

R 5.10 – Base de données

**Projet de NoSQL
2024-2025
Trailerz**

Jean-Jacques VIALE

Luca VIZIO

Valentin LUTHRINGER

Table des matières

1. Présentation générale	3
2. Présentation de la base de données	3
2.1 Source de la base	3
2.2 Structure de la base.....	3
2.3 Type de la base.....	6
3. Architecture du code	6
3.1 Choix des technologies.....	6
3.2 Structure du code.....	6
4. Fonctionnalités développées.....	7
4.1 Fonctionnement du carrousel Aléatoire	8
4.2 Fonctionnement des carrousels de Tops par Catégorie	9
4.3 Fonctionnement du top 1 des tendances	10
4.4 Fonctionnement de la page des derniers films sortis.....	11
4.5 Fonctionnement de la page de Recherche.....	12
4.6 Fonctionnement de la page de génération automatique de film (Random)	13
4.7 Fonctionnement de la page d'affichage du Film	14
4.8 Fonctionnement de l'ajout de films	15
4.9 Fonctionnement de la modification de film	16
4.10 Fonctionnement de la suppression de film	17
4.11 Fonctionnement de la gestion du film en tendance	18
4.12 Fonctionnement de la récupération d'un film, pour le jeu « Devine le film »	19
4.13 Fonctionnement de la recherche et vérification d'un film, pour le jeu « Devine le film »	20

1. Présentation générale

Dans le cadre de la ressource R5.10 – Nouveaux paradigmes de base de données, la réalisation d'un site web mettant en œuvre une interaction avec une base de données de notre choix était de rigueur. Nous avons opté pour la réalisation d'un site de bande-annonce mettant en avant un très grand nombre de films, toute année confondue, et comprenant de nombreuses fonctionnalités. Ce site est en référence à IMDB, d'où nous avons récupéré la base de données.

2. Présentation de la base de données

2.1 Source de la base

Nous avons téléchargé notre base de données sur le site de datasets gratuits « Kaggle » car il comprend un très grand nombre de base de données riches en valeurs et facilement importables dans n'importe quel SGBD.

Le lien de téléchargement : <https://www.kaggle.com/datasets/rezaunderfit/48k-imdb-movies-data?resource=download>

2.2 Structure de la base

La base de données que nous avons utilisée se nomme « Movies », elle comporte 21 champs, 48650 films, et son schéma se dresse de la manière suivante :

```
{
  "movie": {
    "url": "String",
    "name": "String",
    "image": "String",
    "genre": "List<String>",
    "contentRating": "String",
    "actors": [
      {
        "name": "String",
        "url": "String"
      }
    ],
    "director": {
      "name": "String",
      "url": "String"
    },
    "creator": [
      {
        "name": "String",
        "url": "String"
      }
    ],
    "description": "String",
    "datePublished": "String",
    "keywords": "List<String>",
    "aggregateRating": {
      "ratingCount": "int",
      "bestRating": "int",
```

```

    "worstRating": "int",
    "ratingValue": "float"
  },
  "review": {
    "author": "String",
    "dateCreated": "String",
    "language": "String",
    "title": "String",
    "body": "String",
    "rating": "int"
  },
  "duration": "String",
  "trailer": {
    "name": "String",
    "embedUrl": "String",
    "thumbnail": {
      "contentUrl": "String"
    },
    "description": "String",
    "uploadDate": "String"
  }
}

```

Voici un exemple de schéma de donnée :

```

{
  "@context": "http://schema.org",
  "@type": "Movie",
  "url": "/title/tt4532038/",
  "name": "The War with Grandpa",
  "image": "https://m.media-
amazon.com/images/M/MV5BNTlkZDQ1ODEtY2ZiMS00OGNhLWJlZDctYzY0NTFmNmQ2NDAzXkEyXkFqcGd
eQXVyMTkxNjUyNQ@@.V1.jpg",
  "genre": ["Comedy", "Drama", "Family"],
  "contentRating": "PG",
  "actor": [{
    "@type": "Person",
    "url": "/name/nm0000235/",
    "name": "Uma Thurman"
  }],
  "director": {
    "@type": "Person",
    "url": "/name/nm0384722/",
    "name": "Tim Hill"
  },
  "creator": [{
    "@type": "Person",
    "url": "/name/nm0040022/",
    "name": "Tom J. Astle"
  }],
  "description": "The War with Grandpa is a movie starring Robert De Niro, Uma Thurman, and Rob Riggle.
Upset that he has to share the room he loves with his grandfather, Peter decides to declare war in an attempt
to get it back.",
  "datePublished": "2020-08-27",
  "keywords": "mother son relationship,christmas,room,family conflict,family relationships",

```

```

"aggregateRating": {
  "@type": "AggregateRating",
  "ratingCount": 9310,
  "bestRating": "10.0",
  "worstRating": "1.0",
  "ratingValue": "5.5"
},
"review": {
  "@type": "Review",
  "itemReviewed": {
    "@type": "CreativeWork",
    "url": "/title/tt4532038/"
  },
  "author": {
    "@type": "Person",
    "name": "byron-116"
  },
  "dateCreated": "2020-08-28",
  "inLanguage": "English",
  "name": "Suitable for juveniles only...",
  "reviewBody": "It's pathetic to watch such great stars in this film apt for juveniles only. Watch it if you are under 14 years old.....",
  "reviewRating": {
    "@type": "Rating",
    "worstRating": "1",
    "bestRating": "10",
    "ratingValue": "4"
  }
},
"duration": "PT1H34M",
"trailer": {
  "@type": "VideoObject",
  "name": "Official Trailer",
  "embedUrl": "/video/imdb/vi911785497",
  "thumbnail": {
    "@type": "ImageObject",
    "contentUrl": "https://m.media-amazon.com/images/M/MV5BMTdhNWl1N2QtMjQ5Yi00M2M5LWE3YWQtMDE5YmNhMmFmZTVkXkEyXkFqcGdeQXVyYW5zY29kZS13b3JrZmxvdw@@.V1.jpg"
  },
  "thumbnailUrl": "https://m.media-amazon.com/images/M/MV5BMTdhNWl1N2QtMjQ5Yi00M2M5LWE3YWQtMDE5YmNhMmFmZTVkXkEyXkFqcGdeQXVyYW5zY29kZS13b3JrZmxvdw@@.V1.jpg",
  "description": "The next big family-fun film is hitting theaters soon! Check out the trailer for THE WAR WITH GRANDPA starring Robert De Niro, Christopher Walken, Uma Thurman, Rob Riggle, Cheech Marin, Laura Marano and Oakes Fegly. Coming soon to theaters!",
  "uploadDate": "2020-08-13T17:40:20Z"
}
}

```

2.3 Type de la base

Étant donné que nous avons opté pour un sujet de gestion de contenu, plus spécifiquement d'une collection de film, nous avons décidé d'établir une base de données orientée document, avec le SGBD MongoDB.

