Web Frameworks

(CSL253)

Project Report

Faculty name: Mr. Sumit Kumar

Student name: Krishanu Anand (23csu166)

Ronit (23csu171)

Krish Gaur (23csu162)

Semester: 5th

Group: FS-1A

Department of Computer Science and Engineering

The NorthCap University, Gurugram- 122001, India

Session 2023-24

Table of Contents

1. Project Description

The Movie Streaming Platform is a complete end-to-end video streaming solution designed to mimic the functionality and user experience of top streaming services. Initially created as a frontend project, it evolved into a full-scale system during a hackathon, integrating authentication, database management, machine learning recommendations, administrative controls, and adaptive HLS video streaming.

Initial Phase: Frontend-Only React Application

The first version focused solely on UI development:

- Highly responsive design using TailwindCSS
- Netflix-style movie grids
- Smooth navigation using React Router
- Interactive elements such as hover-based movie previews
- Client-side search & filtering
- Placeholder/mock movie data for UI showcase

Hackathon Expansion: Full-Stack Streaming Platform

The project grew into a fully functional application with:

- Secure logins via Google OAuth
- User-specific data such as favorites and watch history
- Real-time watch party using WebSockets
- An ML-based recommendation engine via FastAPI
- Admin panel to upload & manage content
- Adaptive HLS streaming using FFmpeg
- Scalable backend in Spring Boot + MongoDB

Key Outcomes

- Transition from a simple UI to a complete streaming ecosystem
- End-to-end integration across frontend, backend, ML, and media processing

- Production-style features similar to Netflix, Hotstar, and Prime Video

2. Problem Statement

The project had two major phases, each with unique constraints and goals.

Frontend-Only Objective

Create a modern, professional-grade streaming platform UI demonstrating:

- Clean, responsive layouts
- Interactive browsing
- Client-side search
- UX similar to industry standards

However, this version lacked data persistence, authentication, and actual streaming functionality.

Hackathon Objective

Transform the static UI into a complete streaming system with:

- Real authentication using Firebase
- Movie management through an admin dashboard
- MongoDB-backed storage
- Adaptive video streaming using HLS
- Machine learning-powered recommendations
- Real-time watch party synchronization
- Scalable microservice-style architecture

Main Problems to Solve

1. Implement a secure login system
2. Allow admins to upload & process large video files
3. Encode movies into HLS format for adaptive streaming
4. Store and manage user activity at scale

5. Make the UI communicate with the backend efficiently
6. Provide recommendations based on history and genres
7. Support real-time interactions during watch parties

3. Analysis

3.1 Hardware Requirements

Minimum (for development/testing)

- **CPU:** Dual-core 2.0 GHz
- **RAM:** 4 GB
- **Storage:** 10 GB free
- **Network:** 5 Mbps for normal streaming
- **Display:** 1024×768

Recommended

- **CPU:** Quad-core 3.0 GHz+
- **RAM:** 8 GB
- **Storage:** 20 GB SSD
- **Network:** 25 Mbps for HD/1080p testing
- **Display:** 1920×1080

Server Requirements (For Hosting)

- **CPU:** 8 cores or more
- **RAM:** 16–32 GB
- **Storage:** 500 GB SSD (for multiple movie files)
- **Network:** 100 Mbps minimum
- **GPU:** Optional for faster video transcoding

3.2 Software Requirements

Development Environment

- OS: Windows / macOS / Linux
- Node.js (Frontend build)
- Java 17+ (Spring Boot backend)
- MongoDB
- Maven
- Python 3.10+ (ML service)

Runtime Requirements

- Browser (Chrome/Firefox/Edge/Safari)
- JRE for backend
- MongoDB server
- FFmpeg CLI installed

Tools & Libraries

- VS Code / IntelliJ IDEA
- Git + GitHub
- Jest & JUnit
- Postman for API testing
- WebSocket tools
- FastAPI for ML microservice

4. Design

4.1 Data / Input–Output Description

Inputs

- Login tokens (Google Firebase OAuth)
- Movie uploads (video files, poster images)
- Metadata (title, description, genres)

