Project Report

On

Movie Streaming Application

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Nafisa Rehmani | AP22110010598	**Kunj Mittal | AP22110010545**	**Rishabh Ranjan | AP22110010339**
Krishna Sharma | AP22110010128	**Kunal Sharma | AP22110010172**	**Nivedha Sriram | AP22110010510**
	Sonu Sarojini | AP22110010609	

Under the Guidance of

Dr. Tonmoya Sarmah

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[April, 2025]

Certificate

Date: 29-April-2025

This is to certify that the work present in this Project entitled "**Movie Streaming Application**" has been carried out by **Krishna Sharma, Nafisa Rehmani, Sonu Sarojini, Rishabh Ranjan, Nivedha Sriram, Kunj Mittal, and Kunal Sharma** under my supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Dr. Tonmoya Sarmah

Designation,

Professor

Acknowledgment

We sincerely express our gratitude to **Dr. Tonmoya Sarmah,** our faculty for Mobile Application Development with JAVA at SRM University AP, for his guidance and support throughout our project, **"Movie Streaming Application"** Her insights were invaluable in refining our approach and enhancing the quality of our work.

We extend our heartfelt thanks to our **Head of the Department (HOD),** and **Dean,** , for providing us with the opportunity and resources to work on this project. Their support has been instrumental in our learning journey.

We also appreciate our **teammates and peers** for their collaboration and knowledge-sharing, making this experience truly enriching. Lastly, we thank our **families and friends** for their constant encouragement and motivation.

Nafisa Rehmani | AP22110010598

Krishna Sharma | AP22110010128

Sonu Sarojini | AP22110010609

Rishabh Ranjan | AP22110010339

Nivedha Sriram | AP22110010510

Kunj Mittal | AP22110010545

Kunal Sharma | AP22110010172

Table of Contents

Project Title and Novelty

1. Movie Streaming App — Desktop Version (Java AWT)

Title:
Movie Streaming App (Desktop using Java AWT)

Novelty:
Unlike modern web-based or heavy desktop platforms, this project implements a lightweight, desktop-based movie streaming simulation using pure Java AWT. It caters to users with limited internet, no browser dependency, and low-end system compatibility. The project showcases API integration, desktop UI design without external frameworks (no JavaFX, no Swing reliance), and responsive multithreading for smooth media interaction.

2. Movie Streaming App — Mobile Version (Android Studio)

Title:
Movie Streaming App (Mobile using Android Studio)

Novelty:
Complementing the desktop version, this project brings the same streaming simulation experience to **Android mobile devices**. Built using **Android Studio (Java)**, it emphasizes a **native mobile user experience** with optimized layouts, API data fetching, and background processing (via **AsyncTask**, **LiveData**, or **Kotlin Coroutines**) to ensure responsive interaction. It highlights adaptability of the same core concept across platforms: **desktop and mobile**.

Project Background and Motivation

- ## Background

Movie streaming services dominate today's entertainment landscape, offering users the ability to access vast libraries of content anywhere and anytime. However, most of these services rely heavily on **modern web browsers** or **platform-specific native apps** that demand significant system resources and high-speed internet connectivity.

This project addresses that gap by building a **lightweight movie streaming app prototype** in two versions:

- A **desktop-based app using Java AWT**, and
- A **mobile app using Android Studio**.

The **Java AWT version** demonstrates how users can **browse**, **search**, and **simulate playback** of movies without relying on heavy frameworks, modern browsers, or advanced system configurations.
The **Android version** brings the same streaming experience to mobile users, showing platform adaptability and mobile UI optimization.

Both versions integrate API-based data fetching and are designed to provide a smooth user experience despite the platform constraints.

- ## Motivation

Creating a full-featured, UI-based application purely with **Java AWT** is challenging because AWT is an **older, low-level UI toolkit** lacking the rich components and conveniences of modern libraries like JavaFX, React Native, or Flutter. This project is motivated by the challenge of:

- **Building responsive interfaces** using only AWT.
- **Handling real-time operations** like API calls and playback simulation through **multithreading**.
- **Optimizing performance** without external UI frameworks.

Similarly, the **Android version** was developed to **translate the same functionality into a mobile-native experience**, showcasing mobile development skills such as:

- Designing clean, responsive Android layouts.
- Using background threading for smooth API interaction.
- Managing different device capabilities and network conditions.

This dual-platform project demonstrates that, with smart design and engineering, efficient and attractive streaming simulations can be created **even on resource-constrained environments**, without always depending on modern, heavy frameworks.