MongoDB est un bon choix pour une base de données telle que la nôtre, qui traite un très grand nombre de trailers et d'informations sur des films provenant d'IMDB, en raison de :

1. **Son modèle flexible** : Les films ont des données variées (acteurs, genres, avis...) qui peuvent être facilement stockées dans des documents JSON, avec des structures différentes selon les films.
2. **Sa gestion des relations** : Il permet de gérer des relations complexes entre entités (films, acteurs, avis...) par des références ou l'intégration de sous-documents.
3. **Ses performances** : qui sont élevées pour les lectures et écritures, essentielles dans un système dynamique comme la base IMDB que nous exploitons.

3. Architecture du code

3.1 Choix des technologies

Nous avons décidé de gérer notre backend grâce à un serveur Node.js, notre frontend avec HTML/CSS et la liaison backend/frontend avec Javascript.

Ce choix a été fait d'une part d'un point de vue pratique (Node.js est simple d'utilisation), et d'autre part d'un point de vue technique. En effet, l'un des avantages principaux de Node.js est sa rapidité d'exécution et de recherche grâce aux API. Les API avec Node.js sont flexibles et efficaces grâce à leurs modèles **non-bloquants** et **asynchrones**, permettant de traiter plusieurs requêtes simultanément sans ralentir.

3.2 Structure du code

Notre code est organisé de la manière suivante :

- Le dossier « **models** » contenant le fichier « Movie.js », qui est le modèle de la base de données, soit l'ensemble de la structure de données dont nous avons besoin pour nos fonctionnalités.
- Le dossier « **fonctionnalités_js** » contient l'intégralité de la partie Backend. C'est dans ce dossier que l'ensemble des fichiers permettant la création des API (donc des fonctionnalités) sont créés. Chaque fonctionnalité comporte sa propre API (avec sa propre route) et chaque API contient son fichier « .js ». Chaque API récupère les données/entrées dont elle a besoin grâce à Movie.js, puis ces données sont utilisées sous formes de diverses requêtes. Les API sont de la forme « /api/nomApi ».
- Le dossier « **liaisonBackFront** » qui contient le code de liaison backend / frontend. Il contient le code permettant l'interrogation des API et l'affichage de leur résultat.
- Le dossier « **template_api_front** » contenant l'intégralité des pages html simples. Elles communiquent aussi avec les dossiers « **css** » et « **resources** » qui contiennent du code de design et les images natives.
- Le fichier « **top_tendance.json** » qui permet la modification rapide du film à la une. Il est envoyé à une API qui va déterminer grâce au nom, l'ID du film pour le retrouver dans la BD

automatiquement. Il est également possible d'y placer une image d'illustration et un logo de film (c'est un builder manuel).

- Et enfin, le fichier **index.js**. C'est lui qui va lier les routes du serveur aux pages html. Il va également démarrer le serveur Node JS, appeler et connecter la BD, appeler le modèle Movie.js, charger les fonctionnalités souhaitées (APIs) et rester tout le temps ouvert.

4. Fonctionnalités développées

Afin de donner aux visiteurs du site Trailerz une expérience optimale, nous avons réalisé les fonctionnalités suivantes :

- **Sur la page d'accueil :**
 - Récupération d'une sélection de 12 films totalement aléatoires.
 - Récupération des 12 films les mieux notés de la catégorie « Action » en fonction de leur note et du nombre de votes.
 - Récupération des 12 films les mieux notés de la catégorie « Comédie » en fonction de leur note et du nombre de votes.
 - Récupération des 12 films les mieux notés de la catégorie « Guerre » en fonction de leur note et du nombre de votes.
 - Récupération des 12 films les mieux notés de la catégorie « Horreur » en fonction de leur note et du nombre de votes.
- **Sur la page Films :**
 - Affichage de tous les films de la base de données du plus récent au plus ancien.
- **Sur la page Recherche :**
 - Possibilité de rechercher un film par son nom
 - Par sa date (curseur d'encadrement)
 - Par son (ou ses) genre(s)
 - Par sa durée
 - Par tous ces filtres en même temps / certains de ces filtres en même temps
- **Sur la page Random :**
 - Possibilité de récupérer au hasard un film de la BD en sélectionnant un ou plusieurs genres
- **Sur la page Jeux :**
 - Le but est de retrouver le nom du film à partir de l'image Thumbnail du film. A chaque appel à la fonctionnalité, le film est aléatoirement choisi et l'utilisateur peut rechercher dans la zone prévue à cet effet le nom du film auquel il pense, une liste de 5 films ayant un nom similaire s'affichera en temps réel. Si l'utilisateur a faux, il peut recommencer, s'il a juste, une autre image de film aléatoirement choisie est piochée.
- **Sur la page Admin :**
 - Possibilité d'ajouter un film (en entrant le nom, la date, les genres, la durée, l'URL IMDB, l'URL de l'affiche, des mots clés, la description, les acteurs, directeurs et créateurs).
 - Possibilité de modifier un film (en entrant le nom, la date, les genres, la durée, l'URL IMDB, l'URL de l'affiche, des mots clés, la description, les acteurs, directeurs et créateurs).
 - Possibilité de supprimer un film en entrant l'ID du film.
 - Possibilité de modifier le film numéro un des tendances (sur l'accueil)