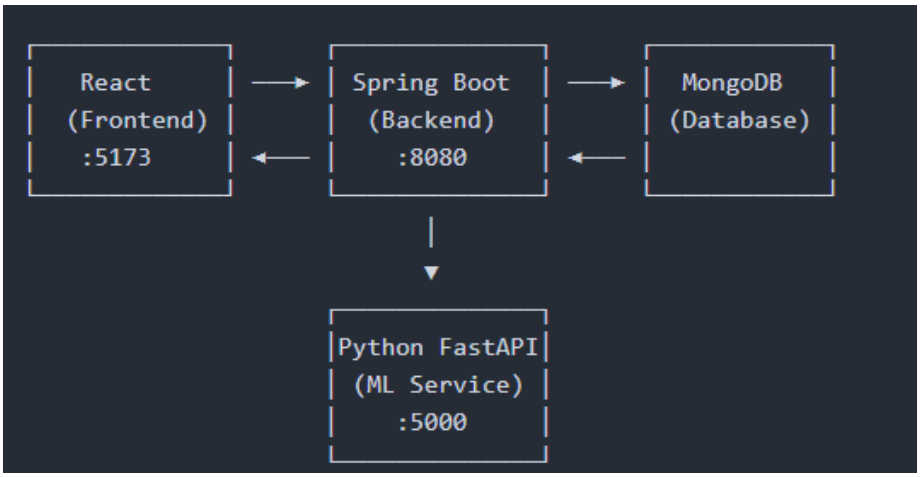
- User actions (search text, watch history, favorites)

Outputs

- Paginated movie lists
- Filtered/sorted search results
- HLS video streams (.m3u8 playlists, .ts segments)
- User dashboards (favorites, history, recommendations)
- Admin-level analytics

4.2 Architecture & Schemas (Expanded)

System Architecture




```
Frontend (React, Vite)
      ⇅ Axios HTTP
Backend (Spring Boot)
      ⇅
MongoDB (NoSQL)
      ⇅
FFmpeg (Video Processing)
      ⇅
Python FastAPI (ML Engine)
      ⇅
WebSocket Server (Real-Time Watch Party)
```

High-Level Flow

Users → Login → Search/Watch Movies → Add to Favorites/Watchlist → Receive Recommendations

Admins → Upload Movie → FFmpeg Converts to HLS → Data Stored in MongoDB → Available for Users

---

ER Diagram (Described)

Entities:

1. **User**
    - id, name, email, picture
    - favorites[], watchHistory[], role
2. **Movie**
    - id, title, description, genres[]
    - rating, release date
    - videoDetails (HLS URLs, duration, size)
    - statistics (views, likes)
3. **Watch Party**
    - roomId
    - movieId
    - participants[]
    - playback state

Relationships:

- User *favorites* many Movies
- User *watches* many Movies
- Movie *has* statistics
- WatchParty *is linked to* a Movie

Users Collection Schema (Detailed)

Includes:

- OAuth info from Firebase
- Activity tracking
- Preferences for ML
- Last active timestamps
- Lists of movie IDs for favorites and watch later

Movies Collection Schema (Detailed)

Includes:

- Metadata (title, genres, rating)
- Posters and thumbnails
- Full video processing details
- Generated HLS manifest URLs
- Social engagement (likes, views)
- Timestamps for auditing

Authentication Flow (Deep Explanation)

1. User logs in using Google
2. Firebase generates a secure ID token
3. Frontend attaches token to every API request
4. Backend middleware validates token with Firebase Admin SDK
5. If valid → user record is created/updated in MongoDB
6. Backend returns authenticated user profile
7. Frontend stores user details in Zustand state

8. User session remains active until logout or expiry

Movie Upload & HLS Processing Flow (Detailed)

1. Admin selects a movie file (MP4/MKV)
2. Frontend validates size (<2GB recommended)
3. Upload via REST API (multipart form)
4. Backend stores raw file in disk/storage
5. FFmpeg is triggered:
    - Extracts duration
    - Generates quality variants (480p/720p/1080p)
    - Creates .m3u8 master playlist
    - Segments video into multiple .ts chunks
6. Thumbnails are extracted
7. Final metadata saved in MongoDB
8. Admin dashboard shows upload status

5. Implementation & Testing

Stage-by-Stage Breakdown

Stage 1 — Initial Setup

- React project initialized with Vite
- Installed Tailwind CSS, Zustand
- Spring Boot backend setup with JWT filter template
- MongoDB connection configured

Stage 2 — Authentication

- Implemented Google OAuth login on frontend
- Setup Firebase Admin SDK on backend to verify ID tokens