Abstraction

The **Movie Streaming App** project is a cross-platform simulation of a modern streaming service, developed in two parallel versions:

1. A **desktop-based prototype using Java's Abstract Window Toolkit (AWT)**, and
2. A **mobile-native application built using Android Studio**.

In an era where multimedia applications predominantly rely on advanced front-end frameworks and platform-specific SDKs, this project explores how lightweight, responsive, and user-friendly interfaces can still be developed using both **traditional Java AWT** and **modern Android tools**.

Desktop (Java AWT) Version:

This version showcases the capability of the legacy Java AWT framework to deliver an interactive movie-browsing experience. The app integrates with the **TMDb (The Movie Database) API** to fetch real-time movie metadata, including titles, posters, descriptions, genres, and ratings. Key features include:

- **Search functionality** based on movie titles.
- **Filtering options** by genre, rating, and release year.
- **Sorting mechanisms** based on popularity or user ratings.
- **Simulated playback**, where clicking on a movie provides an option to open an external link (e.g., YouTube trailer) in the default web browser.

Despite AWT's inherent limitations—such as lack of native multimedia support—the app demonstrates clever use of design simulations to replicate playback features. It uses **custom panels, layouts, and button styling** to elevate the minimalistic AWT UI. Responsive behavior is achieved using **multithreading**, especially during network operations like API calls and image loading. The **Gson library** is used for efficient JSON parsing and data management.

Mobile (Android Studio) Version:

The Android counterpart mirrors the core features of the desktop app while utilizing the full potential of Android's rich UI components and lifecycle management. Key highlights include:

- **Intuitive mobile layout** with RecyclerViews, CardViews, and Navigation Components.
- **Asynchronous API calls** using Retrofit and background threading.
- **Efficient image loading** using Glide.
- **Intent-based redirection** for playback via YouTube or embedded links.

This version provides a native, touch-optimized experience while maintaining functional parity with the AWT desktop version. It demonstrates best practices in Android development, such as view-model separation, API integration, and responsive UI rendering.

Together, the Java AWT and Android Studio implementations of the Movie Streaming App form a **robust proof of concept** that media-centric applications can be effectively built across different platforms, even with varying technical constraints. This project illustrates the power of solid design, creative use of system resources, and modern API integration—bridging the gap between legacy development and current mobile-first demands.

Abbreviations

Abbreviation	Full Form
AWT	Abstract Window Toolkit
API	Application Programming Interface
TMDb	The Movie Database
GUI	Graphical User Interface
JSON	JavaScript Object Notation
IDE	Integrated Development Environment
SDK	Software Development Kit
UI	User Interface
UX	User Experience
JVM	Java Virtual Machine
XML	Extensible Markup Language
HTTP	HyperText Transfer Protocol
URL	Uniform Resource Locator
GSON	Google JSON (Java library for JSON parsing)
APK	Android Package Kit

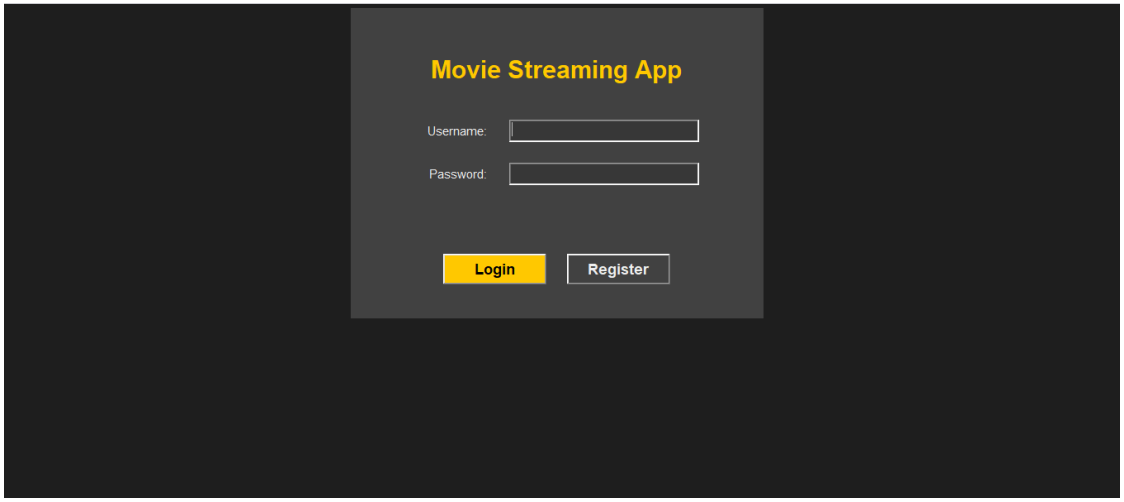
API Level	Android Platform Version Level
RecyclerView	A flexible view for providing a scrollable list
Retrofit	A type-safe HTTP client for Android
Glide	An image loading and caching library for Android
Intent	Messaging object used to request an action from another app component
Gradle	Build automation tool used in Android Studio
ViewModel	Architecture component that manages UI-related data
Activity	A single focused thing the user can do in Android
Fragment	A modular section of an Activity, used for UI design

