return (
  <div className="cover">
    <form className="form" onSubmit={handleSubmit}>
      <div className="tb">
        <div className="td">
          <input
            className="input"
            ref={inputRef}
            type="text"
            placeholder="Search"
            value={searchInputValue}
            onChange={onInputChange}
            required
          />
        </div>
        <div className="td" className="s-cover">
          <button className="button" type="submit">
            <div className="s-circle" />
            <span className="span" />
          </button>
        </div>
      </div>
    </form>
  </div>
)
```

Grâce au `ref={inputRef}` sur l'input, on associe la référence à l'input.

Avec `value={searchInputValue}` et `onChange={onInputChange}`, on assigne deux props à value et onChange, pour que quand l'utilisateur écrit, ce qu'il écrit soit pris en compte.

`searchInputValue` correspond à une propriété du state qui est au départ vide.  
Dans le reducer de la SearchBar:

```
export const initialState = {
  searchInputValue: '',
};
```

puis qui va se remplir et changer grâce à `onInputChange`. Dans le container de la SearchBar:



```
onInputChange: (event) => {
  dispatch(changeSearchInput(event.target.value));
},
```

Dans le reducer, `changeSearchInput` correspond au cas `CHANGE_SEARCH_INPUT`:



```
case CHANGE_SEARCH_INPUT:
  return {
    ...state,
    searchInputValue: action.newValue,
 };
```

Ce dernier modifie le state en assignant la nouvelle valeur tapée par l'utilisateur à `searchInputValue`, et ce à chaque changement, c'est-à-dire à chaque fois que l'utilisateur écrit ou efface une lettre.

Ainsi, le state est toujours à jour, donc ce qui est affiché dans la barre de recherche aussi, et quand l'utilisateur appuie sur entrée, la bonne valeur est envoyée pour faire la requête (nous revenons sur cette requête plus tard, dans la partie Picky Find).

## 2. Sprint 1

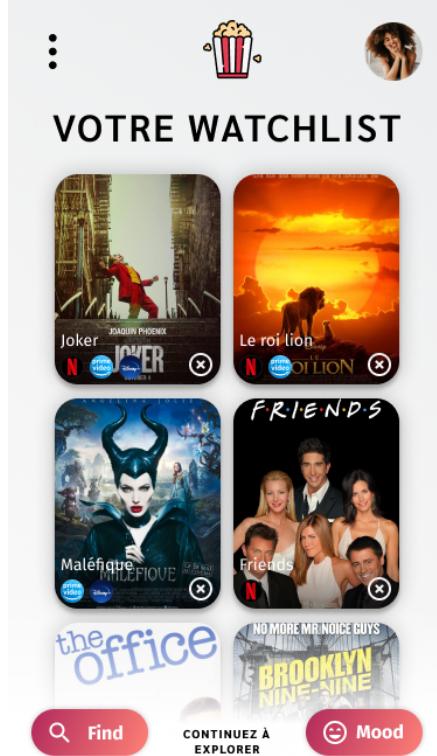
Mes principales réalisations:

11/06/2021	<ul style="list-style-type: none"><li>- Composant Home:<ul style="list-style-type: none"><li>- Ajout d'une transition pour la searchbar qui remonte</li><li>- Responsive</li></ul></li><li>- Composant Cards:<ul style="list-style-type: none"><li>- Création du composant et ajout au composant Home quand l'utilisateur tape dans la barre de recherche</li><li>- Création d'un composant Card pour chaque carte et ajout au composant Cards</li></ul></li></ul>
------------	--

	<ul style="list-style-type: none"> <li>- CSS pour changer la taille et la position des cartes</li> <li>- Scrollbar sur les cartes</li> </ul>
13/06/2021	<ul style="list-style-type: none"> <li>- Composant Cards: <ul style="list-style-type: none"> <li>- Responsive</li> <li>- Commentaires</li> </ul> </li> </ul>
14/06/2021	<ul style="list-style-type: none"> <li>- Composant Wish: <ul style="list-style-type: none"> <li>- Création, ajout des composants Header, cards et footer à l'intérieur</li> <li>- CSS: Dégradé blanc en bas de page</li> <li>- Responsive</li> <li>- Commentaires</li> </ul> </li> <li>- Composant Cards: <ul style="list-style-type: none"> <li>- Bouton pour ajouter/supprimer de la watchlist selon si le programme est déjà dedans ou non</li> <li>- Changements sur le responsive</li> </ul> </li> <li>- Aide à la réflexion sur le fonctionnement de l'application pour Picky Mood</li> </ul>
15/06/2021	<ul style="list-style-type: none"> <li>- Routes: ajout des routes pour pouvoir naviguer entre les différentes pages</li> <li>- Picky Mood: <ul style="list-style-type: none"> <li>- Ajout de la page des résultats</li> <li>- Ajout de la possibilité de sélectionner les émotions et les plateformes</li> </ul> </li> <li>- Sign Up: Création du composant</li> </ul>
16/06/2021	<ul style="list-style-type: none"> <li>- Sign Up: <ul style="list-style-type: none"> <li>- CSS</li> <li>- Création du formulaire</li> <li>- Input contrôlés</li> </ul> </li> <li>- Routes: ajout du Picky Mood aux routes</li> </ul>
17/06/2021	<ul style="list-style-type: none"> <li>- Picky Find: <ul style="list-style-type: none"> <li>- Création du middleware pour l'appel à l'API</li> <li>- Quand l'utilisateur écrit dans la barre de recherche, des résultats apparaissent, avec le titre et l'image des programmes</li> </ul> </li> </ul>

### Composant Card:

J'ai premièrement imaginé les cartes pour les résultats lorsque j'ai créé la maquette graphique version mobile de la watchlist de Picky (Picky Wish), sur Figma:



Je l'ai ensuite reproduit sur notre application, à l'aide de React et de Sass.

Voici un exemple de carte, tiré de l'application:



Le composant React pour chaque carte:

```
● ● ●

import React from 'react';
import posterPopcorn from 'src/assets/poster-popcorn.jpg';

import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faPlusCircle, faTimesCircle } from '@fortawesome/free-solid-svg-icons';

import './card.scss';

// Display of one card
const Card = ({  
    id,  
    title,  
    poster,  
    platformsInfos,  
    platforms,  
    addToWish,  
    removeFromWish,  
    program,  
    bookmarksIds,  
}) => {  
    let isInWatchList = false;  
  
    return (  
        <div className="card">  
            { // Poster of the programme  
            }  
            <img  
                className="card__poster"  
                src={poster ? poster : posterPopcorn}  
                alt="Affiche du programme"  
            />  
            { // Infos on the program (title, platforms and button to add it to/delete from watchlist)  
            }  
            <div className="card__infos">  
                <p className="card__infos__title">{title}</p>  
                <div className="card__infos__elements">  
                    <div className="card__platforms">  
                        { // For every platform, if it is available in France, we get the logo and display it  
                        }  
                        {platformsInfos.map((platformInfo) => (  
                            platforms.map((platform) => {  
                                if(platformInfo.id === platform.id) {  
                                    return (  
                                        <img  
                                            key={platform.id}  
                                            className="card__platform"  
                                            src={platform.logo}  
                                            alt="Logo de la plateforme"  
                                        />  
                                    )  
                                }  
                            )  
                        ))}  
                    </div>  
                    { // If the program is in the watchlist, a button to delete it from it is displayed.  
                    }  
                    { // Otherwise, a button to add it to the watchlist is displayed.  
                    }  
                    {bookmarksIds.map((bookmarkId) => {  
                        if (bookmarkId === id) {  
                            isInWatchList = true;  
                        }  
                    })}  
                    {isInWatchList && (  
                        <FontAwesomeIcon  
                            icon={faTimesCircle}  
                            size="2x"  
                            className='card__watchlistButton'  
                            onClick = {(event) => {  
                                event.stopPropagation();  
                                removeFromWish(id);  
                            }}  
                        />  
                    )}  
                    {!isInWatchList && (  
                        <FontAwesomeIcon  
                            icon={faPlusCircle}  
                            size="2x"  
                            className='card__watchlistButton'  
                            onClick = {(event) => {  
                                event.stopPropagation();  
                                addToWish(program);  
                            }}  
                        />  
                    )}  
                </div>  
            </div>  
        </div>  
    );  
};  
export default Card;
```

Chaque carte est composée de:

une affiche (ou une image pré définie si le programme n'a pas d'affiche)

un titre

les logos des plateformes disponibles en France

un bouton pour ajouter ou supprimer de la watchlist (nous voyons ça plus en détail dans la partie sur la watchlist)

Feuille de style du composant Card, en Sass:

```
● ● ●

@use 'src/styles/vars';

.card {
  display: flex;
  flex-direction: column;
  text-align: left;

  width: 250px;
  height: 360px;
  margin: 10px;

  border-radius: 20px;
  position: relative;
  box-shadow: 0px 0px 20px 0px #00000040;
  box-shadow: 0px 4px 4px 0px #00000040;

  &:last-child {
    &:after {
      display: block;
      content: '';
      min-height: 100px;
      width: 100px;
    }
  }

  &__infos {
    height: auto;
    padding-top: 25px;
    bottom: 0;
    width: 250px;
    position: absolute;

    background: linear-gradient(0deg, rgba(0, 0, 0, 0.5) 7%, rgba(0, 0, 0, 0) 100%);
    border-radius: 0 0 20px 20px;

    &__title {
      margin: 0 10px 10px 10px;

      color:white;
      font-weight: bold;
    }

    &__elements {
      display: flex;
      flex-direction: row;
      justify-content: space-between;

      margin: 0 10px 10px 10px;
    }
  }

  &__poster {
    height: 100%;

    object-fit: cover;
    border-radius: 20px;
  }

  &__platforms {
    color: white;
  }

  &__platform {
    margin-right: 5px;
  }

  &__platform, &__watchlistButton{
    width: 30px;
    height: 30px;

    border-radius: 20px;
    cursor: pointer;
  }

  &__watchlistButton {
    color: white;
    width: 30px;
    height: 30px;
  }
}

@media (max-width: vars.$small) {
  .card {
    width: 160px;
    height: 228px;

    &__infos {
      bottom: 0;
      width: 160px;