- Login API stores user info in MongoDB
- Added role-based access for Admin

Testing:
✓ Token decoding
✓ Unauthorized access blocked
✓ Navigation logic based on login state

Stage 3 — Movie Management

- CRUD operations for movies
- Multipart upload endpoint
- FFmpeg integration for HLS conversion
- Metadata extraction (duration, size)
- Save processed HLS paths to DB

Testing:
✓ Upload of small and large video files
✓ Correct generation of HLS playlists
✓ Error handling for corrupted files

Stage 4 — UI Development

- Movie grid with hover previews
- Dynamic search bar (backend-powered)
- Genre filters
- Pagination
- Watch later, favorites

Testing:
✓ Mobile responsiveness
✓ Smooth transitions
✓ 60 FPS interactions

Stage 5 — Admin Panel

- Upload movie form
- Edit/Delete movies
- View processing status
- Real-time progress indicators

Testing:
✓ Role-checking
✓ Database updates
✓ Display of processing logs

Stage 6 — Streaming

- HLS player integrated using <video> tag + hls.js
- Multi-quality adaptive streaming
- Watch progress tracking
- Likes + views updates

Testing:
✓ Seek, pause, resume
✓ Smooth switching between 480p/720p/1080p
✓ No buffering for stable networks

Stage 7 — Watch Party

- WebSocket room creation
- Sync play/pause/seek
- Multi-user events
- Timeline broadcast

Testing:

✓ Multiple clients sync within milliseconds

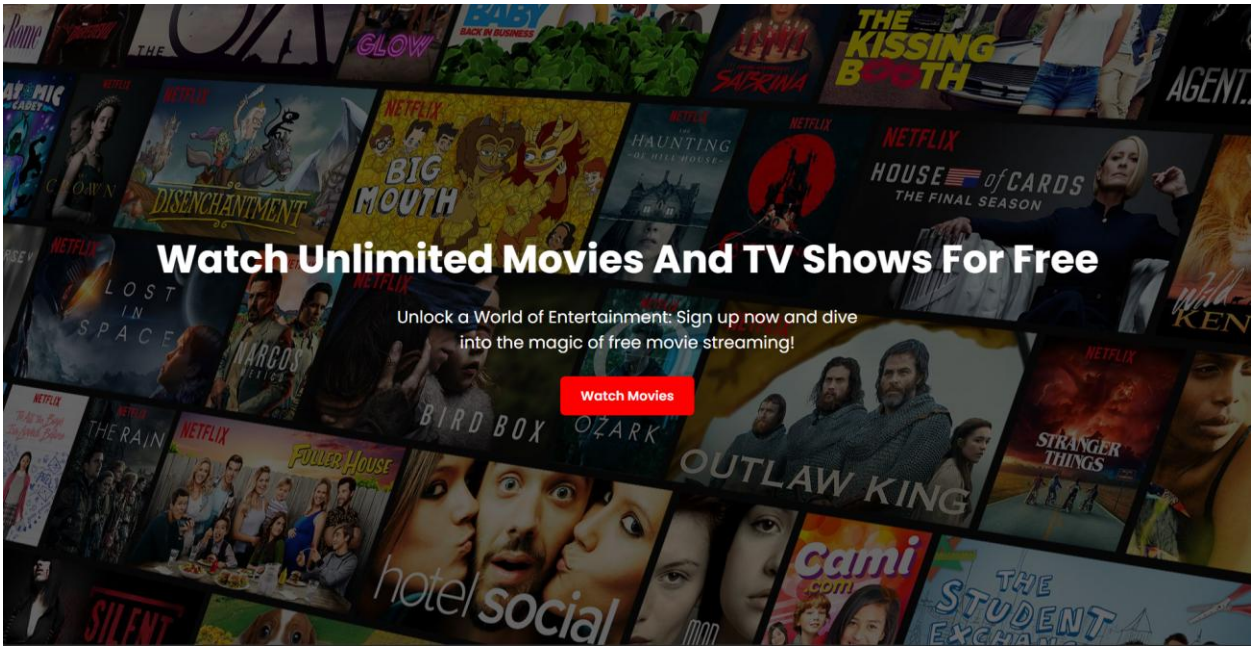
✓ Host controls

✓ Disconnection handling

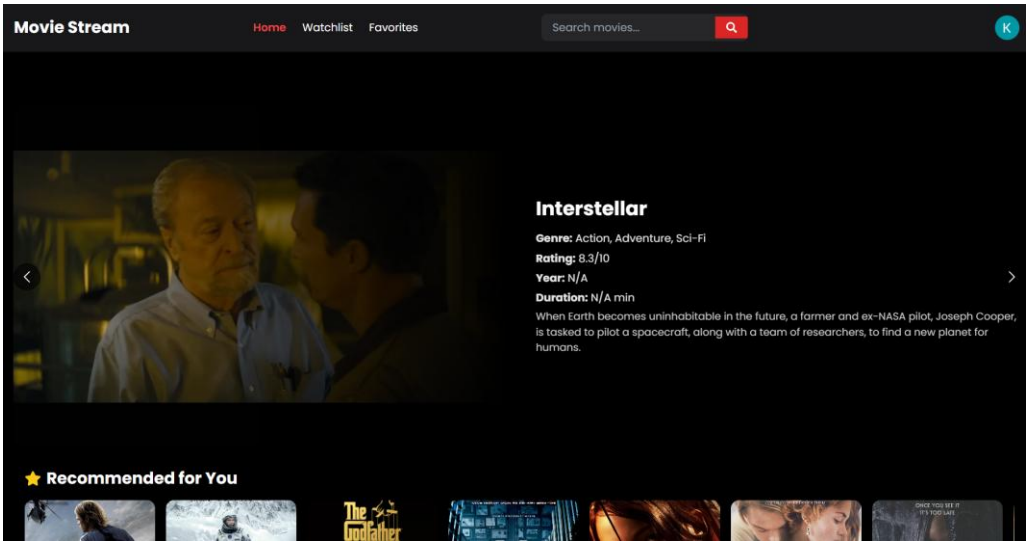
6. Output (Screenshots Explanation)

(Descriptions for actual screenshots you will attach)