List of Figures

- **Figure 1 (Login Panel):**

AWT Version:

The initial screen users encounter when launching the desktop app. It features a simple, clean interface constructed with AWT components like Label, TextField, and Button. Although authentication is simulated (accepts any input for demonstration), the design imitates a real-world login system, demonstrating input handling and event-driven programming.



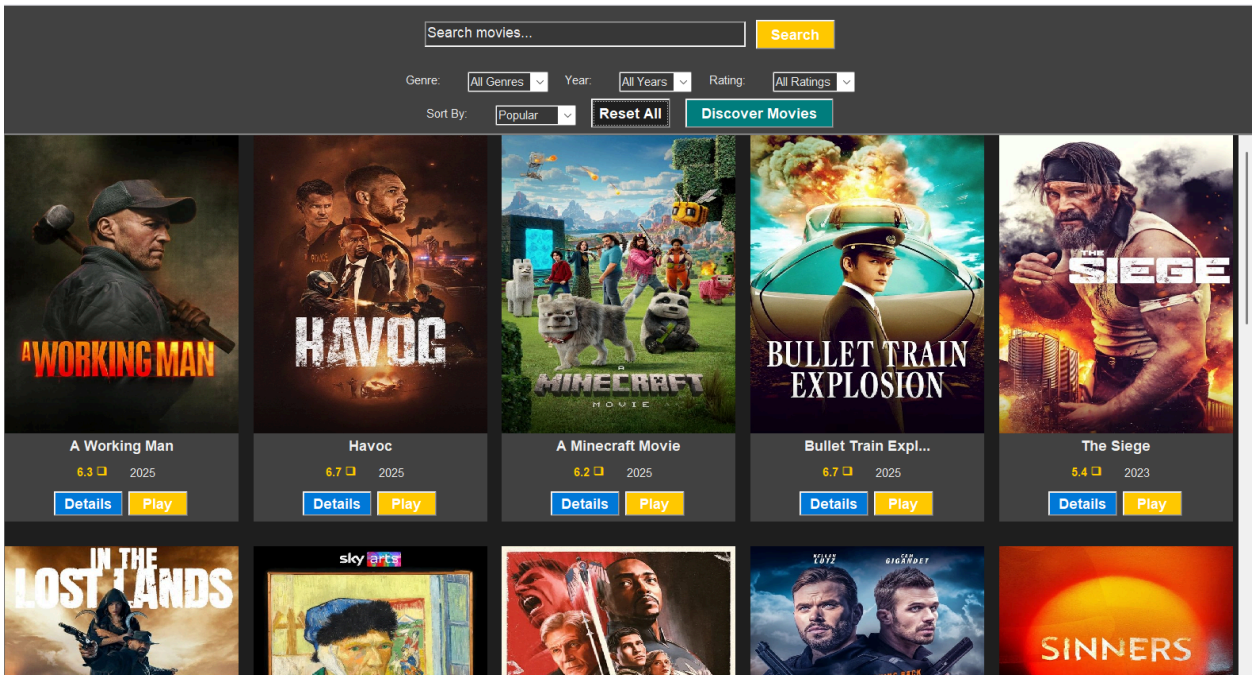
Android Studio Version:

Similarly, the Android app starts with a mobile-friendly **Login Activity** designed using **XML layouts**. It utilizes EditText fields for username/password and a Button for login, combined with **material design components** for a modern look. Validation is basic for demo purposes but mimics a real authentication flow.