      &__title {
        margin: 0 5px 5px 5px;
      }

      &__elements {
        margin: 0 5px 5px 5px;
      }
    }
}
}
```

je crée deux ombres pour renforcer l'impression de relief

je crée un espace entre le dernier élément et le bas de la page pour que la dernière ligne ne soit pas collée au bas de la page quand on a fini de scroller

je crée un dégradé noir sur le bas des cartes pour que les informations ressortent

j'utilise un media query pour adapter les tailles des cartes quand la largeur de l'affichage est à small (soit 420px dans mon fichier vars)  
Picky Find:

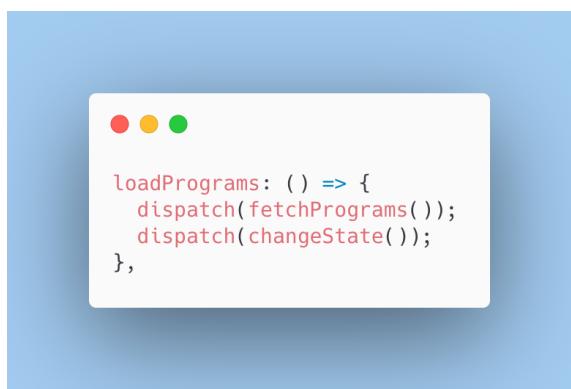
Quand l'utilisateur écrit dans la barre de recherche puis appuie sur entrée, en plus de faire remonter la barre de recherche en haut de la page, cela lance une recherche, avec la fonction `handleSubmit()`, qui se trouve sur le composant `SearchBar`:



`evt.preventDefault()` empêche de recharger la page.

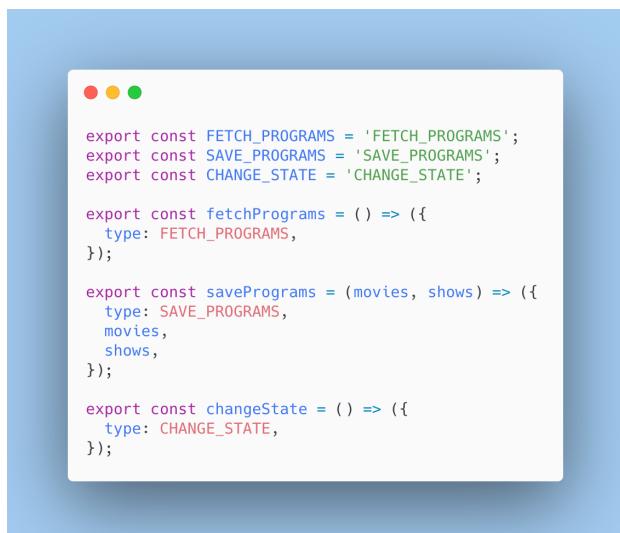
Puis la fonction `loadPrograms()` est appelée, pour lancer une requête.

Dans le container de la `earchBar`, dans la méthode `mapDispatchToProps()`, qui permet d'envoyer des actions aux propriétés du container:



`loadPrograms` envoie les actions `fetchPrograms()` et `changeState()`.

Actions de `pickyFind`:



- Dans le reducer de pickyFind:

```

case CHANGE_STATE: {
  return {
    ...state,
    loading: true,
    research: true,
  }
}

```

*CHANGE\_STATE* modifie le state de pickyFind:

en passant *loading* à *true*: quand *loading* est à *true*, on affiche un loader sur la page

en passant *research* à *true*: cela modifie la page, notamment en faisant remonter la barre de recherche, en enlevant les composants présents précédemment et en ajoutant le composant qui correspond aux cartes des résultats.

- Dans le middleware de pickyFind:

```

case FETCH_PROGRAMS: {
  // First request for movies based on what the user has typed in the searchbar
  const searchInput = store.getState().search.searchInputValue;
  axios.get(`https://projet-picky.herokuapp.com/search/movies/${searchInput}`)
    .then((response) => {
      const { movies } = response.data[0];
      // If the first request worked, second request for shows base on what the user has typed
      axios.get(`https://projet-picky.herokuapp.com/search/shows/${searchInput}`)
        .then((response) => {
          const { shows } = response.data[0];
          // Movies and shows are sent to be displayed
          store.dispatch(savePrograms(movies, shows));
        })
        .catch((error) => {
          console.log(`error`, error)
        });
    })
    .catch((error) => {
      console.log(`error`, error)
    });
  break;
}

```

*FETCH\_PROGRAMS* permet de faire une requête pour avoir des résultats: On récupère ce que l'utilisateur a écrit dans la barre de recherche, et on fait deux requêtes vers le back:

- Une requête pour récupérer les films qui correspondent à la recherche
- Une autre pour récupérer les séries qui correspondent à la recherche

Si les deux requêtes fonctionnent, on sauvegarde les programmes grâce à `savePrograms(movies, shows)`, qui correspond dans le reducer de `pickyFind` au cas `SAVE_PROGRAMS`:



```

case SAVE_PROGRAMS: {
  return {
    ...state,
    movies: action.movies,
    shows: action.shows,
    loading: false,
  };
}

```

Celui-ci stocke dans le state les films et les séries récupérées et remet loading à false pour que le loader disparaîsse et que les résultats apparaissent.

Dans le composant Cards:



```

if (loading === false) {
  if (currentPage === "home") {
    // If the cards display the results of Picky Find
    if (movies.length === 0 && shows.length === 0) {
      // If there is no result, a message is displayed
      return (
        <div className="cards cards__result--empty">
          <div className="result--empty">
            <img className="cards__img__sadpopcorn" src={sadPopcorn} alt="Logo de Picky" />
            <p>Il n'y a aucun résultat pour votre recherche</p>
          </div>
        </div>
      )
    } else {
      return (
        <>
        <div className="cards">
          // The movies are displayed
        </div>
        {movies.map((movie) => (
          <div onClick={() => {openModal(); getDetails(movie.id, 'movie')}} key={movie.id}>
            <Card
              id={movie.id}
              title={movie.title}
              poster={movie.poster}
              platformsInfos={movie.svods}
              key= {movie.id}
              program={movie}
              genre= "movie"
            />
          </div>
        ))}
        // The shows are displayed
      ) {
        <div onClick={() => {openModal(); getDetails(show.id, 'show')}} key={show.id}>
          <Card
            id={show.id}
            title={show.title}
            poster={show.poster}
            platformsInfos={show.svods}
            key= {show.id}
            program={show}
            genre="show"
          />
        </div>
      )}
    </div>
  );
}
}

```

Si loading est à faux et que la page actuelle est home (les cartes affichent parfois les résultats d'autres parties du site comme Picky Mood):

- S'il n'y a pas de résultats, on affiche un message qui le précise.
- S'il y a des résultats, on affiche d'abord les films puis les séries, avec toutes leurs informations.

### 3. Sprint 2

Mes principales réalisations:

18/06/2021	<ul style="list-style-type: none"> <li>- Plateformes: quand l'utilisateur ouvre le site, les plateformes sont récupérées de l'API (pour pouvoir en afficher les logos quand nécessaire)</li> <li>- Picky Find: <ul style="list-style-type: none"> <li>- Quand l'utilisateur écrit au moins 3 lettres, la recherche démarre</li> <li>- Affichage des logos des plateformes du programme</li> </ul> </li> </ul>
21/06/2021	<ul style="list-style-type: none"> <li>- Picky Find: <ul style="list-style-type: none"> <li>- La recherche est basée sur ce que l'utilisateur écrit</li> <li>- La recherche est faite quand l'utilisateur appuie sur entrée</li> <li>- Réparation du problème de scroll sur les résultats</li> <li>- L'utilisateur est obligé d'écrire quelque chose dans la barre de recherche pour envoyer (required)</li> <li>- Commentaires et nettoyage du code</li> </ul> </li> </ul>
22/06/2021	<ul style="list-style-type: none"> <li>- Picky Mood: <ul style="list-style-type: none"> <li>- Affichage de 20 résultats au hasard parmi les résultats de la requête</li> <li>- Message affiché quand il n'y a aucun résultat correspondant à la recherche</li> <li>- Quand il y a une nouvelle recherche, les résultats d'avant s'effacent et il y a un loading qui s'affiche</li> <li>- Commentaires et nettoyage du code</li> </ul> </li> </ul>
23/06/2021	<ul style="list-style-type: none"> <li>- Picky Find: <ul style="list-style-type: none"> <li>- Design de la barre de recherche</li> <li>- Scrollbar invisible</li> </ul> </li> </ul>
24/06/2021	<ul style="list-style-type: none"> <li>- Cards: <ul style="list-style-type: none"> <li>- Les résultats qui n'ont pas de poster ont maintenant une image standard</li> <li>- Ombre autour des cartes</li> <li>- Meilleure position du dégradé</li> </ul> </li> <li>- Picky Find: <ul style="list-style-type: none"> <li>- Quand l'utilisateur clique sur le logo dans le header, les résultats sont remis à zéro</li> </ul> </li> </ul>

## Résultats de Picky Mood:

Lorsque l'utilisateur fait une recherche avec Picky Mood, il choisit selon différents critères: films ou séries, humeur, puis plateformes. Sur cette dernière page, il y a ensuite un bouton Search (recherche):



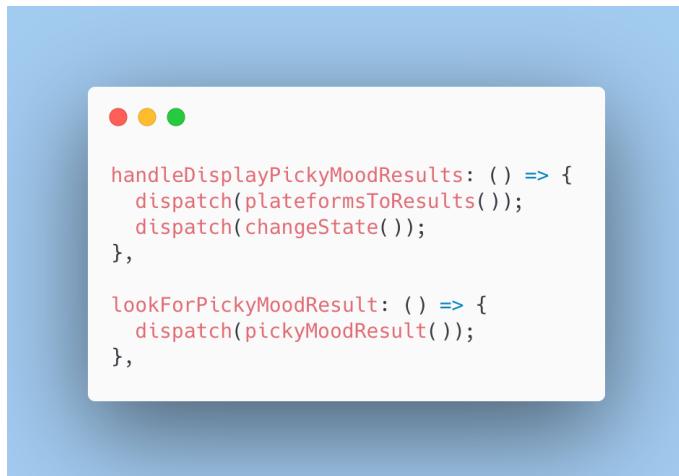
Dans le dossier des composants, un dossier PickyMood contient plusieurs fichiers JavaScript: un pour chaque étape du “questionnaire” (ShowOrSeries, Feelings, Platforms, Results) et un fichier index.

Dans le fichier Platforms du composant PickyMood:



Dans ce fichier également, les fonctions correspondent à des props. Ces props viennent du container de PickyMood, qui est utilisé dans les routes du composant principal, App.

Dans le container PickyMood, dans la méthode `mapDispatchToProps()`, qui permet d'envoyer des actions aux propriétés du container:



`handleDisplayPickyMoodResults` envoie les actions `platformsToResults()` et `changeState()`.

`lookForPickyMoodResult` envoie l'action `pickyMoodResult()`.

Dans les actions de pickyMood, on a donc:



`PICKYMOOD_RESULT` est ensuite utilisé dans le middleware de `pickyMood`, qui permet de faire les requêtes vers le back.

```

import axios from 'axios';
import { PICKYMOOD_RESULT, saveResults } from 'src/actions/pickyMood';

const result = (store) => (next) => (action) => {
  switch (action.type){
    case PICKYMOOD_RESULT: {
      const state = store.getState();
      // Request for programs based on what the user chose in Picky Mood
      axios.post('https://projet-picky.herokuapp.com/moodresults', {
        ShowOrMovie: state.pickyMood.ShowOrMovie,
        emotions: state.pickyMood.emotions,
        platforms: state.pickyMood.platforms,
      })
      .then((response) => {
        if(response.data[0].movies) {
          // If the results are movies:
          let results = [];
          let randomResults = [];
          let newRandomResult = {};
          // All the results are put in a single array
          for (let i=0; i < response.data.length; i++) {
            results = results.concat(response.data[i].movies);
          };
          if (results.length > 20) {
            // If there are more than 20 results, the results are randomized and only 20 are selected
            for(let i=0; i < 20; i++) {
              newRandomResult = results[Math.floor(Math.random()*results.length)];
              if (!randomResults.includes(newRandomResult)) {
                randomResults.push(newRandomResult);
              } else {
                i--;
              }
            };
            store.dispatch(saveResults(randomResults, 'movie'));
          } else {
            // If there are 20 results or less, they are all displayed
            store.dispatch(saveResults(results, 'movie'));
          }
        } else if (response.data[0].shows) {
          // If the results are shows:
          let results = [];
          let randomResults = [];
          let newRandomResult = {};
          // All the results are put in a single array
          for (let i=0; i < response.data.length; i++) {
            results = results.concat(response.data[i].shows);
          };
          if (results.length > 20) {
            // If there are more than 20 results, the results are randomized and only 20 are selected
            for(let i=0; i < 20; i++) {
              newRandomResult = results[Math.floor(Math.random()*results.length)];
              if (!randomResults.includes(newRandomResult)) {
                randomResults.push(newRandomResult);
              } else {
                i--;
              }
            };
            store.dispatch(saveResults(randomResults, 'shows'));
          } else {
            // If there are 20 results or less, they are all displayed
            store.dispatch(saveResults(results, 'shows'));
          }
        }
      })
      .catch((error) => {
        console.trace(error)
      });
      break;
    }
    default:
      next(action);
  }
};

export default result;

```

En décomposant:

```
● ● ●  
axios.post('https://projet-picky.herokuapp.com/moodresults', {  
  ShowOrMovie: state.pickyMood.ShowOrMovie,  
  emotions: state.pickyMood.emotions,  
  platforms: state.pickyMood.platforms,  
})
```

On fait dans un premier temps une requête post vers l'API de Picky grâce à Axios, en passant en deuxième argument les choix de critères de l'utilisateur.