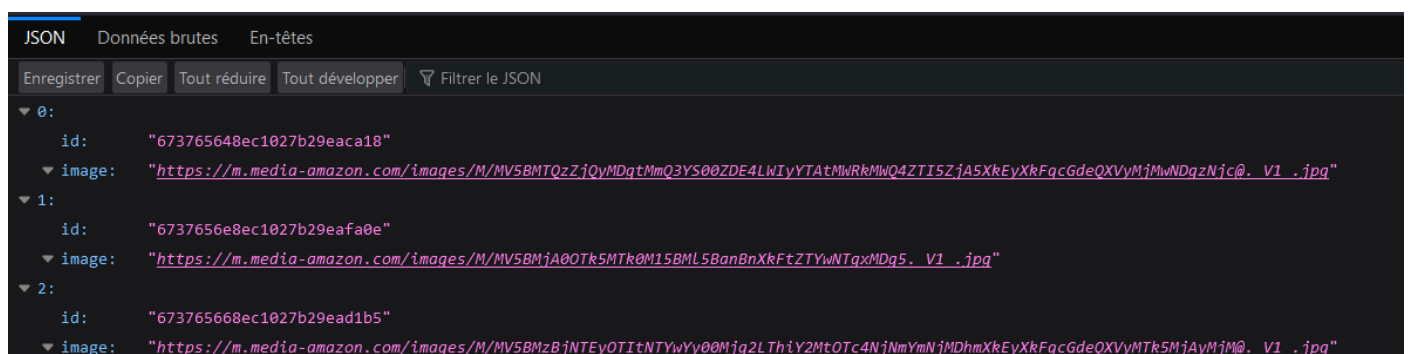
4.1 Fonctionnement du carrousel Aléatoire



Objectif : Afficher une sélection totalement aléatoire de 12 films.

Fonctionnement backend : L'API fournit 12 affiches de films aléatoires. Elle sélectionne des films au hasard dans la base de données, vérifie si l'URL de l'image associée est valide et retourne l'image ou une image par défaut si l'URL est invalide. La réponse est envoyée sous forme de tableau d'objets contenant l'ID du film et l'URL de l'image.

Aperçu JSON brut :



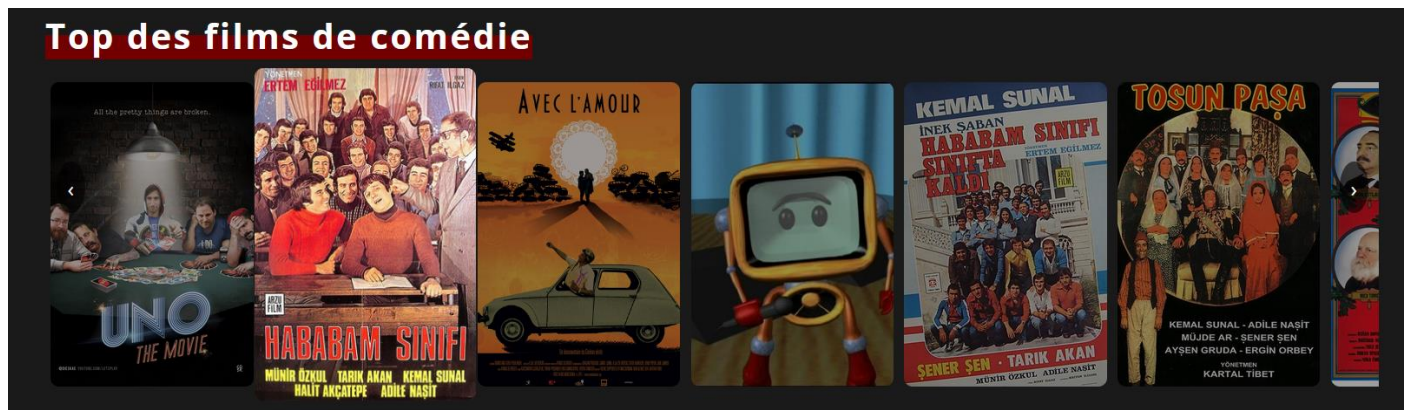
Liaison au frontend : Le script JS appelle l'API et envoie les images dans les div créées sur le template. Plus précisément, lors du chargement de la page (DOMContentLoaded), il effectue une requête HTTP à l'API `/api/carrousel-random` pour récupérer les données des films (avec l'ID et l'image de chaque film). Ensuite, il vide le conteneur du carrousel (`randomCarousel.innerHTML = ""`) et ajoute dynamiquement les affiches de films sous forme de liens, chaque lien redirigeant vers une page détaillée du film. Enfin, il recalcule la largeur du carrousel à l'aide de la fonction `updateCarouselWidth`.

Exemple de requête : <http://localhost:2030/api/carrousel-random>

Fichier backend : `/fonctionnalités_js/carrousel_random.js`

Fichier liaison : `/liaisonBackFront/accueil.js`

4.2 Fonctionnement des carrousels de Tops par Catégorie



Objectif : Afficher les 12 films les mieux notés par catégorie.

Fonctionnement backend : L'API affiche les jaquettes des 12 films les mieux notés et les plus votés dans une catégorie spécifiée via le paramètre `category` de l'URL. Elle filtre les films par genre, vérifie l'existence des données de notation, trie les résultats par note décroissante et nombre de votes, et retourne un tableau d'objets contenant l'ID, l'image, la note et le nombre de votes de chaque film. Si une image est invalide, une image par défaut est utilisée.

Aperçu JSON brut :

```
▼ 0:
  id: "673765698ec1027b29eadfed"
  ▼ image: "https://m.media-amazon.com/images/M/MV5BMjk5ODZjYmMtYTJjNy00MTU2LWI5OTYtYTQ5YjFLMDk3ZjI0XkEyXkFqcGdeQXVyODAyNDE3Mw@@. V1 .jpg"
  rating: 9.3
  ratingCount: 14512
▼ 1:
  id: "673765758ec1027b29eb223f"
  ▼ image: "https://m.media-amazon.com/images/M/MV5BMWY3NTIjMjEtYzRiMi00NWJMLTkzNjItZTVmZjE0MTdjMjhlL2LTYWdLL2LTYWdLXkEyXkFqcGdeQXVyNTQ4NTc5OTU@. V1 .jpg"
  rating: 9.1
  ratingCount: 797212
```

Liaison au frontend : Lors du chargement de la page (DOMContentLoaded), le code appelle la fonction `fetchAndDisplayMovies` pour chaque catégorie (Horror, Action, Comedy, War) en envoyant une requête à l'API `/api/carrousel-bestrategie-cat`. Chaque catégorie est liée à un carrousel spécifique identifié par son ID. Pour chaque catégorie, les données des films (ID, image, etc.) sont récupérées et utilisées pour générer dynamiquement les éléments HTML correspondants dans le carrousel. Une fois les jaquettes affichées, la fonction `updateCarouselWidth` gère la largeur des carrousels et configure les boutons pour naviguer entre les affiches.

