NETFILMS

Sommaire :

Cahier des charges et fonctionnalités

- L'application doit pouvoir permettre à un utilisateur de consulter la page principale, la liste des films et le détail d'un film quelque que soit l'état de connexion (connecté ou non).

- Lors d'une inscription l'utilisateur doit être autolog si la procédure de création de compte se déroule correctement.

- lorsqu'il est connecté l'utilisateur peut, sur le détail d'un film accéder à la liste des commentaires sur ce même film.

- lorsqu'il est connecté l'utilisateur peut, ajouter un commentaire et le supprimer/modifier si il en est l'auteur.

- lorsqu'il est connecté l'utilisateur vois au moyen d'une étoile les films ajouté en favoris via la liste des films. Il peut cocher ou décocher un film pour le mettre/retirer des favoris.

- lorsqu'il est connecté l'utilisateur peut aller dans son onglet favoris pour voir uniquement ses films ajouté en favoris. Il peut en supprimer individuellement ou tous les supprimer.

Architecture de la BDD

- Stack: mongoDB

```
const MOVIE_MODEL = new Schema(
  {
    adult: Boolean,
    backdrop_path: String,
    genre_ids: Array,
    original_language: String,
    original_title: String,
    overview: String,
    popularity: Number,
    poster_path: String,
    release_date: Date,
    title: String,
    vote_average: Number,
    vote_count: Number,
  },
  { timestamps: true }
);
```

```
const USER_MODEL = new Schema(
  {
    firstname: { type: String, required: true },
    lastname: String,
    email: { type: String, required: true, unique: true },
    password: { type: String, required: true },
    favorites: [{ type: Schema.Types.ObjectId, ref: "movies" }],
  },
  { timestamps: true }
);
```

```
const COMMENT_MODEL = new Schema(
  {
    userId: { type: Schema.Types.ObjectId, ref: "users" },
    movieId: { type: Schema.Types.ObjectId, ref: "movies" },
    text: String,
    isDeleted: Boolean,
  },
  { timestamps: true }
);
```

Architecture back-end

- Stack: Express, mongoose,

- Base de donnée hebergé sur Atlas

- Api de type rest découpée selon un routing imbriqué, ou ces même routes appellent des controllers qui eux même peuvent faire appel à différentes query, qui elles même appellent des méthodes des modèles mongoose définis.

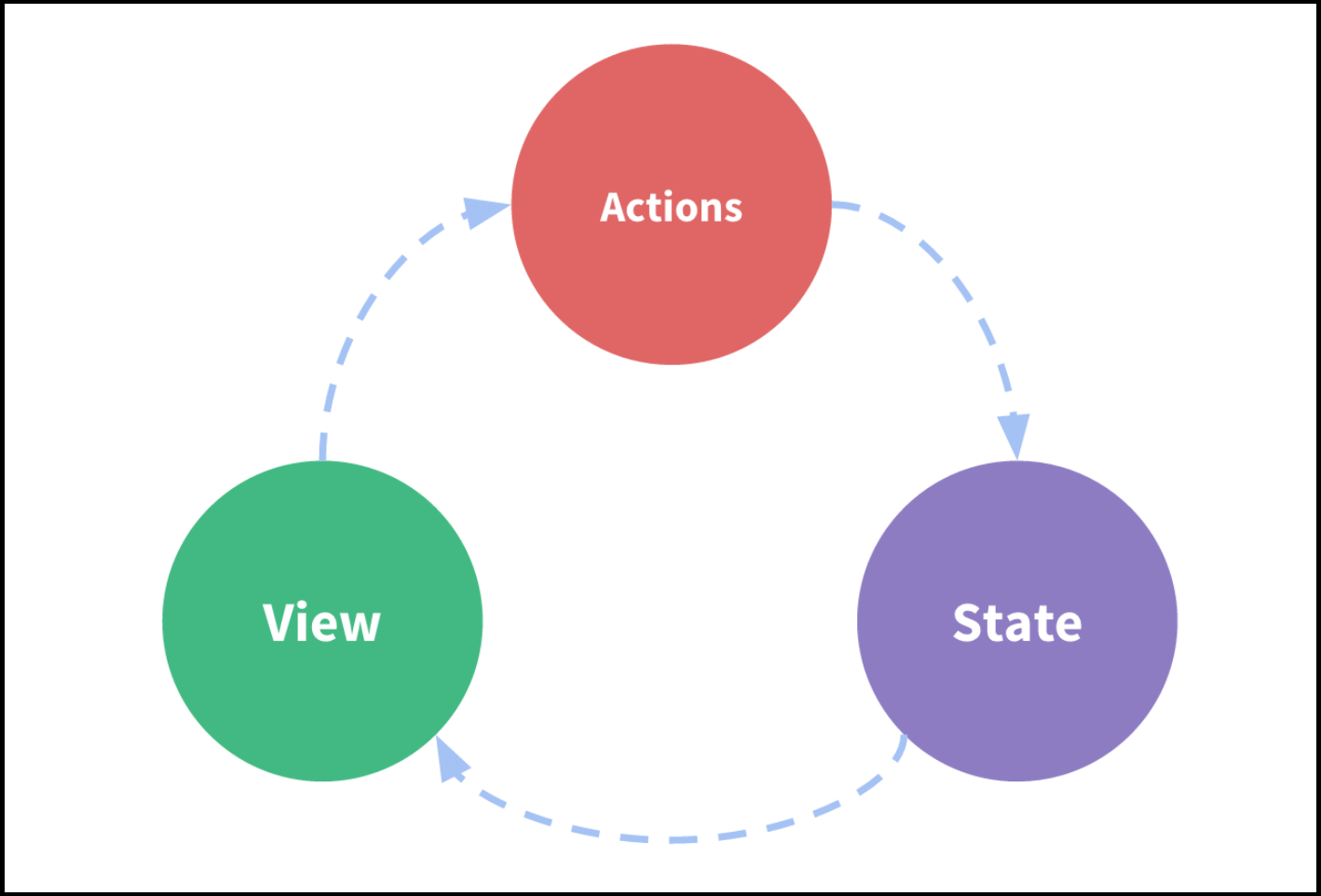
- Middleware en amont de toute les routes pour récupérer l'utilisateur connecté si il y en a un, et le placer sur l'objet req fourni par express.

- Middleware de protection sur les routes préfixés "/private" qui donne a accés a ces routes uniquement si un JWT valide et non modifié est fourni dans le header de la requète.

- Hash du mot de passe via la librairie BCRYPT, avec un salt de 10 round pour éviter des pertes de performance trop importantes.

Architecture front-end

Stack: React, Redux,



- Architecture avec redux, MVC, afin de pouvoir gérer l'état applicatif a tout moment pour tous les type de données de l'application sans passer par l'API context et ainsi éviter des render inutiles.

- Protection du react router afin d'éviter de pouvoir accéder à des pages nécessitant un accés connecté, en faisant un simple rewrite d'url.

- Architecture scss, chaque élément de l'interface a son fichier scss dédié, tout importé dans un app.scss. Utilisation de mixins et de règles générale afin d'éviter la redondance dans les fichiers scss, utilisation de selector BEM afin de faciliter la compréhension du code.