```
● ● ●  
if(response.data[0].movies) {  
  // If the results are movies:  
  let results = [];  
  let randomResults = [];  
  let newRandomResult = {};  
  // All the results are put in a single array  
  for (let i=0; i < response.data.length; i++) {  
    results = results.concat(response.data[i].movies);  
  };  
  if (results.length > 20) {  
    // If there are more than 20 results, the results are randomized  
    // and only 20 are selected  
    for(let i=0; i < 20; i++) {  
      newRandomResult = results[Math.floor(Math.random()*results.length)];  
      if (!randomResults.includes(newRandomResult)) {  
        randomResults.push(newRandomResult);  
      } else {  
        i--;  
      }  
    };  
    store.dispatch(saveResults(randomResults, 'movie'));  
  } else {  
    // If there are 20 results or less, they are all displayed  
    store.dispatch(saveResults(results, 'movie'));  
  }  
}
```

Si les résultats sont des films (on vérifie ici qu'il y ait bien un tableau movies dans le premier élément du tableau data que l'on récupère dans la réponse), on met ensuite tous les objets qui représentent les films dans un seul tableau. En effet, si l'utilisateur a sélectionné plusieurs émotions, il y aura autant de tableaux que d'émotions.

Ensuite, on veut

garder maximum 20 résultats à afficher à la fois, donc s'il y a plus de 20 résultats, on en prend 20 parmi ceux-ci, au hasard. Sinon, s'il y en a moins de 20, on les garde tous. Dans les deux cas, on finit par envoyer l'action `saveResult()`, avec en premier argument les résultats, et en deuxième argument la string 'movie', pour pouvoir ensuite faire la distinction nécessaire entre films et séries.

Si les résultats sont non pas des films mais des séries, on répète exactement la même démarche, en remplaçant à chaque fois movies par shows.

Dans le reducer de pickyMood:



SAVE\_RESULTS modifie le state de pickyMood, pour y stocker les résultats récupérés et leur genre (films ou séries), et remet loading à false pour que le loader disparaît et que les résultats apparaissent.

Enfin, ces informations stockées dans le state vont permettre d'afficher les résultats à travers le composant Cards:



Si les cartes affichent les résultats de pickyMood (ce composant sert à différentes parties du site):

- S'il n'y a pas de résultat, on affiche un message qui l'explique
- S'il y a des résultats, on affiche tous les programmes stockés dans le state à travers le composant Card, avec toutes les informations nécessaires.

#### 4. Sprint 3

Mes principales réalisations:

25/06/2021	<ul style="list-style-type: none"> <li>- Responsive des cartes</li> <li>- Ombre sur les boutons</li> <li>- Picky Wish (watchlist):           <ul style="list-style-type: none"> <li>- L'utilisateur peut voir ce qu'il a dans sa watchlist</li> </ul> </li> </ul>
27/06/2021	<ul style="list-style-type: none"> <li>- Picky Wish:           <ul style="list-style-type: none"> <li>- Quand un programme n'est pas encore dans la watchlist, il y a un signe plus pour l'ajouter. Quand il y est déjà, il y a une croix pour le supprimer</li> <li>- Dans la watchlist, dès qu'on clique sur la croix, le programme n'apparaît plus sur la page</li> </ul> </li> </ul>
28/06/2021	<ul style="list-style-type: none"> <li>- Picky Wish:           <ul style="list-style-type: none"> <li>- Dès qu'on clique sur le bouton d'ajout/suppression, il se met à jour directement</li> <li>- Ajout d'une icône pour le signe plus et la croix</li> <li>- Commentaires</li> <li>- Toastify pour prévenir quand un programme a bien été ajouté/supprimé de la watchlist</li> <li>- Toastify pour prévenir d'une erreur quand un visiteur essaye d'ajouter un programme à sa watchlist sans être connecté</li> <li>- Ajout d'une image quand la watchlist est vide</li> <li>- CSS sur tous les boutons pour les unifier</li> </ul> </li> </ul>
29/06/2021	<ul style="list-style-type: none"> <li>- Amélioration du responsive</li> <li>- Ajout de liens pour naviguer sur le site sur Picky Find, Picky Mood et Picky Lucky une fois que les résultats sont affichés</li> </ul>
30/06/2021	<ul style="list-style-type: none"> <li>- Dégradé de la bonne couleur en dark mode</li> <li>- Réparation des derniers problèmes en CSS</li> </ul>
01/07/2021	<ul style="list-style-type: none"> <li>- Réparation des derniers problèmes en CSS</li> </ul>

### Picky Wish (watchlist):

Sur chaque carte, il y a un bouton qui représente soit un plus soit une croix, selon si le programme se trouve ou non dans la watchlist de l'utilisateur. Ils y ajoutent et suppriment respectivement le programme.



Pour ce faire:

Dans le composant Cards:

```
● ● ●

useEffect(() => {
  lottie.loadAnimation({
    container: container.current,
    renderer: 'svg',
    loop: true,
    autoplay: true,
    animationData: require('./movieloading.json')
  });
  getBookmarksIds();
}, [loading]);
```

cela est nécessaire.

`getBookmarksIds()` est ensuite appelée, pour récupérer les ids des programmes présents dans la watchlist de l'utilisateur:

On utilise le hook d'effet `useEffect`. En lui donnant un tableau avec `loading` en second argument, les actions passées en premier argument se déclenchent dès qu'on arrive sur la page, puis à chaque fois que l'état de `loading` change.

`lottie.loadAnimation` est utilisé pour pouvoir afficher une animation de loading lorsque

Dans les actions de la watchlist:

```
export const GET_BOOKMARKS_IDS = 'GET_BOOKMARKS_IDS';

export const getBookmarksIds = () => ({
  type: GET_BOOKMARKS_IDS,
});
```

GET\_BOOKMARKS\_IDS est ensuite utilisé dans le middleware de pickyWish, qui permet de faire les requêtes vers le back:

```
case GET_BOOKMARKS_IDS: {
  axios.get('https://projet-picky.herokuapp.com/member/bookmark',
    config)
  .then(({response}) => {
    // Gets all the programs that are currently in the watchlist
    const bookmarks = response.data.data;

    // An empty array to put all the ids
    // of the programs in the watchlist is created
    const bookmarksIds = [];

    if(bookmarks !== null) {
      bookmarks.forEach((bookmark) => {
        // For each program in the watchlist, we get its ID
        const bookmarkId = bookmark.betaseries_id;
        // And add it to the array
        bookmarksIds.push(bookmarkId);
      });
    }

    // The array of ids is sent
    store.dispatch(getBookmarksIdsSuccess(bookmarksIds));
  })
  .catch(({error}) => {
    console.log(`error`, error)
  });
  break;
}
```

On fait une requête get avec Axios pour récupérer les programmes présents dans la watchlist. Si la requête se passe bien, on les stocke dans la constante bookmarks, on crée un tableau vide qu'on assigne à la constante bookmarksIds qui contiendra les ids de tous ces programmes, puis si la requête a bien récupéré quelque chose (si bookmarks n'est pas null), on boucle sur bookmarks et on récupère son id que l'on ajoute dans le tableau bookmarksIds.

getBookmarksIdsSuccess, avec les ids en argument.

On envoie ensuite l'action

Dans les actions de la watchlist:

```
export const GET_BOOKMARKS_IDS_SUCCESS = 'GET_BOOKMARKS_IDS_SUCCESS';

export const getBookmarksIdsSuccess = (bookmarksIds) => ({
  type: GET_BOOKMARKS_IDS_SUCCESS,
  bookmarksIds,
});
```

Dans le reducer de la watchlist:

```
● ● ●  
export const initialState = {  
  wish: [],  
  bookmarksIds: [],  
};
```

Dans le state de la watchlist, une propriété `bookmarksIds` est initialement un tableau vide, qui pourra ensuite se remplir de tous les ids correspondants aux programmes ajoutés dans la watchlist.

```
● ● ●  
case GET_BOOKMARKS_IDS_SUCCESS: {  
  return {  
    ...state,  
    bookmarksIds: action.bookmarksIds,  
  }  
};
```

`GET_BOOKMARKS_IDS_SUCCESS` permet de stocker les ids des programmes récupérés dans le state.

Dans le container de Card:

```
● ● ●  
const mapStateToProps = (state) => ({  
  platforms: state.platforms.platforms,  
  bookmarksIds: state.watchlist.bookmarksIds,  
});
```

La méthode `mapStateToProps` permet d'envoyer des variables au container. On assigne ici le tableau `bookmarksIds` du state à `bookmarksIds`, pour pouvoir ensuite l'utiliser dans le composant Card.

Dans le composant Card:

```
● ● ●  
let isInWatchList = false;
```

Une variable `isInWatchList` est déclarée à false.

```
● ● ●  
{bookmarksIds.map((bookmarkId) => {  
  if (bookmarkId === id) {  
    isInWatchList = true;  
  }  
})}
```

Pour chaque id de `bookmarksIds`, qui vient du state et qui représente les id des programmes ajoutés par l'utilisateur à la watchlist, on le compare avec l'id du programme actuellement affiché, qui vient des props, et qui a été donné par le composant Cards. Si cet id correspond à

un de ces `bookmarksIds`, on passe `isInWatchList` à true.

```
  {!isInWatchList && (
    <FontAwesomeIcon
      icon={faPlusCircle}
      size="2x"
      className='card__watchlistButton'
      onClick = {({event}) => {
        event.stopPropagation();
        addToWish(program);
      }}
    />
  )}
```

Si `isInWatchList` est false, et donc si le programme n'est pas dans la watchlist, un bouton représentant un plus est affiché. Lorsque l'utilisateur clique dessus, `event.stopPropagation()` évite que l'événement courant ne se propage plus loin dans les phases de capture et de déploiement, puis `addToWish(program)` ajoute le programme à la watchlist.