Exemple de requête : <http://localhost:2030/api/carrousel-bestrategie-cat?category=Horror>

Fichier backend : `/fonctionnalités_js/ carrousel_bestrategie_cat.js`

Fichier liaison : `/liaisonBackFront/accueil.js`

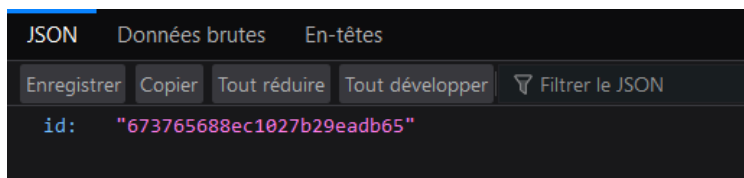
4.3 Fonctionnement du top 1 des tendances



Objectif : Afficher le top 1 des tendances du moment sur la page d'accueil

Fonctionnement backend : L'API permet de trouver l'ID d'un film à partir de son nom. Lorsqu'un utilisateur envoie une requête GET à `/api/trouver-id` avec le paramètre `nom`, l'API vérifie si le paramètre est fourni. Ensuite, elle recherche dans la base de données un film correspondant au nom donné, en ignorant la casse grâce à une expression régulière. Si le film est trouvé, elle renvoie son ID sous forme d'objet JSON. Sinon, un message d'erreur apparaît.

Aperçu JSON brut :



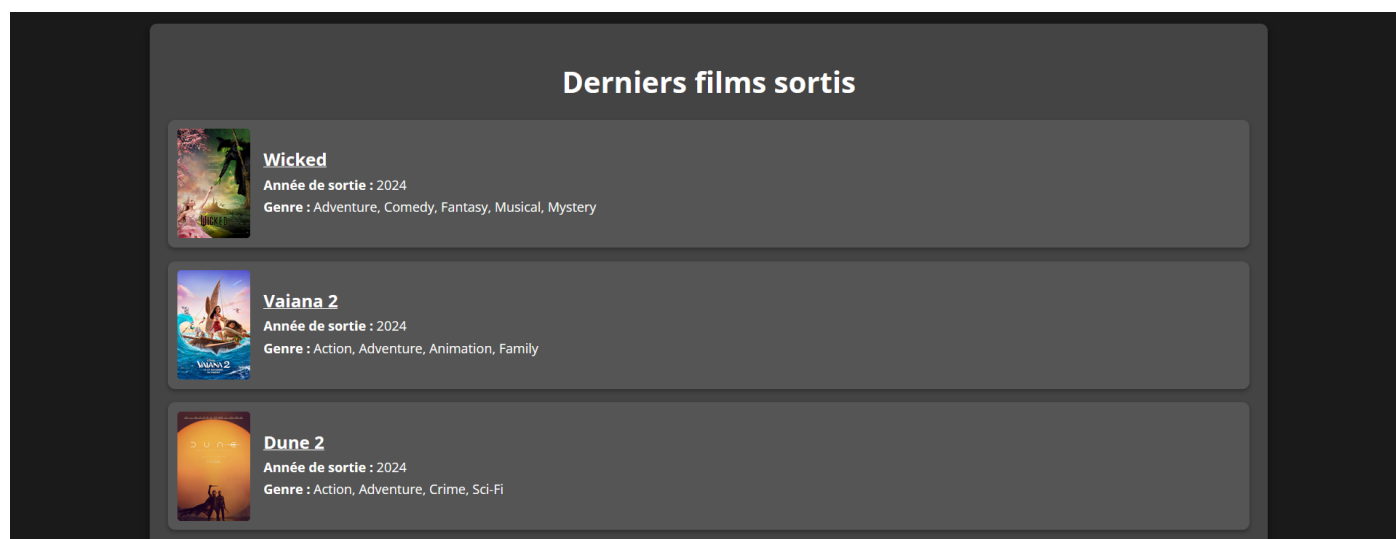
Liaison au frontend : Au niveau du front, deux fonctions sont nécessairement appelées. `getMovieIdByName` ainsi que `fetchTopTendancesData`. La première envoie le nom du film dans une requête GET envoyée à l'API `/api/trouver-id`, et retourne l'ID du film s'il est trouvé. La deuxième fonction, `fetchTopTendancesData`, utilise le fichier JSON disponible via la route `/top_tendances` pour récupérer les données du film tendance, comme le nom du film, l'URL de l'image, et l'URL du logo. Ces données sont entrées manuellement par l'admin. La fonction met ensuite à jour les éléments HTML (le lien, l'image, et le logo) du composant "top tendances".

Exemple de requête : <http://localhost:2030/api/trouver-id?nom=Home Alone>

Fichier backend : `/fonctionnalités_js/trouver_id_nomfilm.js`

Fichier liaison : `/liaisonBackFront/accueil.js`

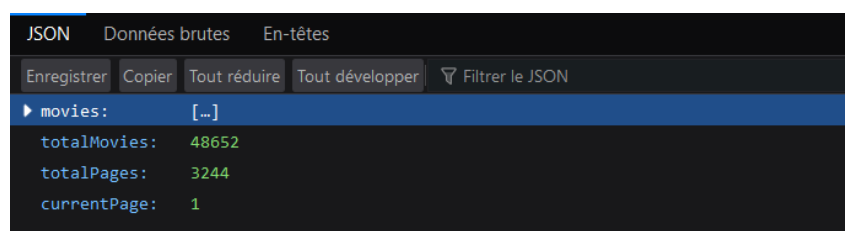
4.4 Fonctionnement de la page des derniers films sortis



Objectif : Afficher tous les films de la base de données du plus récent au plus ancien.

Fonctionnement backend : L'API `/api/derniers-films` fournit une liste paginée des derniers films triés par date de publication décroissante. Elle prend en compte un paramètre `page` pour gérer la pagination avec 15 films par page. La réponse inclut les films, le nombre total de films, le nombre total de pages, et la page actuelle.

