Compte Rendu Technique - Livrable 4

DRAVET Timothée - FISA INFO 4A

Phase 4 : Projet Final

Table des matières

1 Introduction

Ce rapport présente l'aboutissement du projet CinéExplorer, une application web développée avec Django intégrant une architecture hybride SQLite et MongoDB. L'objectif principal est de démontrer la capacité à gérer une stratégie multi-bases pour optimiser les performances de lecture et de recherche. Le code source complet et l'historique du projet sont disponibles à l'adresse suivante : github.com/timotheedvt/cinexplorer.

2 Architecture Globale du Système

L'architecture repose sur trois piliers technologiques :

— **Django (Framework Web)** : Assure la logique métier et le routage.
— **SQLite** : Stockage relationnel normalisé pour la recherche et les listes.
— **MongoDB Replica Set** : Stockage orienté documents structurés pour les détails riches et les statistiques.

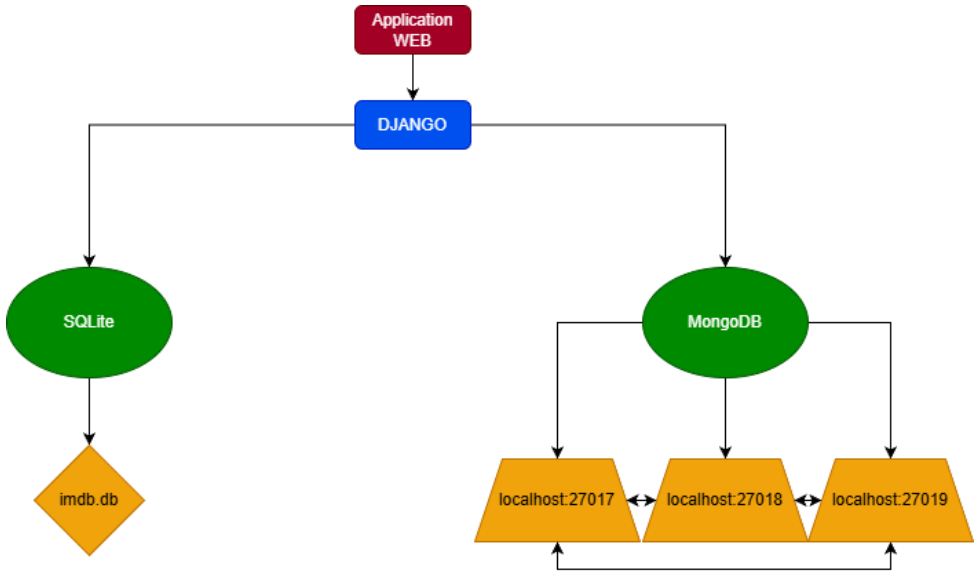


FIGURE 1 – Architecture hybride du système CinéExplorer

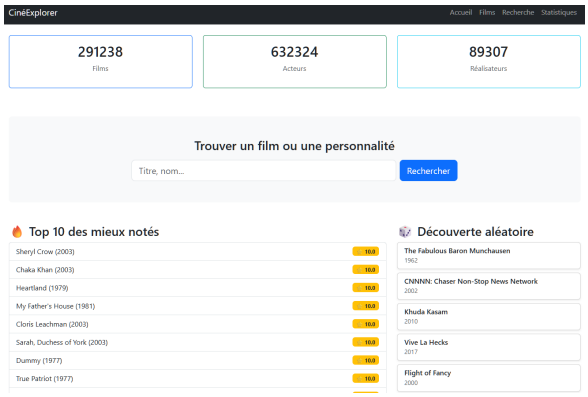
3 Choix Technologiques Justifiés

— **Stratégie Hybride** : SQLite est utilisé pour sa rapidité sur les requêtes de filtrage simples (genre, année). MongoDB est choisi pour le détail des films afin d'éviter les jointures SQL coûteuses (N jointures pour 1 document).
— **Replica Set (3 nœuds)** : Garantit la haute disponibilité des données structurées et la tolérance aux pannes.
— **Chargement Asynchrone** : Utilisation de Chart.js et d'appels API AJAX pour éviter le gel de l'interface lors du calcul de statistiques lourdes.

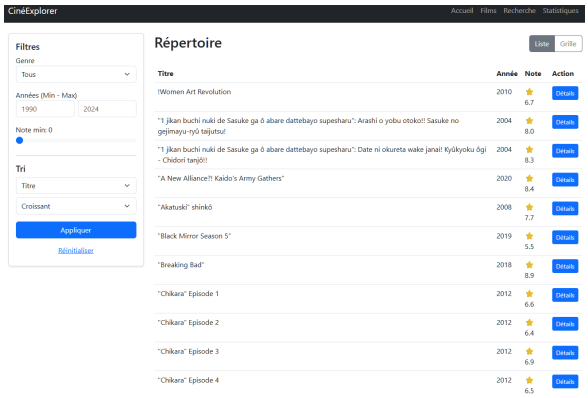
4 Description des Fonctionnalités

4.1 Interface Utilisateur

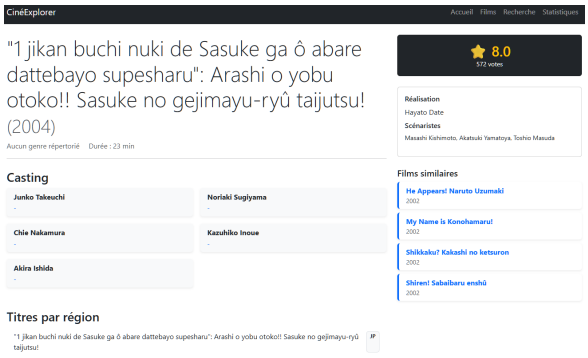
L'interface a été conçue avec **Bootstrap 5** pour garantir un rendu responsive.