```
  {isInWatchList && (
    <FontAwesomeIcon
      icon={faTimesCircle}
      size="2x"
      className='card__watchlistButton'
      onClick = {({event}) => {
        event.stopPropagation();
        removeFromWish(id);
      }}
    />
  )}
```

Si `isInWatchList` est true, et donc si le programme est dans la watchlist, un bouton représentant une croix est affiché. Lorsque l'utilisateur clique dessus, `event.stopPropagation()` évite que l'événement courant ne se propage plus loin dans les phases de capture et de déploiement, puis `removeFromWish(id)` supprime le programme de la watchlist.

Dans le container de Card, dans la méthode `mapDispatchToProps`:

```
  addToWish: (program) => {
    dispatch(addToWish(program));
  },
  removeFromWish: (programId) => {
    dispatch(removeFromWish(programId));
  },
```

`addToWish` envoie l'action `addToWish` en lui passant le programme en argument.

`removeFromWish` envoie l'action `removeFromWish` en lui passant l'id du programme en argument.

Dans les actions de la watchlist, on a donc:

```
export const ADD_TO_WISH = 'ADD_TO_WISH';
export const REMOVE_FROM_WISH = 'REMOVE_FROM_WISH';

export const addToWish = (programswish) => ({
  type: ADD_TO_WISH,
  programswish,
});

export const removeFromWish = (programId) => ({
  type: REMOVE_FROM_WISH,
  programId,
});
```

ADD\_TO\_WISH et REMOVE\_FROM\_WISH sont ensuite utilisés dans le middleware de pickyWish, qui permet de faire les requêtes vers le back.

Dans le middleware:

On a la configuration avec le token nécessaire pour identifier l'utilisateur:

```
const config = {
  headers: {
    "Bearer": `${store.getState().status.token}`,
    "Accept": "application/json",
    "Content-Type": "application/json",
  },
};
```

- Pour ajouter:

```
case ADD_TO_WISH: {
  // We get all the information from the programs in the watchlist
  const bodyParameters = {
    betaseries_id: action.programswish.id,
    poster: action.programswish.poster,
    platform: action.programswish.svods,
    title: action.programswish.title,
  };

  // Adds the program to the watchlist with every information needed
  axios.post('https://projet-picky.herokuapp.com/bookmark',
    bodyParameters,
    config
  )
  .then(() => {
    store.dispatch(getBookmarksIds());
    store.dispatch(getBookmark());
    notifyAdd();
  })
  .catch((error) => {
    console.log(`error`, error);
    notifyVisitor();
  });
  break;
};
```

On stocke dans un objet l'id du programme dans BetaSeries (l'API tierce), l'affiche, les plateformes et le titre du programme qui a été passé en argument quand l'utilisateur a cliqué sur le bouton d'ajout.

On fait ensuite une requête post vers l'API de Picky avec Axios, en passant en

deuxième argument l'objet correspondant au programme, et en troisième argument la configuration qui permettra notamment de reconnaître l'utilisateur et donc d'ajouter le programme à la watchlist du bon utilisateur.

Si tout se passe bien pendant la requête, on appelle `getBookmarksIds()`, qui permet de récupérer les ids des programmes présents dans la watchlist de l'utilisateur (action vue précédemment), `getBookmark()`, qui permet de récupérer les programmes en entier et les stocker dans le state, et `notifyAdd()`.

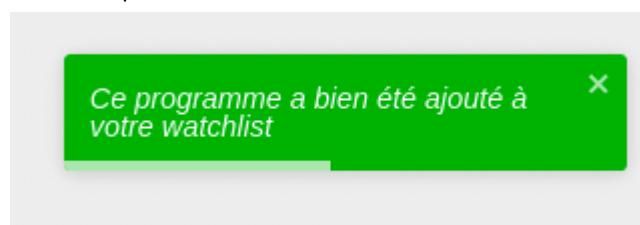
`notifyAdd()` est une fonction qui utilise `Toastify` et qui est déclarée plus tôt dans le fichier actuel:



```
const notifyAdd = () => {toast.success(`Ce programme a bien été ajouté à votre watchlist`),{ position: "top-right", autoClose: 5000, closeOnClick: true, pauseOnHover: true, draggable: true, }};
```

Cette fonction envoie un message qui s'affiche dans un toast à l'utilisateur pour lui confirmer que le programme a bien été ajouté à sa watchlist.

Voilà à quoi ressemble le toast:



Sinon, on appelle `notifyVisitor()`:



```
const notifyVisitor = () => {toast.warning(`Vous devez être connecté pour ajouter un programme à votre watchlist`),{ position: "top-right", autoClose: 5000, closeOnClick: true, pauseOnHover: true, draggable: true, }};
```

Cette fonction envoie un message qui s'affiche dans un toast à l'utilisateur pour le prévenir qu'il doit être connecté pour ajouter un programme à sa watchlist.

Voilà à quoi ressemble le toast:



*Vous devez être connecté pour ajouter un programme à votre watchlist*

- Pour supprimer:



```
case REMOVE_FROM_WISH: {
  const programId = action.programId;

  // Deletes the program from the watchlist thanks to its ID
  axios.delete(`https://projet-picky.herokuapp.com/bookmark/${programId}`, config)
    .then(() => {
      store.dispatch(getBookmarksIds());
      store.dispatch(getBookmark());
      notifyDelete();
    })
    .catch((error) => {
      console.log(`error`, error)
    });
  break;
};
```

On stocke dans une constante l'id du programme, qui a été passé en argument quand l'utilisateur a cliqué sur le bouton de suppression.

On fait ensuite une requête delete vers l'API de Picky avec Axios, avec l'id du programme et on met en deuxième argument la configuration avec le token.

Si tout se passe bien pendant la requête, on appelle `getBookmarksIds()`, qui permet de récupérer les ids des programmes présents dans la watchlist de l'utilisateur et les stocker dans le state (action vue précédemment), `getBookmark()`, qui permet de récupérer les programmes en entier et les stocker dans le state, et `notifyDelete()`.

`notifyDelete()`:



```
const notifyDelete = () => {toast.success(`Ce programme a bien été supprimé de votre watchlist`),{
  position: "top-right",
  autoClose: 5000,
  closeOnClick: true,
  pauseOnHover: true,
  draggable: true,
}};
```

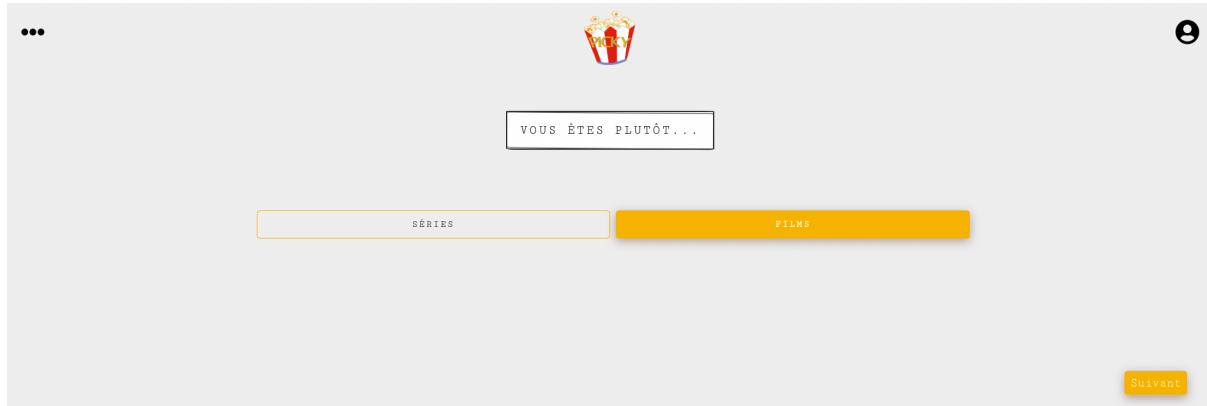
`notifyDelete()` envoie un message qui s'affiche dans un toast à l'utilisateur pour lui confirmer que le programme a été supprimé de sa watchlist.

## B. Jeu d'essai

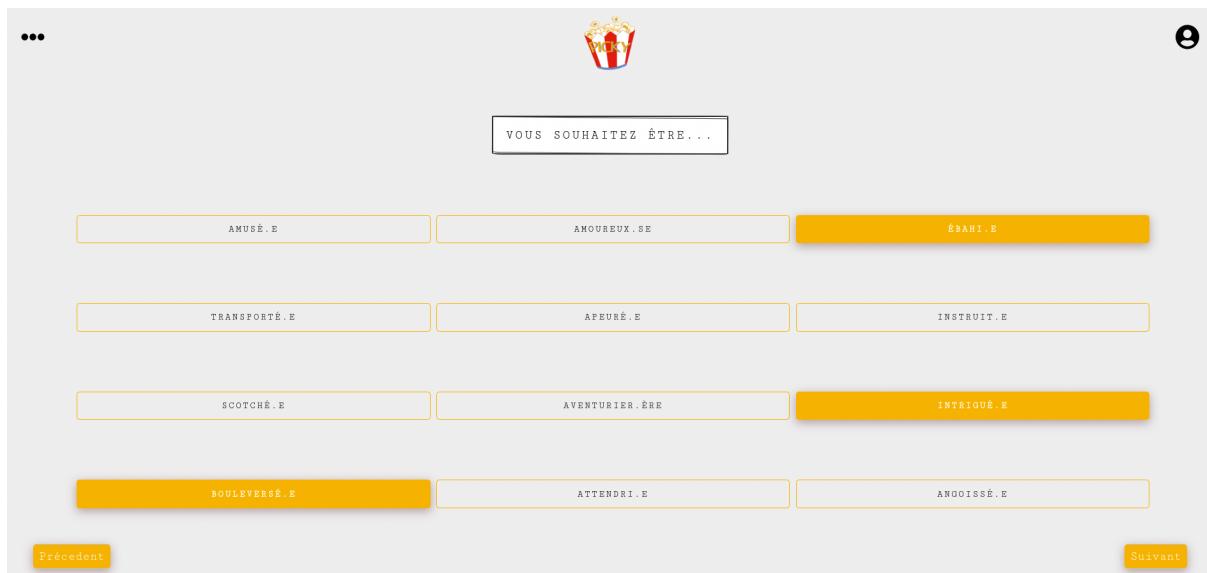
Picky Mood est une des fonctionnalités les plus importantes et représentatives de Picky. Celle-ci permet de rechercher un programme d'après certains critères (série ou film, humeur, et plateformes), et n'est accessible qu'aux utilisateurs connectés.

### Données en entrée:

Sur la première page de Picky Mood, je dois choisir entre films et séries. Dans notre exemple, disons que je choisis films.



Je clique ensuite sur suivant. Je dois choisir au moins une émotion parmi celles présentes. Je clique sur ébahi.e, intrigué.e, bouleversé.e, puis sur suivant.



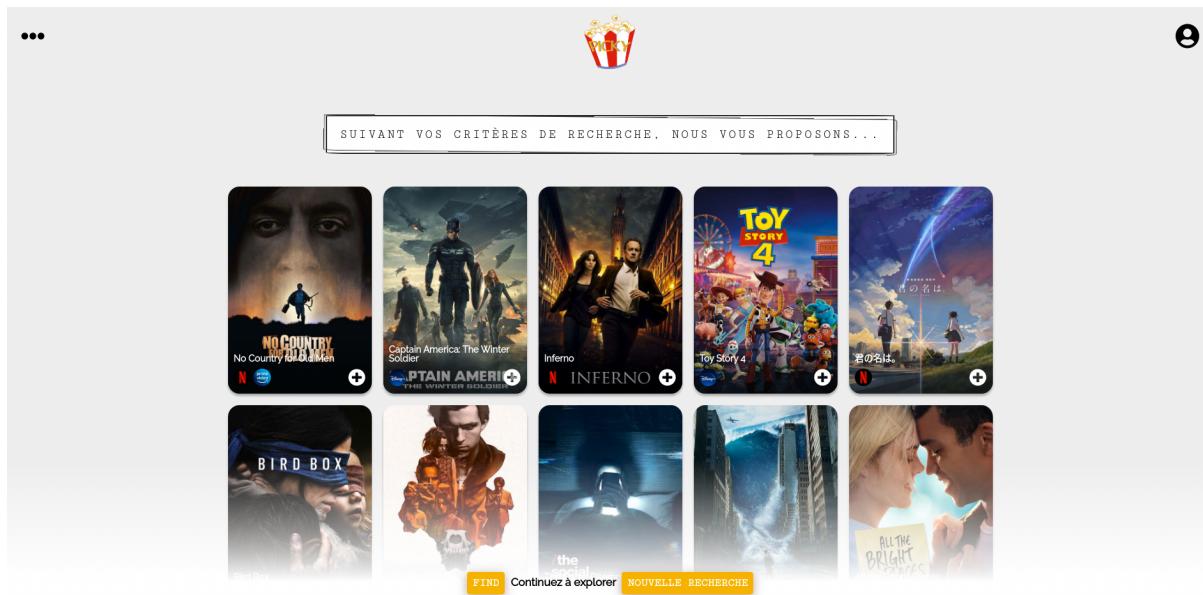
Je sélectionne ensuite parmi les plateformes celles que j'ai (au moins une): Amazon Prime Video, Netflix et Disney. Je clique ensuite sur rechercher pour avoir les résultats.



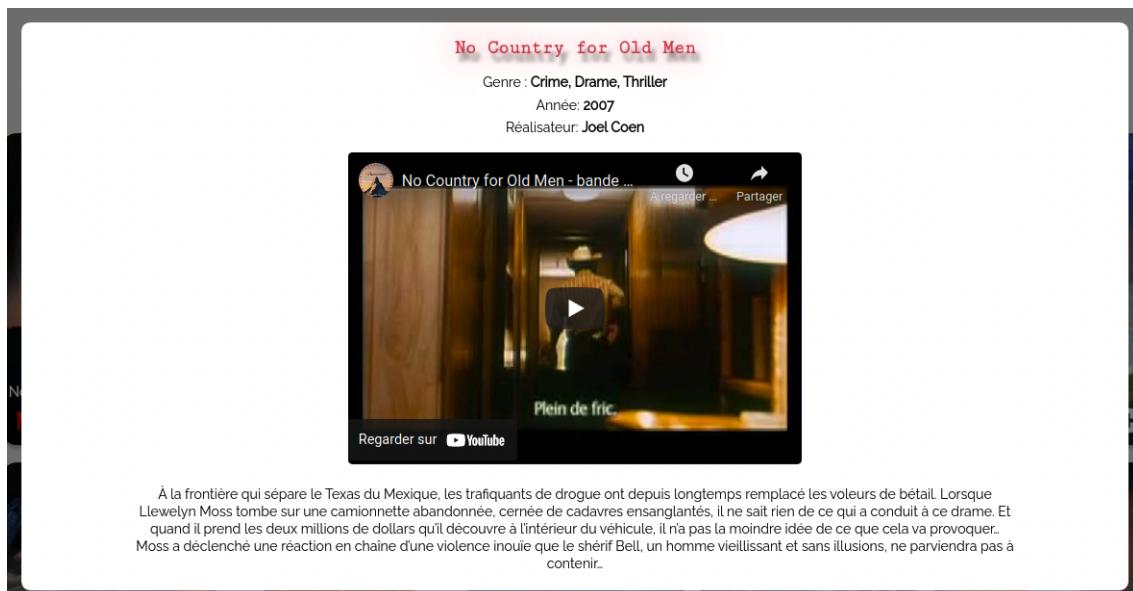
### Données attendues:

Jusqu'à 20 résultats de films dont le genre correspond aux émotions ébahie, intrigué, bouleversé, et qui se trouve sur Netflix, Amazon Prime Video et/ou Disney +.

### Donnés obtenues:



L'application me montre 20 résultats qui correspondent à ma recherche, avec leur affiche, leur titre, les logos des plateformes sur lesquelles les programmes sont présents, et un bouton pour l'ajouter à ma watchlist. Si je clique sur un des résultats, je peux voir le détail du programme.



A chaque fois que je relance une recherche, avec les mêmes critères, l'application me donne de nouveaux résultats.

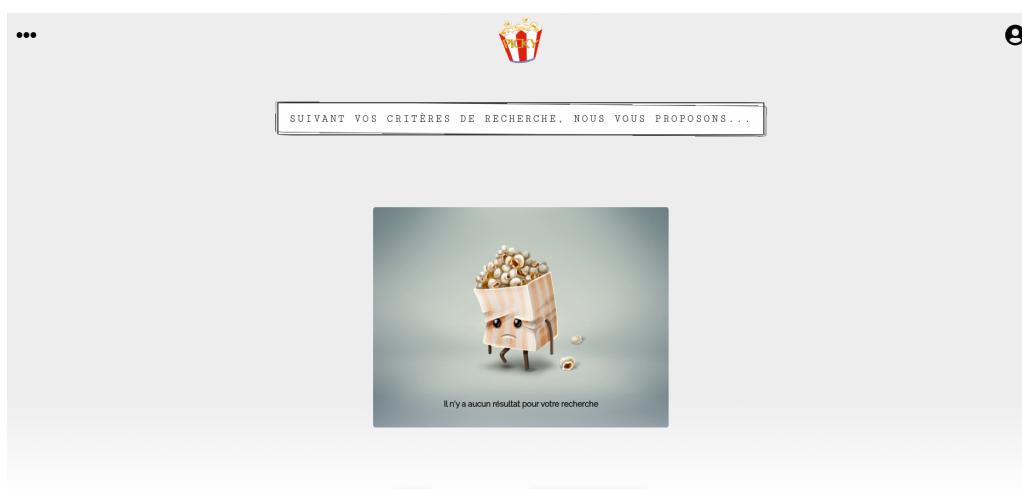
Résultat: La donnée attendue est identique à la donnée obtenue, le test est donc réussi.

#### *Scénario alternatif:*

Si j'utilise Picky Mood mais que l'application ne trouve aucun résultat qui correspondent à mes critères, je ne devrais voir affiché aucun programme au moment de la recherche.

Par exemple, si je choisis séries, apeuré.e, et Disney + (données en entrée), il ne devrait y avoir aucun résultat (données attendues).

#### Données obtenues:



L'application ne trouve effectivement aucun résultat, et l'indique.

Résultat: le test est réussi.

Données en entrée	Données attendues	Données obtenues	Résultats
<ul style="list-style-type: none"><li>- films</li><li>- ébahi.e, intrigué.e, bouleversé.e</li><li>- Netflix, Amazon Prime Video, Disney +</li></ul>	Jusqu'à 20 films qui se trouvent sur Netflix, Amazon Prime Video et/ou Disney +, et qui correspondent à ébahi.e, intrigué.e, bouleversé.e	20 films qui se trouvent sur Netflix, Amazon Prime Video et/ou Disney +, et qui correspondent à ébahi.e, intrigué.e, bouleversé.e	Test réussi
<ul style="list-style-type: none"><li>- séries</li><li>- apeuré.e</li><li>- Disney +</li></ul>	Pas de résultat	Pas de résultat	Test réussi

## IX. Recherche à partir d'un site anglophone, extrait et traduction

Lors de la création des cartes de résultat pour les programmes, je me suis rendue compte que si l'utilisateur traversait tout ou une partie des résultats, qui pouvaient être nombreux, et allait éventuellement jusqu'au bas de la page, il pouvait rapidement devenir pénible de devoir faire de nouveau défiler la page dans l'autre sens pour revenir tout en haut et continuer à naviguer sur le site.

Il m'a donc semblé nécessaire de créer un bouton qui permettrait à l'utilisateur d'être redirigé vers le haut de la page. Je voulais que ce bouton ne s'affiche que lorsque l'utilisateur avait déjà fait défiler la page verticalement d'un certain nombre de pixels. Il fallait pour cela que je trouve un moyen de trouver cette information.

En cherchant sur internet avec des termes comme "javascript get scroll position", j'ai finalement trouvé sur le MDN Web Docs (Mozilla Developer Network) la propriété pageYOffset. Je vais ici vous présenter un extrait du site, en anglais (la traduction française de cette propriété n'existe pas sur le MDN), puis je vais le traduire.

Extrait:

The read-only Window property pageYOffset is an alias for scrollY; as such, it returns the number of pixels the document is currently scrolled along the vertical axis (that is, up or down) with a value of 0.0, indicating that the top edge of the Document is currently aligned with the top edge of the window's content area.

There is slightly better support for pageYOffset than for scrollY in older browsers, but if you're not concerned about browsers more than a handful of years old, you can use either one.

The corresponding pageXOffset property, which returns the number of pixels scrolled along the horizontal axis (left and right), is an alias for scrollX.

Traduction personnelle:

La propriété en lecture seule pageYOffset de l'objet Window est semblable à scrollY. Elle retourne le nombre de pixels dont l'utilisateur a fait défiler la page, le long de l'axe vertical (vers le haut ou vers le bas), précisée au sous-pixel près. Une valeur de 0.0 signifie par exemple que le haut de l'interface Document est actuellement aligné avec le haut de la zone de contenu de la fenêtre.

Les navigateurs plus anciens supportent un peu mieux pageYOffset que scrollY, mais si vous ne vous intéressez pas aux navigateurs qui ont plus de quelques années, vous pouvez utiliser l'un ou l'autre indifféremment.

La propriété correspondante pageXOffset, qui retourne le nombre de pixels dont l'utilisateur a fait défiler la page le long de l'axe horizontal (vers la gauche et vers la droite), est, quant à elle, semblable à scrollX.

## X. Veille sur la sécurité

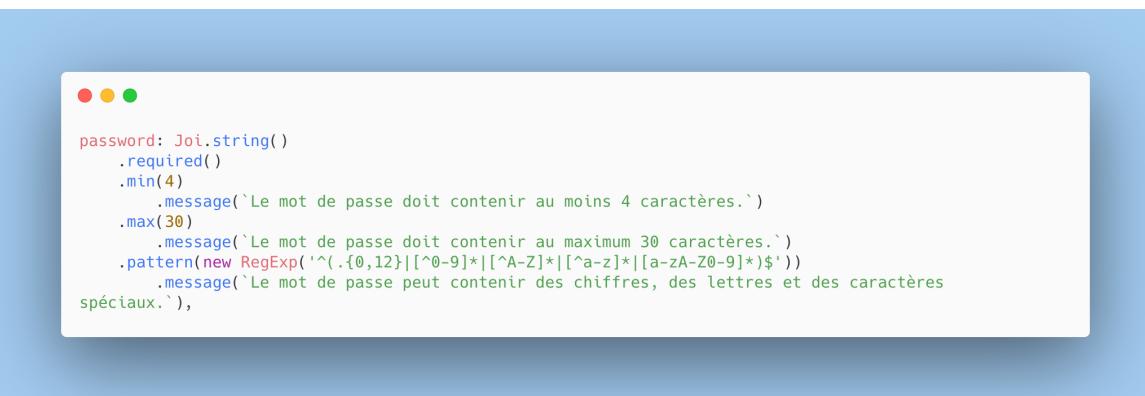
### Validation des données des formulaires:

Pour vérifier au mieux les informations rentrées par l'utilisateur dans les inputs, notamment dans le formulaire d'inscription, de vérification, mais aussi de modification du profil, j'ai effectué des recherches sur la validation des données de formulaires. Ainsi, une première vérification est effectuée côté client, puis une deuxième pourra être effectuée côté serveur, pour la meilleure sécurité possible.

Nous avons donc appliqué plusieurs éléments de validation de formulaire. Premièrement, l'attribut required nous a permis de rendre obligatoire toutes les entrées, ce qui empêche l'utilisateur de valider le formulaire s'il n'a pas rempli l'entrée.

Ensuite, nous avons utilisé **Joi** pour vérifier que les données entrées ressemblaient bien à celles que l'on attendait, côté serveur. Joi est un langage de description de schéma et validateur de données pour JavaScript. On peut, grâce à Joi, indiquer pour chaque données le schéma qu'elle doit suivre. On peut notamment dire quel type de donnée elle doit être (string, object...), si la donnée est requise, quel nombre de caractères elle doit faire au minimum, au maximum, et même indiquer une expression régulière à laquelle la donnée doit correspondre.

Par exemple, pour le mot de passe:



```
password: Joi.string()
  .required()
  .min(4)
    .message('Le mot de passe doit contenir au moins 4 caractères.')
  .max(30)
    .message('Le mot de passe doit contenir au maximum 30 caractères.')
  .pattern(new RegExp('^(.{0,12}|[^0-9]*|[^A-Z]*|[^a-z]*|[a-zA-Z0-9]*)$'))
    .message('Le mot de passe peut contenir des chiffres, des lettres et des caractères spéciaux.'),
```

Le mot de passe:

- doit être une chaîne de caractères (string)
- est requis
- doit contenir minimum 4 caractères
- doit contenir maximum 30 caractères
- doit correspondre à l'expression régulière donnée (il peut contenir des chiffres, des lettres et des caractères spéciaux)

Autre exemple, pour la confirmation du mot de passe:

```
confirmationPassword: Joi.string()
  .required()
  .valid(Joi.ref('password'))
  .messages({'any.only': 'Les mots de passe doivent être identiques.'})
```

La confirmation doit être une string, est requise, et surtout elle doit correspondre au mot de passe (être identique).

De plus, grâce aux messages personnalisés que l'on a choisis pour chaque vérification, en cas d'erreur par l'utilisateur dans les informations qu'il a écrit dans les inputs, on va pouvoir lui afficher côté client, pour qu'il puisse facilement détecter quel est le problème et le corriger le plus facilement et rapidement possible.

Pour chaque champ, on a un tableau errorMessage, qui peut être vide, ou contenir une ou plusieurs erreurs correspondants à ce que l'on a écrit dans Joi:

```
{errorMessage.length > 1 && <p className="input-middle-errorMessage">
  {errorMessage}
</p>
}
```

S'il contient au moins une erreur, on l'affiche.

Ainsi, lorsque l'utilisateur remplit un de nos formulaires, et qu'il essaye de le valider, il voit directement la/les différentes erreurs qu'il a commises, et les résoudre est pour lui beaucoup plus simple. En plus d'améliorer l'expérience utilisateur, cela améliore naturellement grandement la sécurité de l'application, puisque tant qu'une donnée n'est pas comme ce que l'on désire, elle n'a aucune chance d'être ajoutée à la base de données ou de modifier quoi que ce soit d'indésirable.

### Hachage du mot de passe:

Une fois que nous avons récupéré les informations nécessaires à la création d'un nouvel utilisateur ou à la modification des informations d'un utilisateur déjà existant, nous savions qu'il ne fallait pas stocker le mot de passe tel quel. **Bcrypt**

est une fonction de hachage, qui permet de crypter le mot de passe avant de le stocker en base de données.

Nous l'avons utilisé pour modifier les mots de passe avant de les stocker. Par exemple, lorsqu'on crée un nouvel utilisateur:

```
// CREATING CRYPTED PASSWORD WITH BCRYPT
const saltRound = 10;
const salt = await bcrypt.genSalt(saltRound);
const bcryptPassword = await bcrypt.hash(password, salt);

// CREATING NEW USER IN DATABASE
const newMember = await authDataMapper.insertMember({
  pseudo,
  email,
  password: bcryptPassword,
});
```

On choisit un nombre de tours (ici 10), puis on crypte le mot de passe, et enfin on ajoute un nouveau membre dans la base de données avec le mot de passe nouvellement haché.

Lorsqu'un visiteur va ensuite essayer de se connecter, on va venir comparer le mot de passe qu'il vient d'entrer à celui qui correspond à l'email qu'il a entré dans la base de données ainsi:

```
// COMPARING IF PASSWORD IN DATABASE IS THE SAME AS THE ONE TYPED
const correctPassword = await bcrypt.compare(password, member.password);

// IF PASSWORD IS INCORRECT THROW 401 STATUS
if (!correctPassword) {
  return res.status(401).json('Email ou mot de passe incorrect.');
}
```

On les compare grâce à `bcrypt.compare()`, puis s'ils sont différents, on envoie un message d'erreur.

Ainsi, le mot de passe ajouté à la création ou modification du compte n'est jamais révélé, reste toujours haché et n'est jamais stocké tel quel dans la base de données, ce qui le garde en sécurité.

## JWT:

Pour que les utilisateurs puissent s'authentifier, nous avons utilisé **JSON Web Token (JWT)**. JWT est un jeton permettant d'échanger des informations de manière sécurisée. Il offre une authentification stateless.

Lorsqu'un visiteur se connecte avec succès, on crée un jeton, basé sur l'id du membre:

```
res.json({ member: member.email, pseudo: member.pseudo, token: jwtGenerator(member.id), member_id: member.id });
```

De cette manière:

```
// REQUIRE JSON WEB TOKEN
const jwt = require('jsonwebtoken');
// REQUIRE DOTENV TO HAVE ACCESS TO OUR ENVIRONMENT VARIABLES
require('dotenv').config();

function jwtGenerator(member_id) {
    // CREATING THE PAYLOAD
    const payload = {
        // SENDING THE DATA AKA MEMBER ID
        id: member_id,
    };
    // RETURN THE TOKEN AND SIGNING IT
    return jwt.sign(payload, process.env.JWT_TOKEN, { expiresIn: 60 * 60 });
}

module.exports = jwtGenerator;
```

Puis on le stocke dans le state. Ainsi, dès qu'un utilisateur du site veut accéder à une page ou faire une action qui nécessite d'être connecté, on passe le token stocké dans le state en argument de la requête Axios en front:

```
const config = {
  headers: {
    "Bearer": `${store.getState().status.token}`,
    "Accept": "application/json",
    "Content-Type": "application/json",
  },
}
```

Puis on vérifie qu'il y ait bien un token valide dans son state:



```
const jwt = require('jsonwebtoken');
require("dotenv").config();

module.exports = async (req, res, next) => {
  try {
    const jwtToken = req.header("Bearer");

    if (!jwtToken) {
      return res.status(403).json("You are not authorized dammit.");
    }

    const payload = jwt.verify(jwtToken, process.env.JWT_TOKEN);

    req.member = payload;