- **Figure 2 (Movie Browsing Grid):**

AWT Version:

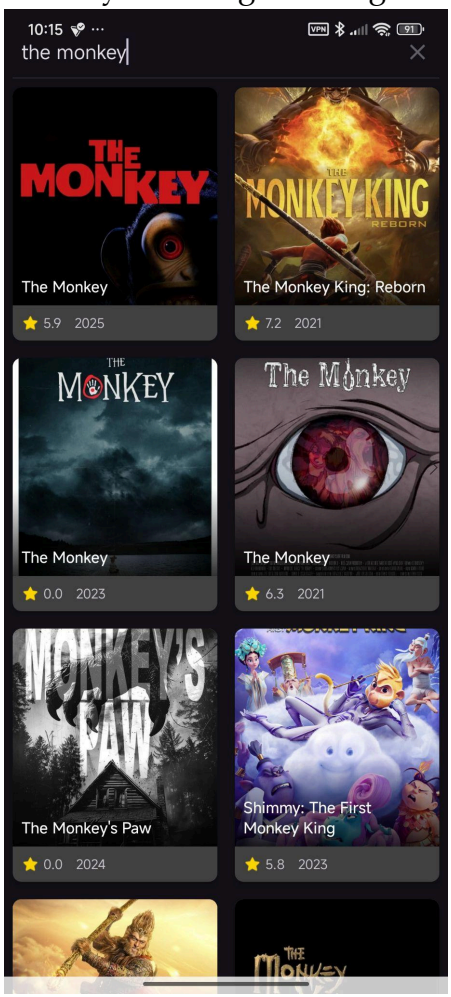
Upon login, users are redirected to the **Movie Browsing Screen**. Movies are displayed using a GridLayout, dynamically populated with movie posters and titles fetched live from the TMDb API. Basic hover effects (changing border colors or highlights) and scroll handling are implemented for a smoother desktop experience.



Android Studio Version:

The Android app presents the movies in a **RecyclerView GridLayout**, offering a

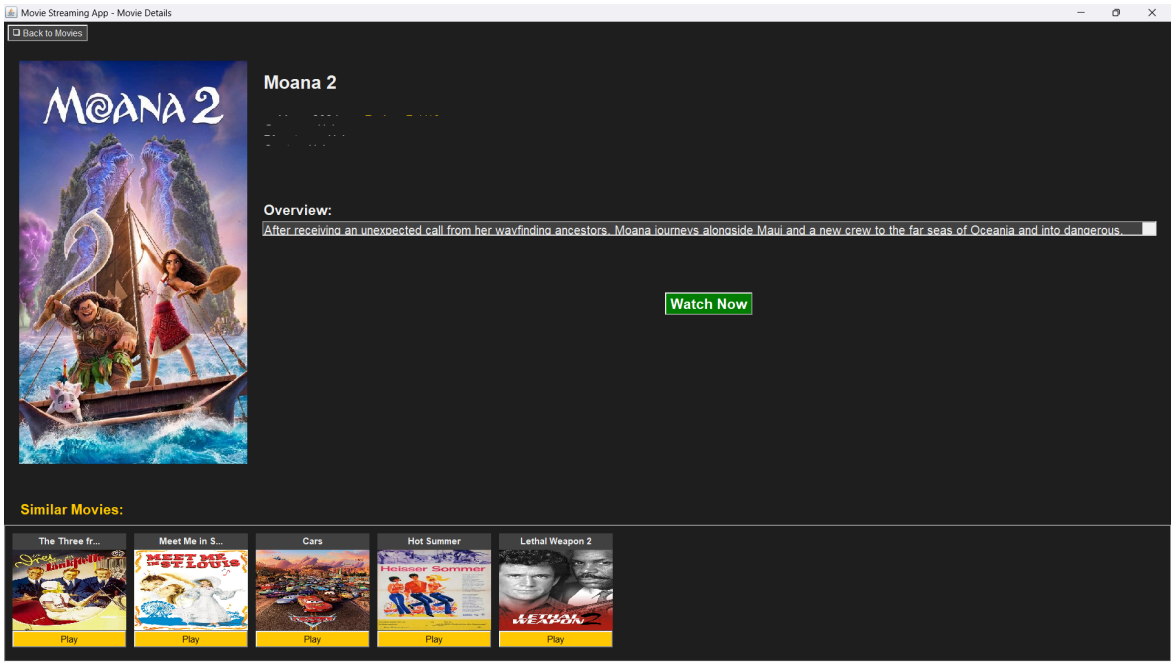
responsive, scrollable grid of movie posters and titles. Images are loaded efficiently using the **Glide** library. Smooth scrolling, animations, and efficient memory handling are integrated to provide a modern mobile experience.