1. **Landing Page**

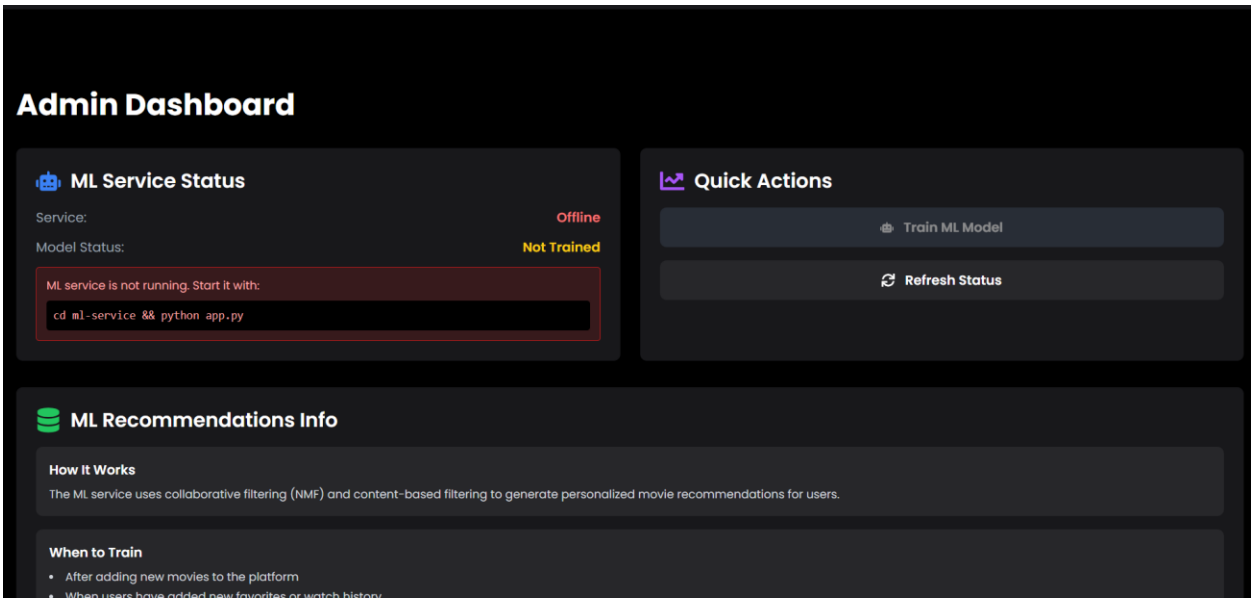


2. **Movie Grid**

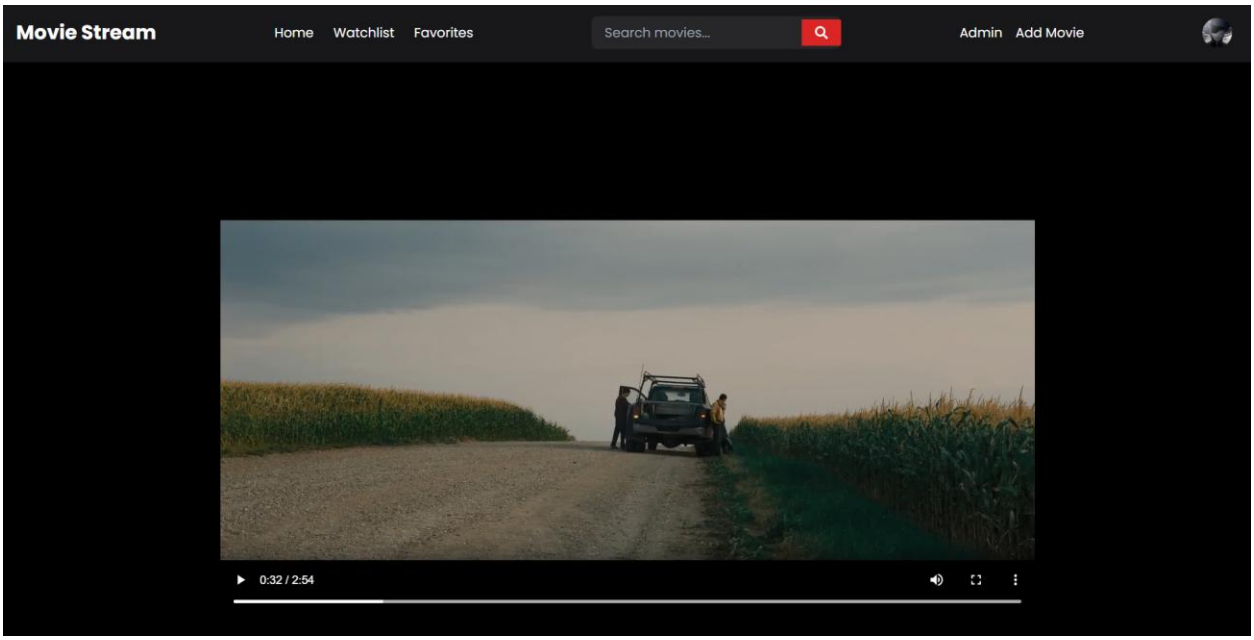


3. **Movie Details Page**

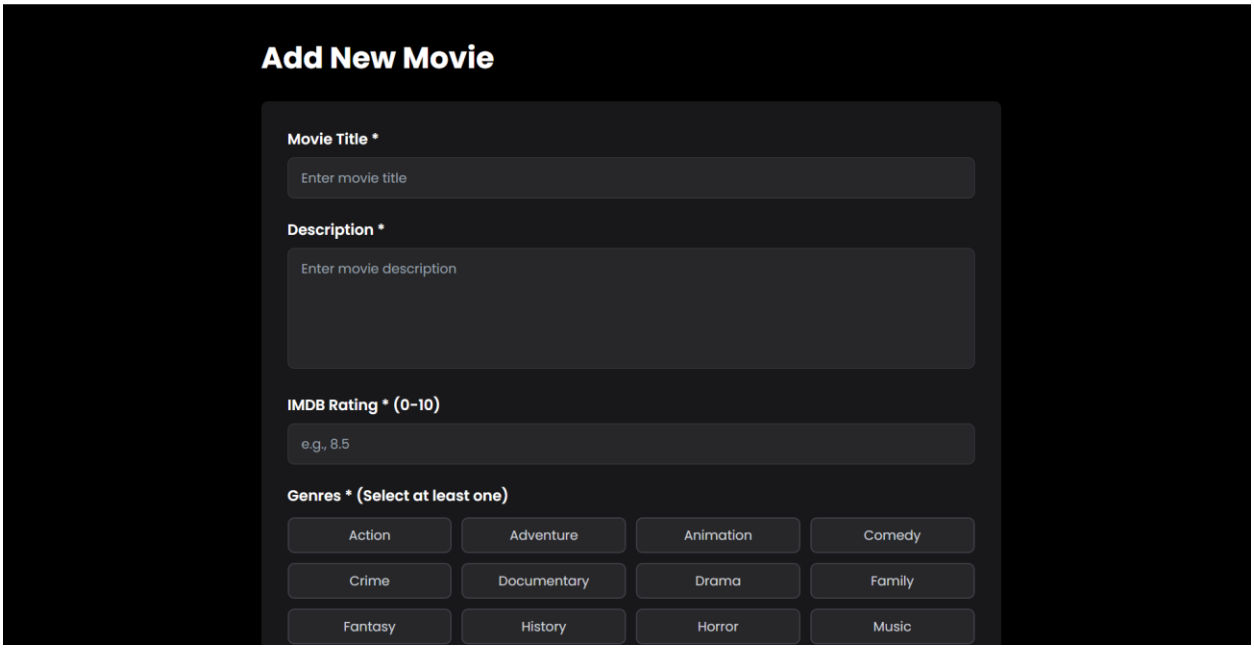