    next();
  } catch (error) {
    console.error(error);
    return res.status(403).send("You are not authorized.");
  }
};
```

Le token étant unique pour chaque utilisateur, il permet également de faire des actions spécifiques à ce dernier. Par exemple, l'utilisateur peut ajouter un programme à sa watchlist, et c'est grâce à son token que l'on va savoir de quel utilisateur nous allons alimenter la watchlist.

Ainsi, les token sécurisent l'application en permettant l'authentification et en vérifiant pour chaque action si elle vient d'un utilisateur connecté, et de quel utilisateur elle vient.

## XI. Difficultés rencontrées

Premièrement, lorsque j'ai créé la fonctionnalité Picky Find, qui permet à l'utilisateur d'écrire dans la barre de recherche et de trouver des programmes qui correspondent à ce qu'il a écrit, j'avais dans un premier temps voulu lancer la recherche à partir de 3 caractères écrits et en relancer une à chaque changement, dès que l'utilisateur écrivait ou effaçait un caractère.

Malheureusement, la requête et l'affichage des résultats prenaient beaucoup trop de temps, et le temps que l'utilisateur écrive le(s) terme(s) entier(s), la recherche sur les 3 premières lettres était encore en train de se faire, et le résultat qui s'affichait à la fin ne correspondait donc pas du tout à ce que l'utilisateur voulait. Le résultat souhaité n'était donc pas du tout atteint.

J'ai donc décidé de changer mon approche au problème, et de remplacer cette recherche dynamique par une unique recherche à chaque fois que l'utilisateur appuie sur entrée.

Ensute, je voulais que l'utilisateur puisse scroller sur les cartes des résultats directement, en ne faisant bouger que ces éléments là et en laissant les autres parties de la page (header, titre de la page et footer) statiques. Bien que j'aie réussi à mettre un scroll sur les cartes, il était très difficile de le faire fonctionner et la plupart du temps ça ne réagissait pas.

J'ai donc décidé de changer de solution: j'ai enlevé le scroll sur cette partie de la page et je l'ai mis sur la page entière. Néanmoins, ce faisant, lorsque l'utilisateur faisait défiler la page, il ne voyait plus le header, et prenait parfois du temps à retourner en haut de la page en la faisant défiler en sens inverse. J'ai donc créé un bouton en bas à droite de la page qui ne s'affiche que lorsque l'utilisateur a déjà fait défiler un peu la page vers le bas, et j'ai laissé le footer visible tout le long pour changer facilement de page. Ainsi, lorsque l'utilisateur a fait défiler des résultats, il peut cliquer sur un des deux liens du footer, ou cliquer sur le bouton pour retourner tout en haut de la page et pouvoir accéder à toutes les autres pages.

Enfin, le fait d'utiliser une API tierce (en l'occurrence celle de Beta Series) a créé quelques problèmes. En effet, le fait d'utiliser une API tierce signifie nécessairement qu'il y a des limites à ce que l'on peut faire puisqu'on ne choisit pas la manière d'accéder aux données. Il faut donc savoir s'adapter et faire avec les moyens du bord.

Par exemple, pour les résultats de Picky Mood, nous voulions qu'à chaque recherche, 20 résultats maximum s'affichent, de manière totalement aléatoire, parmi une grande quantité de programmes correspondants aux critères sélectionnés, permettant ainsi à l'utilisateur d'avoir chaque fois des résultats différents même s'il faisait de nombreuses fois la même recherche.