- **Figure 3 (Movie Detail Page):**

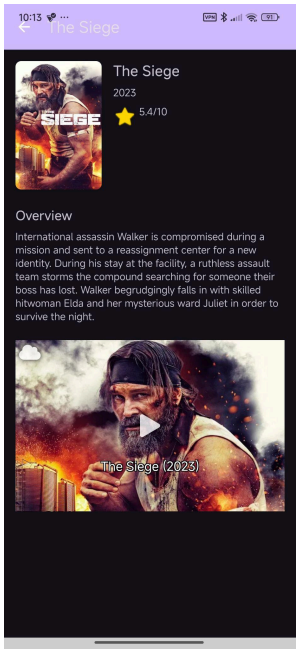
  AWT Version:

  Clicking a movie poster opens a detailed view. It provides essential information like overview, rating, genre, release date, director, and cast. Additionally, a carousel or horizontally scrollable list of "Similar Movies" is shown to enhance discovery.



Android Studio Version:

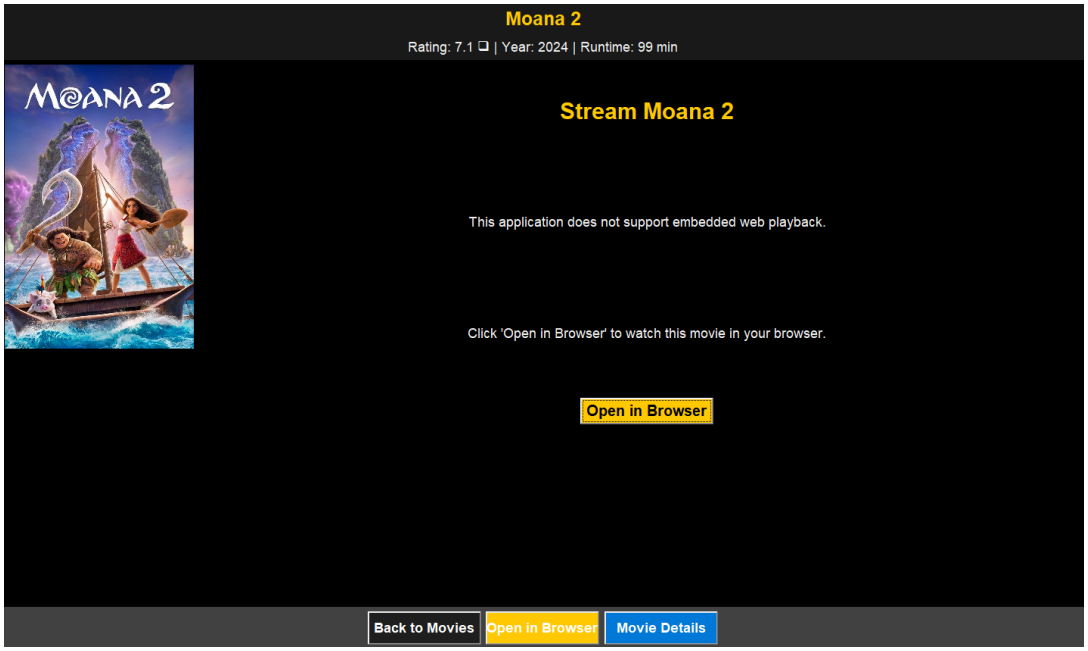
In the Android app, tapping a movie card navigates to a **Detail Activity** or **Fragment**. Rich information is displayed using TextViews, ImageViews, and RecyclerView for similar movies. The screen features elegant transitions, collapsible toolbars (using CollapsingToolbarLayout), and responsive layouts for phones and tablets.