Aperçu JSON brut :



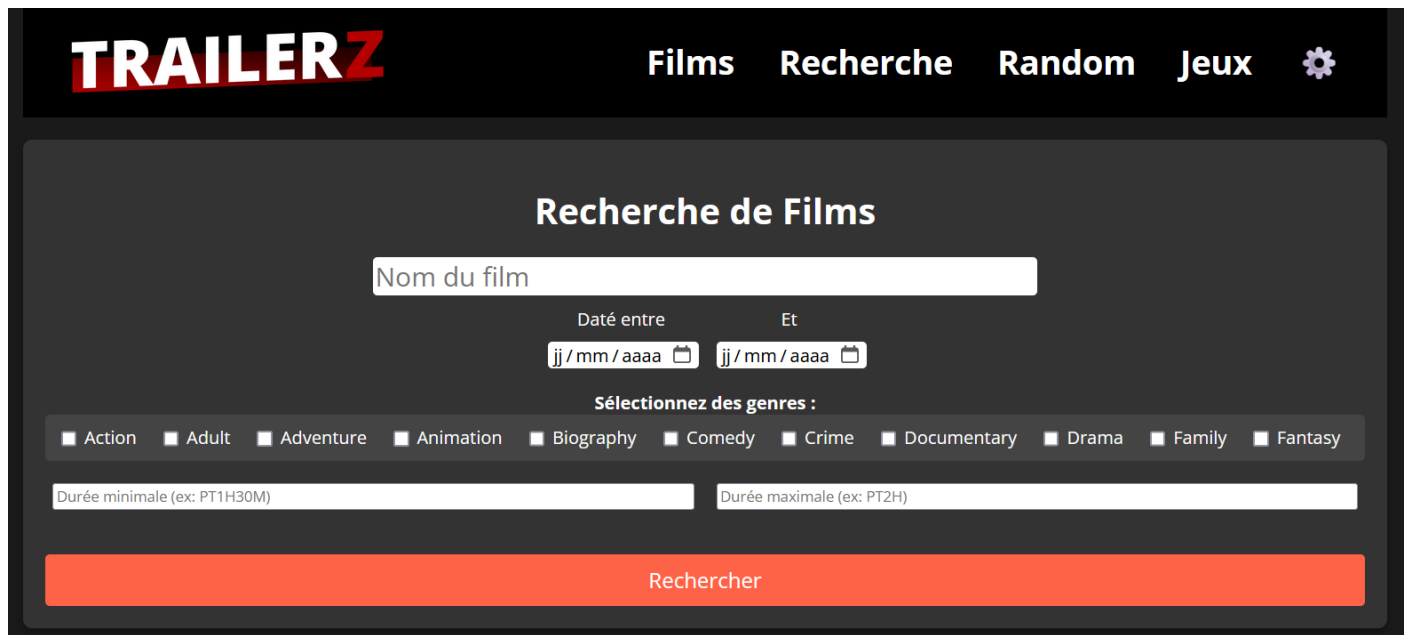
Liaison au frontend : Le code permet d'afficher les derniers films paginés. La fonction `fetchMovies` envoie une requête GET à l'API `/api/derniers-films` avec le numéro de page en paramètre, récupère les données JSON, et appelle `displayMovies` pour afficher les films. La fonction `displayMovies` insère dynamiquement le contenu des films dans le html, tandis qu'`updatePagination` met à jour les boutons de navigation et l'information sur les pages. Les événements sur les boutons "Précédent" et "Suivant" modifient la page courante et rappellent `fetchMovies` pour recharger les films. Enfin, le chargement initial s'effectue à la page 1 via `fetchMovies(currentPage)`.

Exemple de requête : <http://localhost:2030/api/derniers-films?page=1>

Fichier backend : `/fonctionnalités_js/filmsRecents.js`

Fichier liaison : `/liaisonBackFront/filmsRecents.js`

4.5 Fonctionnement de la page de Recherche



Objectif : Rechercher un film par son nom, sa date (curseur d'encadrement), son (ou ses) genre(s) et/ou sa durée.

Fonctionnement backend : L'API `/api/search-movie` permet de rechercher des films dans la base de données en fonction de multiples critères : nom, dates de publication, genres, acteurs, réalisateurs, et durée. Elle construit dynamiquement une requête en fonction des paramètres fournis dans l'URL. Les films correspondants sont récupérés et formatés en JSON, incluant des détails comme le titre, l'image, le genre, la note, le réalisateur, la durée, et la date.

Aperçu JSON brut :

```
▼ 0:
  name: "Inception"
  image: "https://m.media-amazon.com/images/M/MV5BMjAxMzY2NicxNF5BMl5BanBnXkFtZTcwNTI5OTM0Mw@@_V1_.jpg"
  genre:
    0: "Action"
    1: "Adventure"
    2: "Sci-Fi"
    3: "Thriller"
  director:
    name: "Réalisateur inconnu"
    url: "#"
  aggregateRating:
    ratingValue: "8.8"
    ratingCount: 2850656
  duration: "148 minutes"
  datePublished: "2010-07-14"
  id: "673765758ec1027b29eb216f"
```

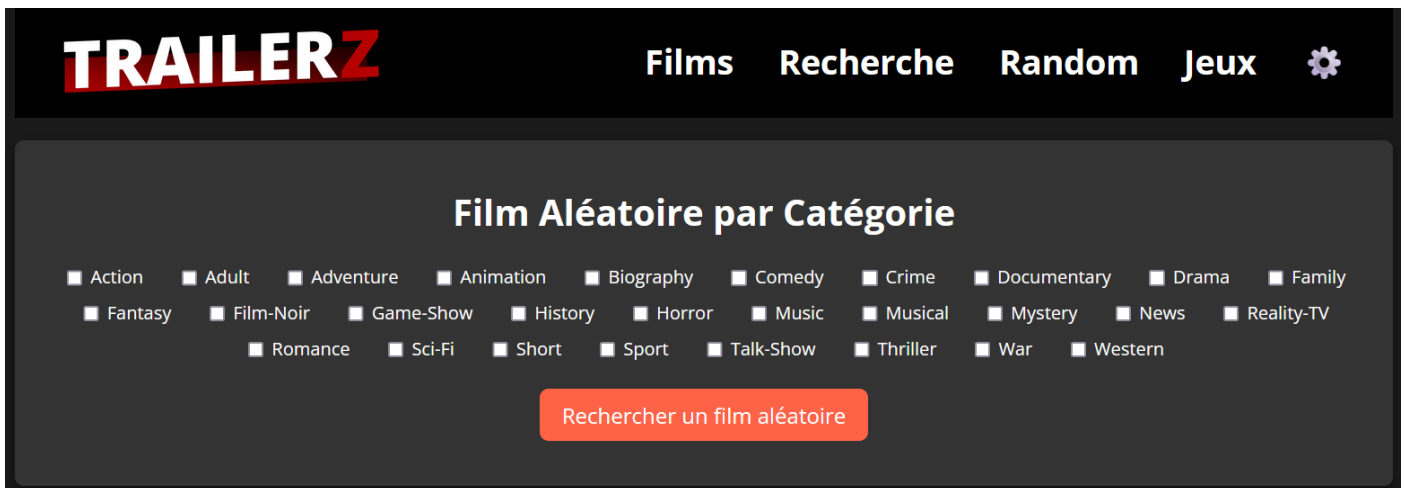
Liaison au frontend : Le code récupère les critères de recherche (nom, dates, genres, durée) d'un formulaire et construit une chaîne de requête pour l'API `/api/search-movie`. Il envoie cette requête à l'API pour obtenir les films correspondants. Ensuite, il injecte les résultats (images, titres, informations sur le film) dans le html pour les afficher à l'utilisateur.