4. **Admin Dashboard**

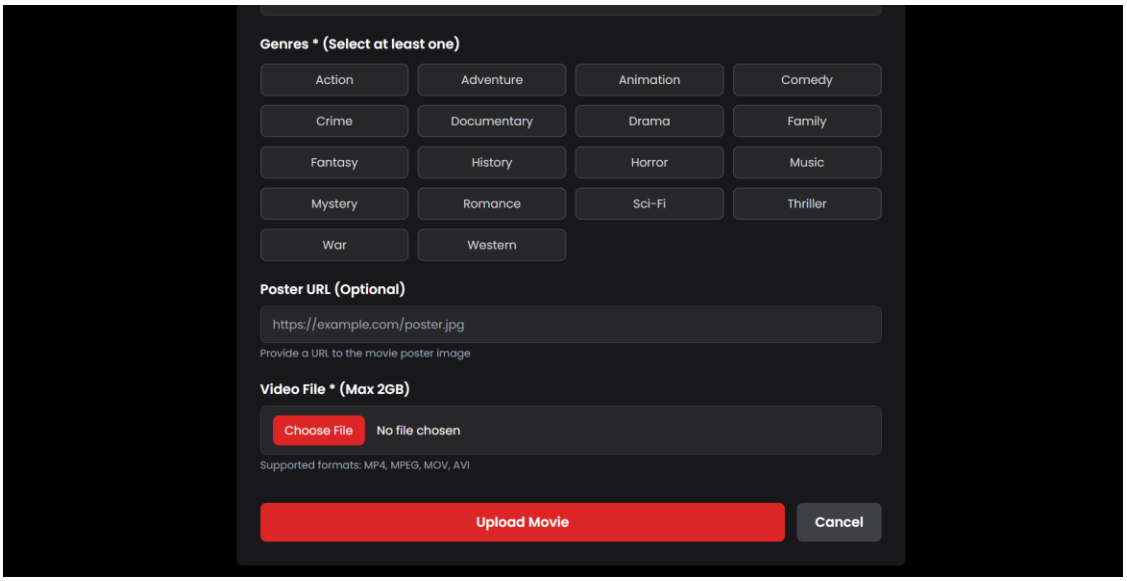


5. **HLS Player**

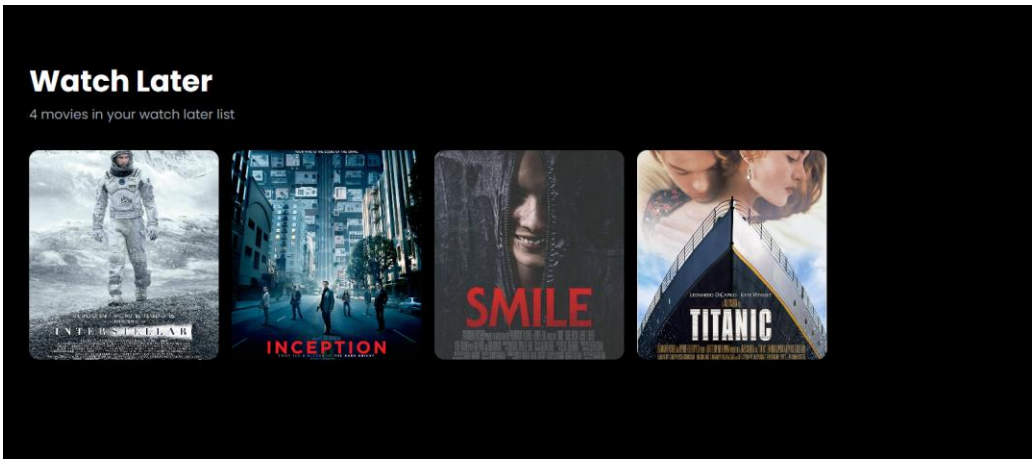

Movie Stream
Home
Watchlist
Favorites
Search movies...
Admin
Add Movie
0:32 / 2:54