- **Figure 4 (Redirect Popup):**

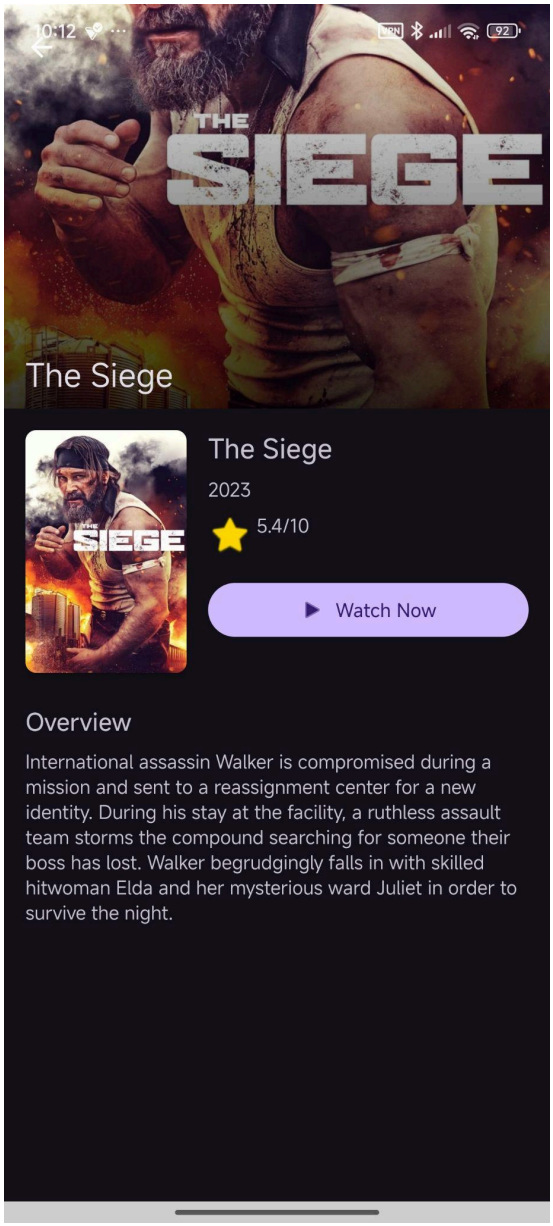
AWT Version:

Since AWT does not support embedded video streaming, clicking the **"Play"** or **"Watch Movie"** button pops up a small `Dialog` window asking users for confirmation. Upon confirmation, the app opens the default web browser to stream the video via an external link (e.g., YouTube or another service).



Android Studio Version:

On Android, clicking the **Watch Now** button triggers an **AlertDialog** asking for user confirmation. If the user agrees, the app uses an **Intent** to open the external streaming link in a mobile browser or supported app like YouTube. This maintains a clean separation between browsing and actual media playback, ensuring fast performance.



Code Snippets and Results

Movie Fetching Code

```
public static List<Map<String, Object>> getPopularMovies() {

    String url = TMDB_BASE_URL + "/movie/popular?api_key=" + TMDB_API_KEY;

    String response = makeApiCall(url);

    return parseMoviesResponse(response);

}
```

Poster Loading in Background

```
new Thread(() -> {

    posterImage = Toolkit.getDefaultToolkit().getImage(posterUrl);

    repaint();

}).start();
```

Results:

- Posters loaded without freezing UI.
- API fetch successful, displayed real-time popular movies.

Introduction

The Movie Streaming App demonstrates the capability of developing desktop-based and mobile-based, API-driven movie browsers using legacy tools like Java AWT and Android Studio. The app supports login, browsing, searching, filtering, and redirection to movie Streaming.

Project Overview

- User Login and Demo Registration.
- Grid Display of Movies fetched dynamically from TMDb.
- Search Functionality based on title, genre, year, and ratings.
- Detail View with movie redirection.
- Responsive and lightweight UI even without modern libraries.

Features and Structures

- Authentication: (Demo Mode)
- Movie Browsing: Real-time API data display.
- Movie Details: Rating, cast, similar movies shown.
- Movie View: Redirects to a web-based player.
- Search and Filter: Genre, rating, year.

Design and Functionality

- UI Panels:
    - LoginPanel.java
    - MovieListPanel.java
    - MovieDetailPanel.java
    - PlayerPanel.java

- CardLayout: Switching between Login, Browse, Details.

- Modularity: Clear separation between UI and API logic.

System Architecture

The system architecture for the Movie Browser Application can be divided into the following main layers:

- **User Interaction Layer (Front-End):**
  - Java AWT-based and Android Studio GUI for login, search bar, movie cards, and navigation.
  - Users input movie queries, browse movies, and select options.

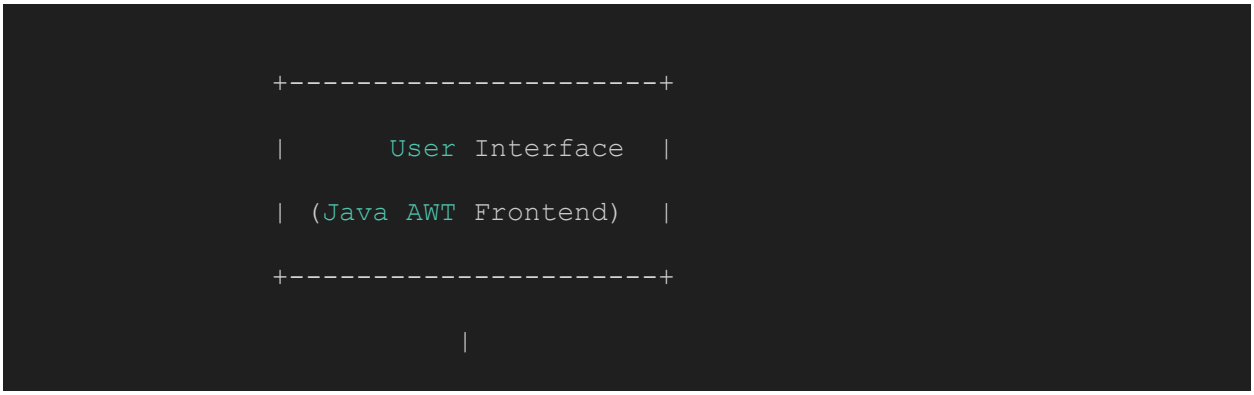
- **Controller Layer:**
  - Handles the user actions like clicking "search" or selecting a movie.
  - Manages GUI updates according to user inputs and API responses.