Exemple de requête : [http://localhost:2030/api/search-movie?name=inception&startDate=2010-01-01&endDate=2010-12-31&genres=Action,Sci-Fi&actors=Leonardo DiCaprio&directors=Christopher Nolan&minDuration=PT1H30M&maxDuration=PT3H](http://localhost:2030/api/search-movie?name=inception&startDate=2010-01-01&endDate=2010-12-31&genres=Action,Sci-Fi&actors=Leonardo%20DiCaprio&directors=Christopher%20Nolan&minDuration=PT1H30M&maxDuration=PT3H)

Fichier backend : `/fonctionnalités_js/searchPage.js`

Fichier liaison : `/liaisonBackFront/recherche.js`

4.6 Fonctionnement de la page de génération automatique de film (Random)



Objectif : Récupérer au hasard un film de la BD en sélectionnant un ou plusieurs genres

Fonctionnement backend : L'API `/api/film-aleatoire` permet de récupérer un film aléatoire en fonction des catégories spécifiées dans la requête. Elle valide les catégories fournies et s'assure qu'elles sont valides parmi une liste prédéfinie. Ensuite, elle effectue une recherche dans la base de données pour trouver un film correspondant aux catégories sélectionnées, en choisissant un film aléatoirement. Si un film est trouvé, les informations pertinentes (nom, image, genre, réalisateur, note) sont renvoyées au format JSON.

Aperçu JSON brut :

```
{
  "id": "6737655d8ec1027b29eaa807",
  "name": "It Happened in Brooklyn",
  "image": "https://m.media-amazon.com/images/M/MV5BMGQ1ZjE3N2QtZmZLNz00OGVhLWFrMmMhOWE4MzYxYmMyMWE5XkEyXkFqcGdeQXVyNjc0MzMzNjA0._V1_.jpg",
  "genre": {
    "0": "Comedy",
    "1": "Musical",
    "2": "Romance"
  },
  "director": {
    "name": "Richard Whorf",
    "url": "https://www.imdb.com/name/nm0926636/"
  },
  "aggregateRating": {
    "ratingValue": "6.5",
    "ratingCount": 745
  }
}
```

Liaison au frontend : Le code permet de récupérer un film aléatoire en fonction des catégories sélectionnées par l'utilisateur. Il collecte les catégories choisies, les envoie à l'API `/api/film-aleatoire` via une requête GET. Si l'API renvoie un film, ses détails sont affichés dynamiquement sur la page (nom, image, genre, réalisateur, note).

Exemple de requête : <http://localhost:2030/api/film-aleatoire?categories=Action,Comedy,Drama>

Fichier backend : `/fonctionnalités_js/random_by_cat.js`

Fichier liaison : `/liaisonBackFront/randomByCat.js`

4.7 Fonctionnement de la page d'affichage du Film



Objectif : Afficher toutes les informations sur un film (Jaquette, bande annonce, synopsis, acteurs...)

Fonctionnement backend : Cette API sert à afficher les informations détaillées d'un film en fonction de son identifiant, récupéré depuis le modèle de données. Lorsque l'utilisateur accède à l'endpoint /film/:id, l'API extrait les données du film (titre, description, genre, acteurs, etc.) et remplit un modèle HTML grâce à un modèle de variables définies par {{variable}}. Les fonctionnalités incluent le formatage de la durée et de la date de sortie, la génération d'étoiles pour les notes, et l'ajout d'URL IMDb. Elle peut également rechercher un trailer sur YouTube et intégrer son URL dans la page. Si le film n'est pas trouvé ou une erreur survient, des messages appropriés sont affichés.

Nous avons décidé de ne pas réaliser de fichier de liaison backend-frontend sur cette page spécifiquement, car l'envoi des données directement dans le front grâce aux noms des variables était beaucoup plus rapide et optimisé.

Côté Frontend :

```
<div class="detail-item">
  <h3>Informations générales</h3>
  <p><strong>Lien IMDB :</strong> <a href="{{url}}" target="_blank">{{name}}</a></p>
  <p><strong>Synopsis :</strong> {{description}}</p>
  <p><strong>Durée :</strong> {{duration}}</p>
  <p><strong>Date de sortie :</strong> {{releaseDate}}</p>
  <p><strong>Genres :</strong> {{genres}}</p>
</div>
```

Avec un simple code {{variable}}, l'API détecte automatiquement les crochets et affiche l'information à l'intérieur.

Exemple de requête : <http://localhost:2030/film/673765688ec1027b29eadb65>

Fichier backend : /fonctionnalités_js/filmPage.js

4.8 Fonctionnement de l'ajout de films

Ajouter un Film

Nom du film :
Entrez le nom du film

Date de sortie :
jj/mm/aaaa

Genres :

☐ Action ☐ Adult ☐ Adventure ☐ Animation ☐ Biography ☐ Comedy ☐ Crime ☐ Documentary ☐ Drama ☐ Family ☐ Fantasy ☐ Film-Noir ☐ Game-Show ☐ History ☐ Horror ☐ Music ☐ Musical ☐ Mystery ☐ News ☐ Reality-TV ☐ Romance

Durée (en minutes) :
Durée en minutes

URL du film :
Lien vers le film

URL de l'Affiche du film :
Lien vers l'affiche du film

Mots clés (séparés par des virgules):
Mots,clés,virgules

Description :
Description du film

Acteurs :

Nom de l'acteur URL de l'acteur Supprimer

Ajouter un acteur

Directeurs :

Nom du directeur URL du directeur Supprimer

Ajouter un directeur

Createurs :

Nom du createur URL du createur Supprimer

Ajouter un createur

Ajouter

Objectif : Permettre à un administrateur d'ajouter un film à la base de données via un formulaire.