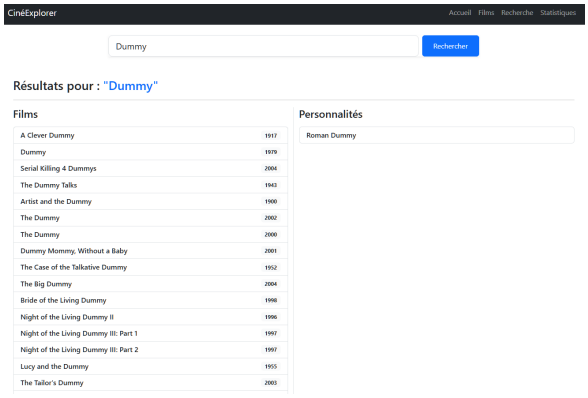
(a) Page d'accueil



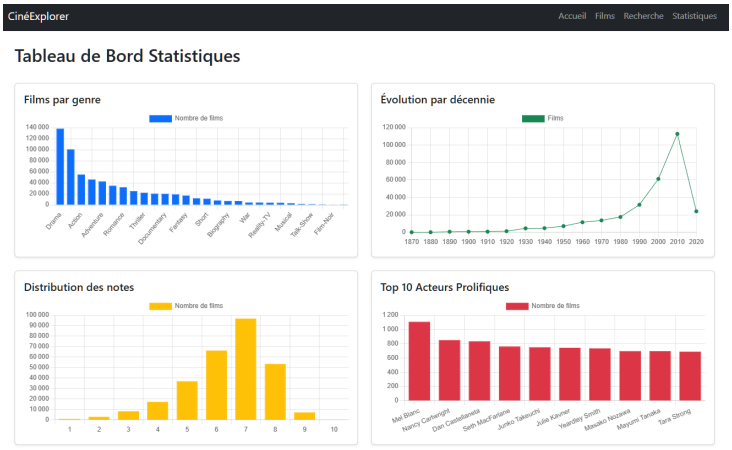
(b) Répertoire des films



(c) Détails d'un film (MongoDB)



(d) Moteur de recherche (SQLite)



(e) Statistiques (Chart.js)

FIGURE 2 – Aperçu des interfaces de l'application Django CinéExplorer

4.2 Recherche Groupée

La recherche interroge simultanément les films et les personnes via SQLite en utilisant l'opérateur `LIKE`.

5 Stratégie Multi-Bases

Le projet implémente des services distincts pour isoler la logique d'accès aux données.

```python
def get_movie_detail(movie_id):
    db = get_mongo_client()
    return db.MOVIE_COMPLETE.find_one({"_id": movie_id})

def get_similar_movies(movie, limit=4):
    db = get_mongo_client()
    query = {
        "_id": {"$ne": movie["_id"]},
        "$or": [
            {"genres": {"$in": movie.get("genres", [])}},
            {"directors.name": {"$in": [d["name"] for d in movie.get("directors", [])]}}
        ]
    }
    return list(db.MOVIE_COMPLETE.find(query).limit(limit))
```

Listing 1 – Extrait du service MongoDB (Récupération structurée)

6 Benchmarks de Performance

L'utilisation de MongoDB pour les détails complexes permet une réduction drastique de la latence par rapport à SQLite (jointures multiples).

Opération	**SQLite (ms)**	**MongoDB (ms)**
Détail complet (Casting + Genres + Titres)	$\sim$ 150ms	$\sim$ 40ms
Recherche textuelle simple	$\sim$ 15ms	N/A
Agrégation statistique (Genres)	$\sim$ 800ms	$\sim$ 200ms

TABLE 1 – Comparaison des temps de réponse moyens

7 Difficultés et Solutions

7.1 Problème : Lenteur des statistiques

Description : Le calcul des statistiques bloquait le rendu de la page pendant plusieurs secondes. **Solution** : Implémentation d'un point d'accès API asynchrone et d'un écran squelette (Skeleton Screen) pour une meilleure UX.

```
def stats_api(request):
    sqlite_data = sqlite_service.get_stats_data()
    ratings_data = mongo_service.get_rating_distribution()
    return JsonResponse({
        'genres': sqlite_data['genres'],
        'decades': sqlite_data['decades'],
        'ratings': ratings_data
    })
```

Listing 2 – Point d'accès API pour le chargement asynchrone

7.2 Problème : Accès aux IDs MongoDB dans Django

Description : Django interdit l'accès aux variables commençant par un souligné (`_id`). **Solution** : Création d'un template filter personnalisé `get_id`.

8 Conclusion

Le projet CinéExplorer démontre la complémentarité des mondes SQL et NoSQL. Si SQLite reste imbattable pour la recherche relationnelle, MongoDB offre une flexibilité et une performance de lecture supérieure pour les objets complexes.