- **Backend/API Communication Layer:**
  - Sends HTTP requests to the TMDb API with query parameters (e.g., search keyword).
  - Receives movie data including JSON formatted metadata.
  - Parses JSON data using Gson and maps it into Java objects.

- **External System (TMDb API):**
  - Provides movie information, poster URLs, and streaming links dynamically.
  - Acts as the major data source for the application.

Simple Flow:
User Input → Controller → TMDb API Request → JSON Response → Data Parsing → GUI Update → Redirection

System Architecture of the Movie Application




v
+----------------------+
| Controller Layer |
| (Handles user input, |
| triggers API calls) |
+----------------------+
|
v
+----------------------+
| API Handler |
| (HTTP Requests to |
| TMDb API, handle |
| responses) |
+----------------------+
|
v
+----------------------+
| JSON Parser |
| (Gson parses API |
| responses into |
| Java Objects) |
+----------------------+
|
v
+----------------------+
| GUI Updater |
| (Update movie cards, |


| images, play) |
+----------+
|
v
+----------+
| External Resource: |
| TMDb API |
| (Movie details, |
|movie streaming, images) |
+----------+

Tools and Technologies Used

- **Java SE (Standard Edition):**
  Core programming language used for building the application backend and frontend with AWT (Abstract Window Toolkit).

- **Java AWT (Abstract Window Toolkit):**
  Library used to create the graphical user interface (GUI) components like frames, buttons, panels, labels, and text fields.

- **Gson Library:**
  Used for parsing JSON responses received from The Movie Database (TMDb) API to easily map them into Java objects.

- **The Movie Database (TMDb) API:**
  External API that provides access to movie data including title, poster image, overview, movie streaming links, ratings, etc.

- **IntelliJ IDEA / Eclipse (IDE):**
  Integrated Development Environment used to write, debug, and organize the Java project.

- **Operating System:**
  Windows 10 for development and testing purposes.

- **Internet Browser (for Viewing):**
  Used to open the movie URLs obtained via API.

Methodology

Requirement Gathering and Analysis

- Needed minimal GUI components (AWT).
- Required real-time movie data from TMDb API.

Design and Planning

- Planned modular classes.
- API calls handled separately from UI.

Development

- Implemented Login and Registration panels.
- Integrated Movie Database API.
- Added multithreading for poster loading.

Testing and Optimization

Unit Testing:

- Individual Java classes were tested for functionality such as API connection, JSON parsing, and GUI response.
- Special attention was given to exception handling during API failures.

Integration Testing:

- After combining GUI components with API communication, the end-to-end functionality was tested.

- Tests involved scenarios like empty search, wrong movie names, and Movie redirection validation.

GUI Testing:

- Manual testing of button clicks, text field entries, and event listeners.
- Ensured that the search, grid layout, and detail popup are responsive and user-friendly.

Performance Testing:

- Observed response time of API calls and GUI loading.
- Verified that the application does not freeze even during multiple rapid search queries.

Error Handling Testing:

- Checked how the system behaves if the API key is wrong, no network is available, or TMDb returned invalid results.

Deployment and Maintenance

- Desktop executable JAR file generated.
- Future plans: SQLite database for account storage.

Discussion

Project Overview

Successfully implemented lightweight movie streaming apps on desktop (Java AWT) and mobile (Android Studio), demonstrating cross-platform adaptability using API-driven design.

Design

- **AWT App**: Utilized CardLayout for flexible navigation, custom hover effects, and responsive panels.
- **Android App**: Employed RecyclerView, Fragments, and Material UI components for modern, mobile-friendly navigation and interaction.