Malheureusement, l'API ne renvoyant que 20 résultats maximum par genre sélectionné, et surtout renvoyant toujours les mêmes résultats pour une même recherche, il était impossible d'arriver au résultat voulu.

J'ai donc fait en sorte d'afficher 20 programmes aléatoires parmi ceux qu'on récupérait. Cela veut dire que malheureusement si l'utilisateur fait plusieurs fois la même requête avec un seul genre, il retombera à chaque fois sur les mêmes programmes. Mais si l'utilisateur fait plusieurs fois la même requête avec 3 genres par exemple, 20 programmes seront sélectionnés aléatoirement parmi les 60 programmes maximums récupérés, ce qui fera quand même une différence à chaque recherche.

## XII. Conclusion

Ce mois d'apothéose a conclu en beauté la formation de O'Clock. Au début de ces six mois, je n'avais que de vagues notions de ce qu'était le développement web, et quand je regarde en arrière, je suis impressionnée par la progression faite grâce aux enseignants d'O'Clock, et par le fait même d'avoir pu créer un projet comme Picky. La création de cette application web a été tellement enrichissante, tant du point de vue relationnel que de celui de l'apprentissage.

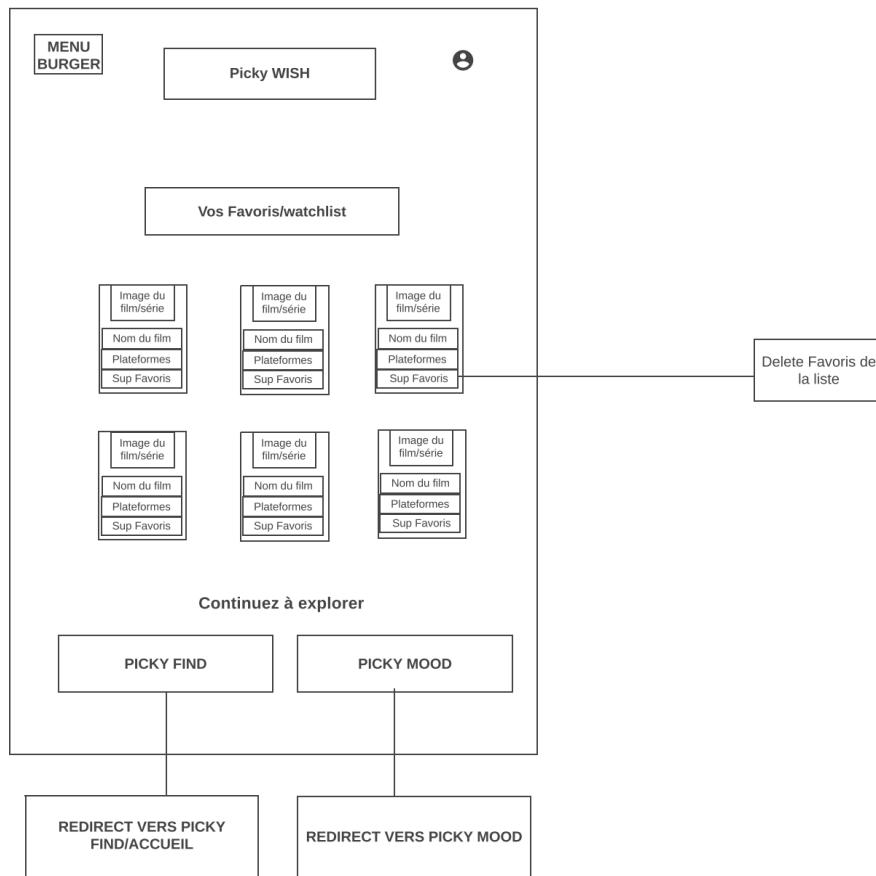
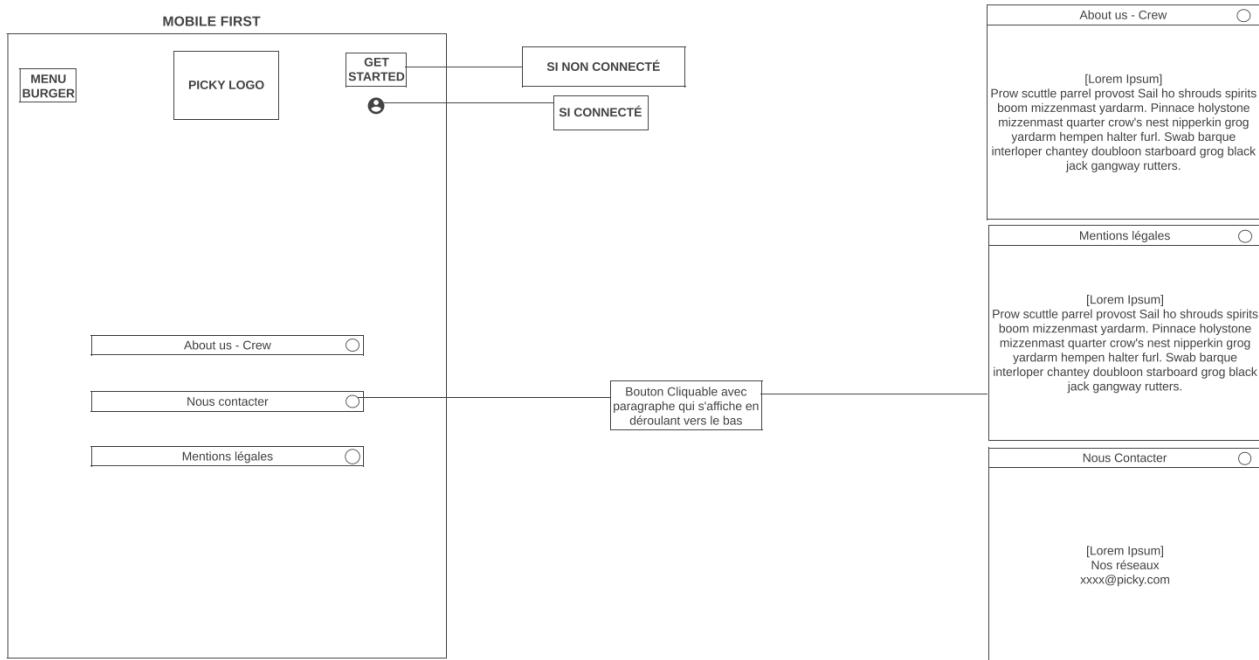
Nous avons su nous organiser et communiquer dès le départ pour progresser rapidement et dans la bonne direction. Chacun laissait les autres s'exprimer et prendre part librement à chaque étape du projet, chacun a trouvé sa place, et les rôles ont été répartis équitablement. Cela a également été une vraie leçon du point de vue de l'entraide. Le fait d'avoir quelqu'un vers qui se tourner lorsque l'on est bloqué permet de progresser beaucoup plus vite, et le fait d'aider et d'expliquer à quelqu'un d'autre est également très enrichissant et permet soi-même de mieux comprendre chaque concept, d'aller plus loin.

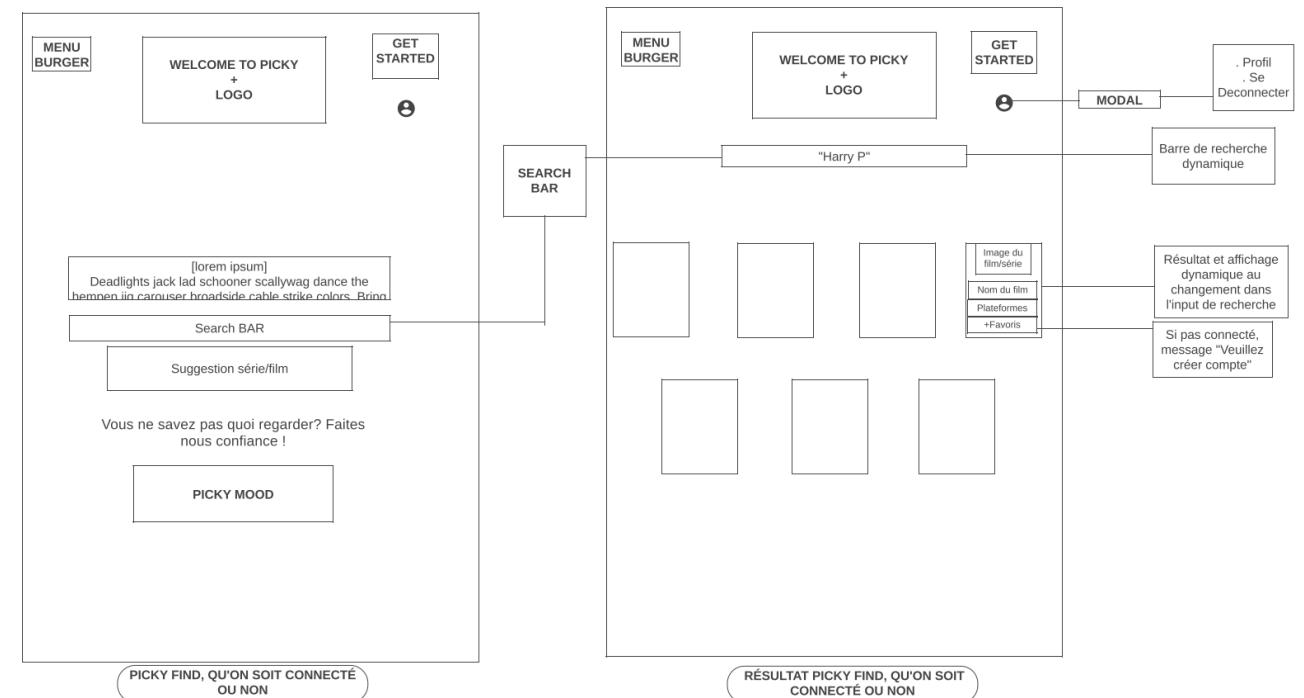
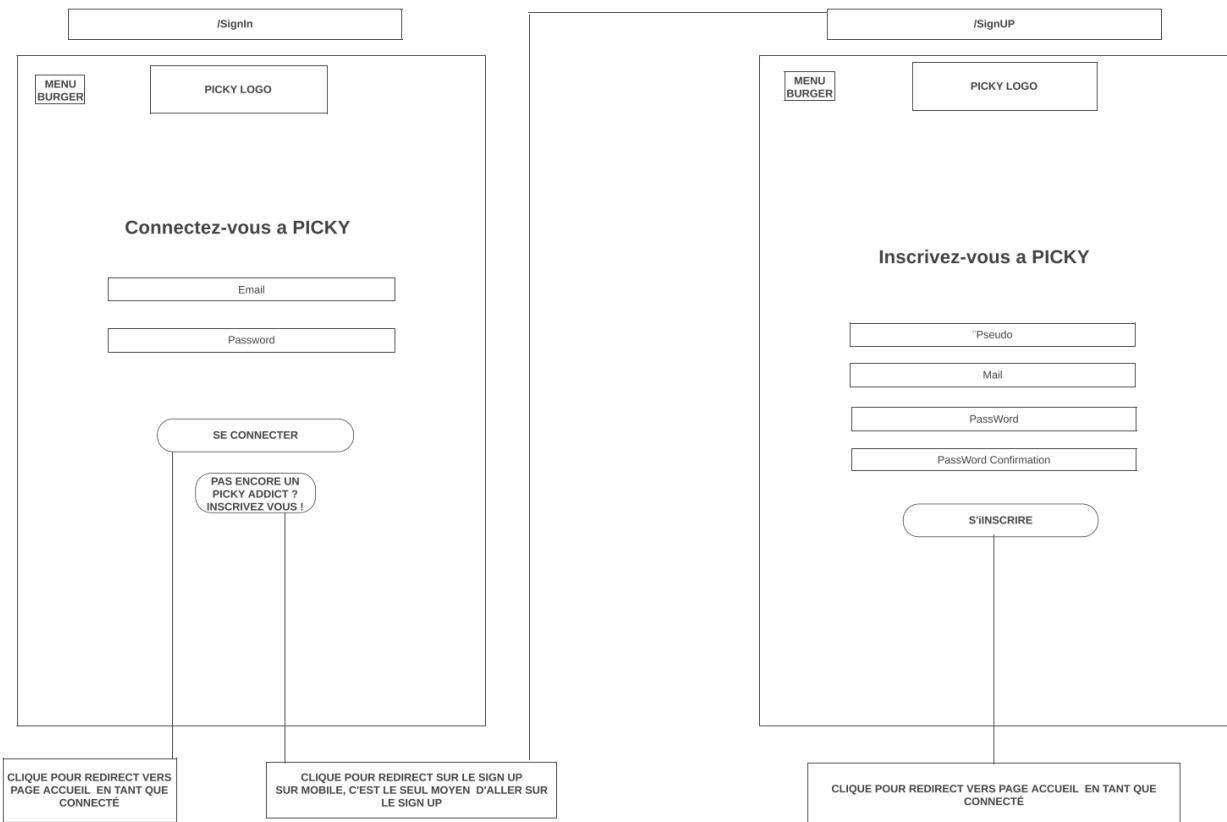
Grâce à Picky, j'ai pu mettre en pratique tout ce que j'avais appris lors de la formation, mais cela m'a également fait approfondir et progresser sur les notions vues, et appris de nouvelles choses. En travaillant avec React et Redux sur un projet réel et aussi important, j'ai pu vraiment apprendre à mieux maîtriser leur utilisation, et je me sens maintenant capable de les utiliser pour créer d'autres applications. J'ai aussi eu l'occasion de beaucoup m'entraîner sur le CSS avec Sass, en créant des visuels à partir d'idées que l'on a eu nous-mêmes, donc en choisissant tout le design et le placement des éléments, ce qui a été très enrichissant. J'ai également pu en apprendre plus sur le backend grâce aux autres et en participant également à sa création. Ensuite, j'ai appris à utiliser Git, technologie dont je ne connaissais pas grand-chose. Enfin, ce mois m'a enseigné beaucoup de choses sur l'organisation sur un projet réel, et sur le travail en équipe.

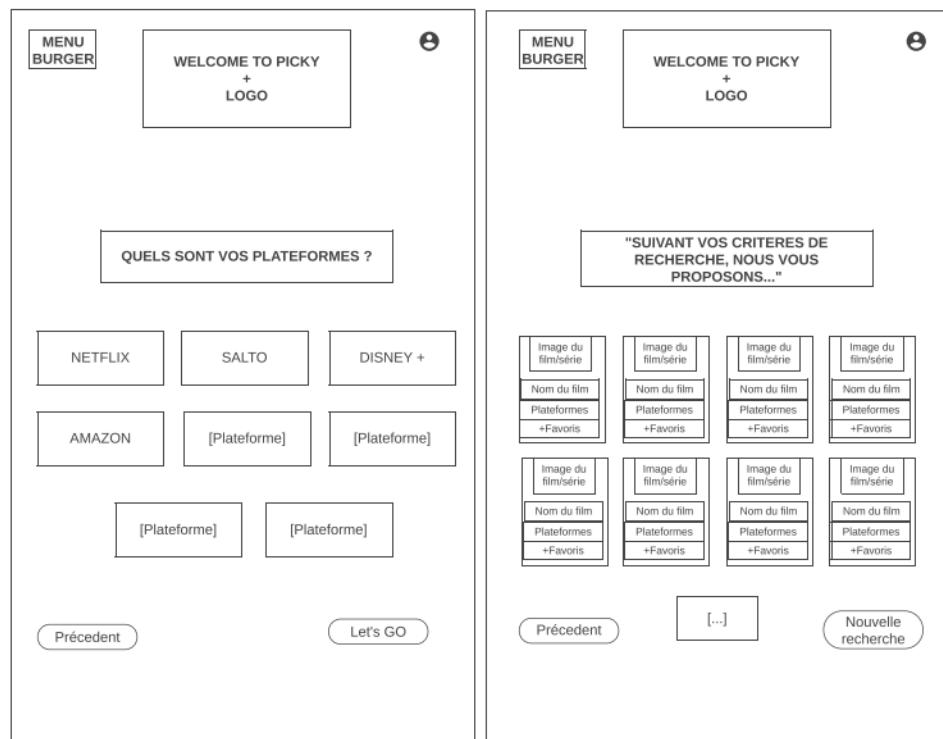
Cette expérience m'a appris sur moi-même et sur le développement web. Au départ, le fait de créer une application à partir de zéro me semblait être un travail démesuré, et je ne savais pas du tout par où commencer et dans quel ordre faire les choses. Ce mois d'apothéose m'a prouvé que c'était possible, et je me sens maintenant beaucoup plus préparée à créer d'autres applications et à démarrer dans le métier du développement web.

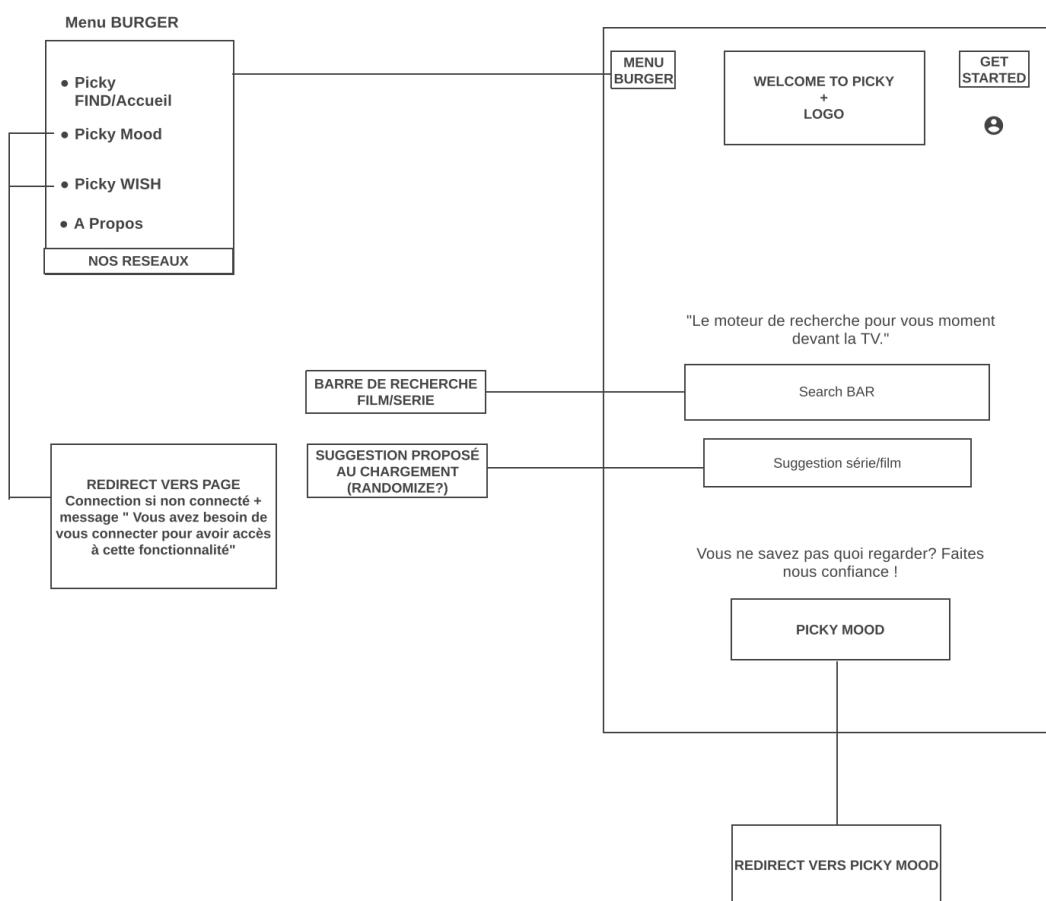
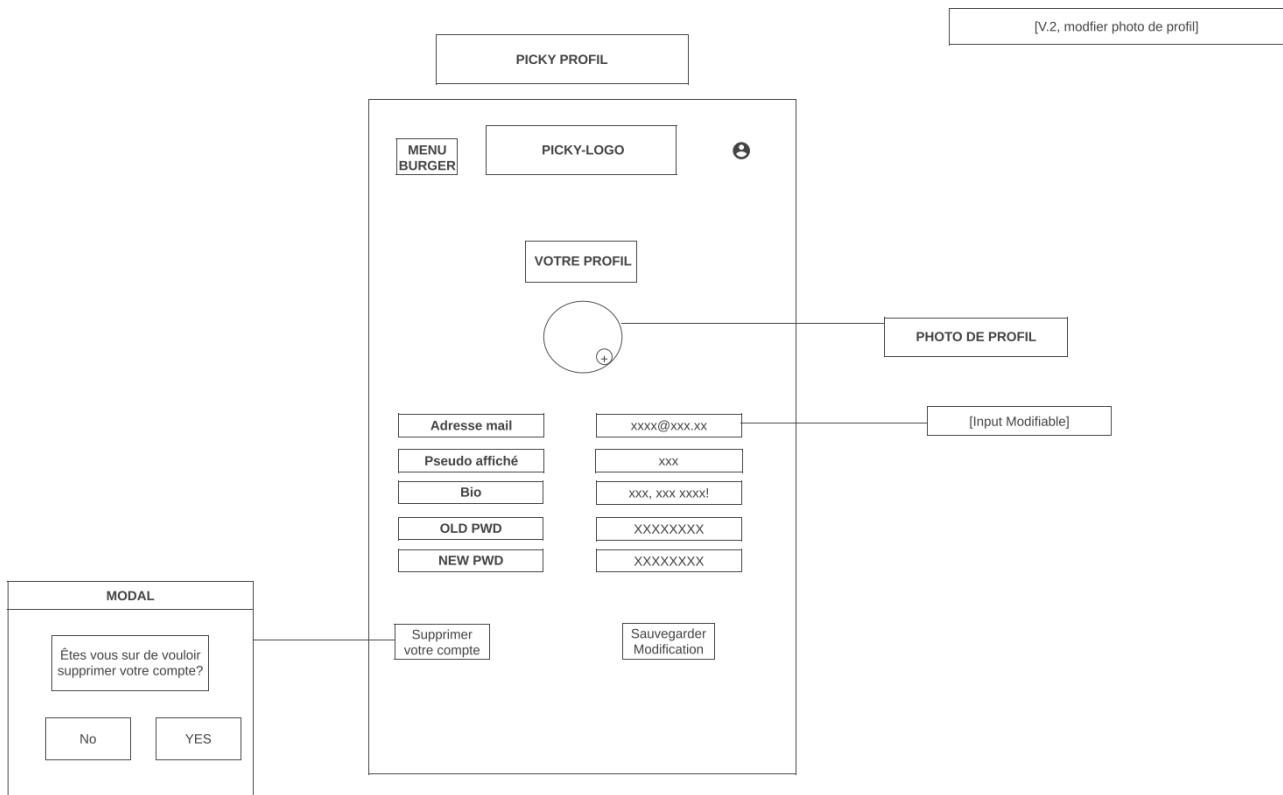
## XIII. Annexes

### A. Wireframes mobile









## B. Wireframes desktop