Fonctionnement backend : L'API `/api/add-movie` reçoit une requête POST contenant les données d'un film sous forme de JSON. Ces données incluent des informations comme le nom du film, la date de publication, les acteurs, réalisateurs, createurs, genres, durée, URL, affiche du film, mots-clés, etc. Le backend traite ces informations, valide les données, et les enregistre dans la base de données. Si tout est valide, une confirmation est renvoyée ; sinon, un message d'erreur est retourné.

Liaison au Frontend : Le script JavaScript côté frontend interagit avec l'API en envoyant les données du formulaire de film (comme le nom, la date de sortie, les acteurs, etc.) au backend. Lorsque l'utilisateur soumet le formulaire, les données sont collectées dans un objet JSON. Le script envoie ensuite cette requête à l'API via une méthode POST pour enregistrer le film dans la base de données.

Fichier Backend : `/fonctionnalites_js/admin_movie_api.js`

Fichier Liaison : `/liaisonBackFront/adminMovie.js`

4.9 Fonctionnement de la modification de film

Modifier un Film

ID du film :

Entrez l'ID du film

Nom du film :

Entrez le nouveau nom du film

Date de sortie :

jj/mm/aaaa

Genres :

Action

Adult

Adventure

Animation

Biography

Comedy

Crime

Documentary

Drama

Family

Fantasy

Film-Noir

Game-Show

History

Horror

Music

Musical

Mystery

News

Reality-TV

Romance

Durée (en minutes) :

Durée en minutes

URL de l'image :

Lien vers l'image

URL du film :

Lien vers le film

Mots clés (séparés par des virgules):

Mots, clés, virgules

Acteurs :

Nom de l'acteur

URL de l'acteur

Supprimer

Ajouter un acteur

Directeurs :

Nom du directeur

URL du directeur

Supprimer

Ajouter un directeur

Créateurs :

Nom du créateur

URL du créateur

Supprimer

Ajouter un créateur

Modifier

Objectif : Modifier les informations d'un film existant.

Fonctionnement backend : L'API `/api/update-movie` reçoit une requête PUT contenant un objet JSON avec les informations mises à jour du film. Cet objet peut inclure des champs comme le nom, la date de publication, les acteurs, etc. Le backend met à jour les informations du film dans la base de données et renvoie une confirmation de la mise à jour ou un message d'erreur si la modification échoue.

Liaison au frontend : Le script JavaScript du formulaire de mise à jour (`update-movie-form`) recueille les données modifiées du film, puis envoie une requête HTTP PUT à l'API `/api/update-movie` avec les informations mises à jour sous forme de JSON. Si la requête réussit, un message de succès est affiché et le formulaire est réinitialisé.

Fichier Backend : `/fonctionnalites_js/admin_movie_api.js`

Fichier Liaison : `/liaisonBackFront/adminMovie.js`

4.10 Fonctionnement de la suppression de film

The screenshot shows a dark-themed web form titled "Supprimer un Film". On the left, there is a label "ID du film :" followed by a text input field containing the hexadecimal string "6737599db6aa19b8b18cf93f". To the right of the input field is a red button with the white text "Supprimer".

Supprimer un Film

ID du film :

6737599db6aa19b8b18cf93f

Supprimer

Objectif : Supprimer un film de la base de données.

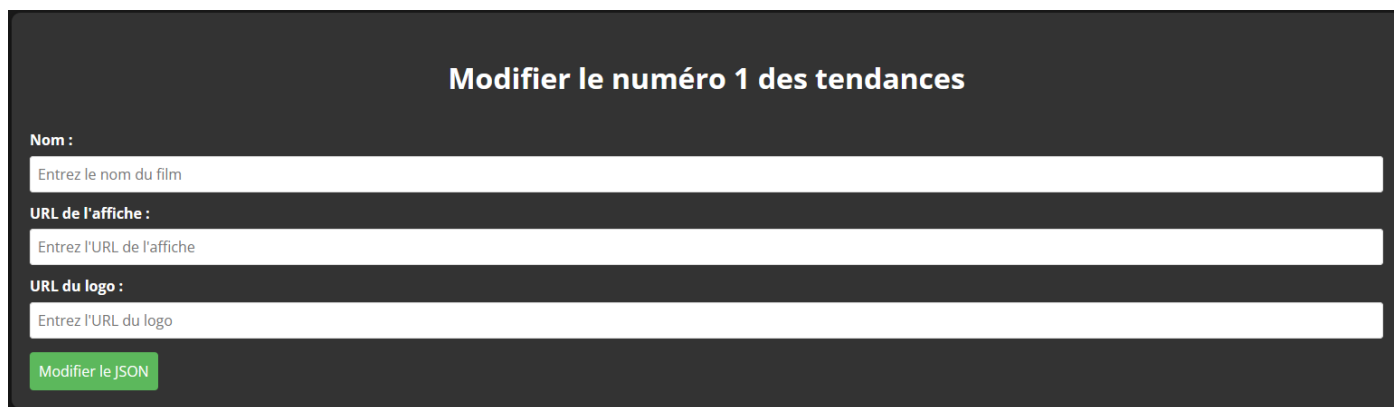
Fonctionnement backend : L'API /api/delete-movie reçoit une requête DELETE contenant l'ID du film à supprimer. Le backend supprime le film de la base de données et renvoie une confirmation de la suppression ou un message d'erreur en cas de problème.

Liaison au frontend : Le script JavaScript du formulaire de mise à jour (update-movie-form) recueille les données modifiées du film, puis envoie une requête HTTP PUT à l'API /api/update-movie avec les informations mises à jour sous forme de JSON. Si la requête réussit, un message de succès est affiché et le formulaire est réinitialisé.