Technical Implementation

Java AWT App:

- GUI built with AWT.
- TMDb API for movie metadata.
- JSON parsing using Gson.
- Multithreaded operations for responsiveness.

Android Studio App:

- Java for frontend and backend logic.
- API data fetched via Retrofit.
- Glide for image loading.
- Navigation Component and ViewModel for structured UI/UX.

Challenges and Solutions:

Challenge	Solution
Lack of advanced components in AWT	**Created customized buttons and layouts.**
No embedded media playback (AWT)	**Redirected users to external browsers.**
TMDb API rate limit	**Caching genre data locally.**
Maintaining responsiveness (Android)	**Used ViewModel and LiveData for UI state management.**
Efficient image loading (Android)	**Integrated Glide for smooth and cached image loading.**

Limitations

Although the Movie Browser Application successfully meets its core objectives, there are certain limitations in both the desktop (Java AWT) and mobile (Android Studio) versions:

- **Static Login (AWT)**: Uses hardcoded credentials; no secure auth.
- **No Embedded Playback (AWT)**: Videos open externally; lacks built-in player.
- **In-App Playback (Android)**: Supported via native video player components.
- **Basic Error Handling**: Minimal feedback on failures.
- **No Pagination**: All results load at once; no scroll-based loading.
- **Language Limitation**: English-only content; no region filtering.
- **API Dependency**: Heavy reliance on TMDb API availability.

These limitations can be addressed in future enhancements.

Conclusion

The **Movie Streaming App** showcases the development of a cross-platform movie browsing interface through two implementations: a **Java AWT desktop app** and an **Android Studio mobile app**.

The AWT version proves that even with limited GUI capabilities, it is possible to build a responsive and interactive application using **multithreading**, **API integration (TMDb)**, and **custom component design**. Users can browse movies, view details, filter results, and simulate playback through external redirection—all within a lightweight desktop environment.

The Android version complements this with a modern, native mobile experience using **Material UI**, **RecyclerView**, **Glide**, and **Intents**, offering smoother visuals and better performance on handheld devices.

Together, both versions demonstrate strong design, adaptability, and effective use of APIs across legacy and modern platforms. This project serves as a proof of concept that streaming interfaces can be both lightweight and functional on diverse systems.

Future Work

- **AWT App Enhancements:**
  - Embed video playback using **Java Media Framework (JMF)**.
  - Implement real **database-based login authentication**.
  - Add support for **user reviews and ratings**.
  - Improve visuals through **custom-styled AWT components**.

- **Android App Enhancements:**
  - Integrate **Firebase or SQLite** for secure login and user data.
  - Enable **in-app video playback** using **ExoPlayer**.
  - Add **push notifications** for new movie releases.
  - Include **user profiles**, **watchlists**, and **offline support**.

These upgrades aim to bring both versions closer to real-world streaming apps while showcasing the versatility of Java and Android platforms.

References

- **The Movie Database (TMDb) API Documentation**
  https://developer.themoviedb.org/

- **Java Abstract Window Toolkit (AWT) Documentation**
  https://docs.oracle.com/javase/8/docs/api/java/awt/package-summary.html

- **Gson Java Library for JSON Parsing**
  https://github.com/google/gson

- **Official Java Networking API Documentation**
  https://docs.oracle.com/javase/tutorial/networking/

- **Java Event Handling and GUI Programming Guide**
  https://docs.oracle.com/javase/tutorial/uiswing/events/index.html

- **Java Multithreading and Concurrency Concepts**
  https://docs.oracle.com/javase/tutorial/essential/concurrency/

- **AWT Layout Managers and GUI Design**
  https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

- **Streaming Media Concepts: What is Video Streaming?**
  https://www.cloudflare.com/learning/video/what-is-video-streaming/

- **Building a Simple Media Streaming Server in Java**
  https://medium.com/@rossbulat/build-a-simple-streaming-server-with-java-1dc91405ad22

- **YouTube Data API Documentation** *(for redirect functionality)*
  https://developers.google.com/youtube/v3

- **Principles of Good UI/UX Design**
  https://www.nngroup.com/articles/ten-usability-heuristics/

- **Introduction to Multimedia Systems (Overview for Streaming Technologies)**
  https://link.springer.com/book/10.1007/978-3-540-92892-8

- **OpenJDK Official Documentation**
  https://openjdk.org/

- **HTTPURLConnection Tutorial for Java Developers**
  https://www.baeldung.com/java-http-request

- **How Netflix Works: The (Simplified) Backend Explained**
  https://netflixtechblog.com/