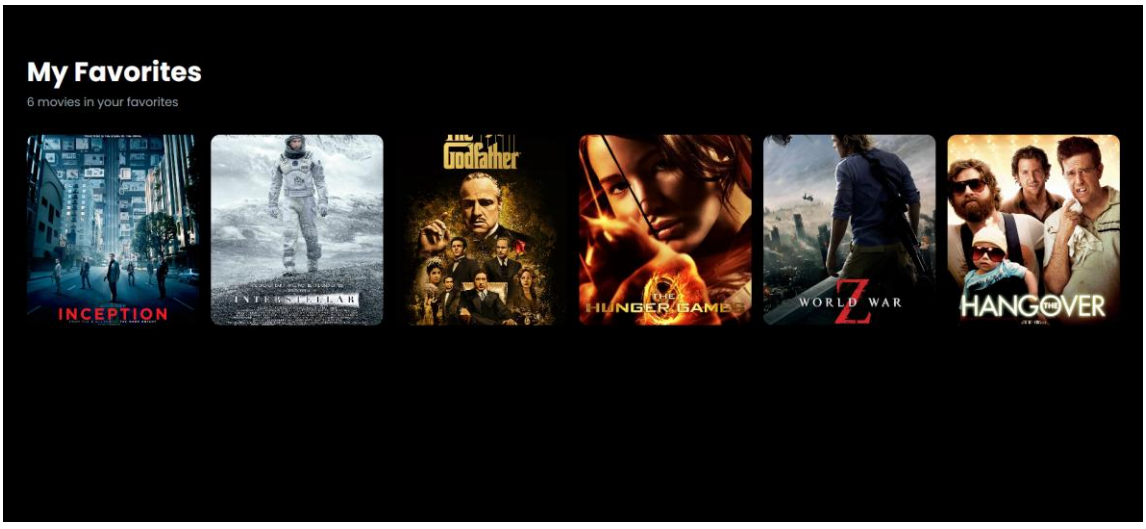
6. **Add Movie**


Add New Movie
Movie Title *
Enter movie title
Description *
Enter movie description
IMDB Rating * (0-10)
e.g., 8.5
Genres * (Select at least one)
Action
Adventure
Animation
Comedy
Crime
Documentary
Drama
Family
Fantasy
History
Horror
Music



7. **Watch Party and Favourites**





7. Conclusion & Future Scope

Conclusion

The project successfully evolved from a simple frontend demo to a comprehensive streaming ecosystem, achieving:

- Full-stack integration
- Secure OAuth login
- Efficient movie upload/processing pipeline
- Adaptive HLS streaming powered by FFmpeg
- Real-time watch party
- Personalized recommendations
- Clean and responsive UI
- Admin-level content management

The system demonstrates real-world production features and provides an excellent foundation for scalable media platforms.

Future Scope (Detailed)

Short-Term Enhancements

- Chat and reactions in watch party
- PWA support for offline caching
- Smarter ML models for personalization
- Social sharing, reviews, ratings
- Cross-platform mobile apps

Mid-Term Goals

- Live streaming modules
- Offline downloads (DRM required)
- Globalization (multi-language UI)

- Deep analytics dashboards
- CDN-based HLS distribution

Long-Term Vision

- AI-based moderation of content uploads
- Personalized feed ranking using deep learning
- Multi-tenant streaming service architecture
- 4K / HDR / 360° immersive streaming
- AR/VR watch party experiences

Performance Metrics

- Low-latency HLS loading
- Real-time sync under 200 ms
- Scalable to thousands of users
- High search performance with indexed queries
- Optimized FFmpeg workflows

Business Impact

The architecture can be adapted for:

- OTT platforms
- E-learning video systems
- Corporate training portals
- Event broadcasting
- Community-based streaming services

It is modular, maintainable, and production-ready with real-world technologies.