Fichier Backend : /fonctionnalites_js/admin_movie_api.js

Fichier Liaison : /liaisonBackFront/adminMovie.js

4.11 Fonctionnement de la gestion du film en tendance

The screenshot shows a dark-themed web form titled "Modifier le numéro 1 des tendances". It contains three input fields: "Nom :" with placeholder text "Entrez le nom du film", "URL de l'affiche :" with placeholder text "Entrez l'URL de l'affiche", and "URL du logo :" with placeholder text "Entrez l'URL du logo". Below these fields is a green button labeled "Modifier le JSON".

Modifier le numéro 1 des tendances

Nom :
Entrez le nom du film

URL de l'affiche :
Entrez l'URL de l'affiche

URL du logo :
Entrez l'URL du logo

Modifier le JSON

Objectif : Permettre à l'administrateur de modifier et de mettre à jour les informations relatives à un film dans un fichier JSON. L'interface frontend permet à l'administrateur de saisir le nom du film, l'URL de l'affiche du film et l'URL du logo, puis d'envoyer ces données au backend pour mettre à jour le fichier JSON.

Fonctionnement backend : L'API expose une route POST `/api/update-json` qui permet de mettre à jour le fichier JSON contenant les informations sur les films. Lorsque l'administrateur soumet un formulaire avec les nouvelles informations (nom du film, URL de l'affiche et du logo), ces données sont envoyées au backend sous forme de requête JSON. Le backend utilise la méthode `fs.writeFile` de Node.js pour enregistrer les données dans le fichier JSON. Si l'écriture est réussie, l'API renvoie un message de succès ; sinon, un message d'erreur est renvoyé.

Liaison au frontend : Le frontend contient un formulaire permettant à l'administrateur de saisir les informations du film. Lorsque le formulaire est soumis, une requête POST est envoyée au backend via l'API `/api/update-json` pour mettre à jour le fichier JSON. La réponse du backend est utilisée pour afficher un message de succès ou d'erreur à l'utilisateur. Lors du clic sur le bouton "Mettre à jour", le frontend récupère les données du formulaire, les valide (s'assurer que tous les champs sont remplis), puis envoie les données via `fetch` à l'API backend pour qu'elles soient enregistrées dans le fichier JSON.

Fichier Backend : `/fonctionnalites_js/admin_movie_api.js`

Fichier Liaison : `/liaisonBackFront/modifMovie.js`

4.12 Fonctionnement de la récupération d'un film, pour le jeu « Devine le film »

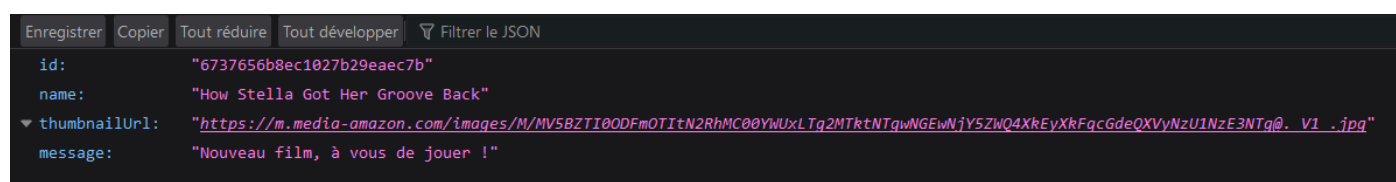


Objectif : Récupérer un film aléatoire à partir de la base de données, avec son ID et son image associée, pour l'afficher dans le jeu.

Fonctionnement backend : `/api/random-movie` : Cette API sélectionne un film aléatoire dans la base de données. Elle s'assure que le film sélectionné a une image valide associée à l'URL `thumbnailUrl`. Si l'image est invalide ou manquante, un message d'erreur est renvoyé. Sinon, le film (avec son ID, son nom et son image) est renvoyé au frontend pour affichage.

La récupération aléatoire se fait en utilisant l'agrégation MongoDB avec une étape `$sample`, qui permet de choisir un document au hasard.

Aperçu JSON brut :



Liaison au frontend : Le frontend appelle l'API `/api/random-movie` pour récupérer un film aléatoire au chargement de la page (ou après qu'une réponse correcte soit donnée). L'image du film est affichée et le jeu commence. Le nom du film est stocké dans une variable pour être comparé avec la réponse de l'utilisateur. Si la réponse est correcte, un nouveau film est chargé en appelant à nouveau l'API.

Exemple de requête : <http://localhost:2030/api/random-movie>

Fichier Backend : `/fonctionnalites_js/devine_api.js`

Fichier Liaison : `/liaisonBackFront/devinelefilm.js`

4.13 Fonctionnement de la recherche et vérification d'un film, pour le jeu « Devine le film »



Objectif : Permettre à l'utilisateur de rechercher des films en fonction de la saisie partielle du nom, et vérifier si la réponse fournie par l'utilisateur correspond à un film proposé.

Fonctionnement backend : /api/search-movies : L'API recherche des films dont le nom correspond partiellement à la requête de l'utilisateur. Elle utilise une recherche insensible à la casse dans la base de données et retourne un maximum de 5 films correspondants. Chaque film retourné contient le nom du film et l'URL de son image. Si aucun film ne correspond à la requête, une liste vide est renvoyée.

Aperçu JSON brut :

```
▼ movies:
  ▼ 0:
    ▼ trailer:
      ▼ thumbnailUrl: "https://m.media-amazon.com/images/M/MV5BMTQ1ZmIzOTAtNDcwZi00NDVhLWE4NWItYWNhZGY1MmVLZGU0XkEyXkFqcGdeQWVb2xpbnhk. V1 .jpg"
      _id: "673765758ec1027b29eb216f"
      name: "Inception"
```

Liaison au frontend : Lors de la saisie dans le champ de recherche (élément #movieInput), le frontend envoie une requête à l'API /api/search-movies pour obtenir des suggestions de films. Les résultats sont affichés sous forme de liste dans l'élément #suggestions. L'utilisateur peut cliquer sur une suggestion pour remplir le champ de saisie.

Lorsque l'utilisateur appuie sur "Entrée", le frontend envoie la réponse de l'utilisateur pour vérifier si le film correspond au film actuel (récupéré via /api/random-movie). Si la réponse est correcte, un nouveau film est chargé après un court délai. Sinon, un message d'erreur est affiché.

Exemple de requête : <http://localhost:2030/api/search-movies?query=inception>

Fichier Backend : /fonctionnalites_js/devine_api.js

Fichier Liaison : /liaisonBackFront/devinelefilm.js



NICE CÔTE D'AZUR