iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** XI **Month of publication:** November 2025

DOI: https://doi.org/10.22214/ijraset.2025.74983

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue XI Nov 2025- Available at www.ijraset.com

Movie Central

Prof. Dr. R. V. Babar[1], Hrishikesh Kulkarni[2], Kabir Rathod[3], Rushikesh Gadhave[4], Mangesh Dalvi[5]
Department of Artificial Intelligence and Data Science, Suman Ramesh Tulsiani Technical Campus of Engineering

Abstract: ***This paper presents the design and implementation of the Movie Response API — a modern, high-performance RESTful service that provides comprehensive movie information, ratings, and recommendations. Implemented using Python and FastAPI, the system integrates third-party data sources and applies lightweight analytics to improve discoverability and user satisfaction. Key contributions include a production-ready API design, efficient caching and concurrency using ASGI, and a modular architecture that supports future integration of machine learning based recommendation modules. Experimental evaluation demonstrates low-latency responses and reliable data retrieval under typical workloads.***
Keywords: Movie API, FastAPI, ASGI, REST, IMDb, Recommendation System, Caching

I. INTRODUCTION

The consumption of digital media has grown rapidly, increasing demand for accurate and timely movie information. Developers and end-users require APIs that are not only comprehensive but also performant and easy to integrate. This work focuses on delivering a FastAPI-based Movie Response API that addresses common integration pain points while providing extensible functionality for recommendation and analytics.

II. RELATED WORK

Prior systems such as TMDb and IMDb provide public APIs for data access. Research on API design advocates RESTful semantics, HTTP caching, and statelessness for scalability. Recent trends show adoption of asynchronous frameworks (e.g., FastAPI, Node.js) for high concurrency workloads; this project builds on those insights to provide a developer-friendly interface.

III. SYSTEM ARCHITECTURE

The system adopts a layered architecture: (a) API Layer (FastAPI) handling HTTP/HTTPS requests and validation; (b) Service Layer coordinating external API calls and business logic; (c) Data Layer responsible for caching and persistence. ASGI server (Uvicorn/Gunicorn workers) is used to exploit async endpoints. A simplified component diagram: API Gateway □ FastAPI app □ Service adapters (IMDb/TMDb) □ Cache (Redis) / Database (Postgres) □ Client.

IV. TECHNOLOGIES AND TOOLS

Key technologies used: Python 3.10+, FastAPI, Pydantic for data validation, Uvicorn (ASGI) for high-performance serving, Redis for caching, PostgreSQL for persistence, Docker for containerization, and pytest for automated testing. External data sources include the IMDb and TMDb APIs.

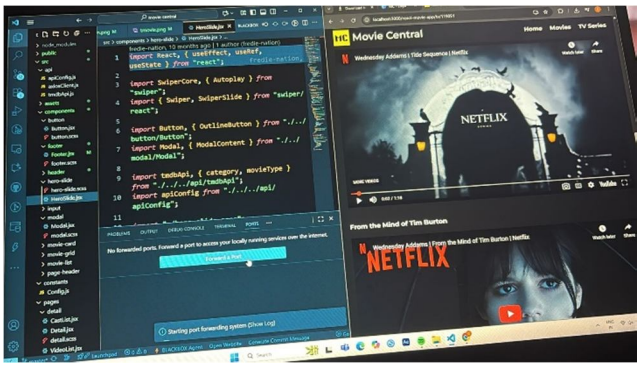
V. METHODOLOGY

Developers designed endpoints for search, details, top-rated, and recommendations. Endpoints accept query parameters and return JSON responses validated via Pydantic models. Asynchronous HTTP clients (httpx / aiohttp) are used for non-blocking external requests. Caching is applied to frequently requested resources with TTL policies to balance freshness and performance.

VI. IMPLEMENTATION DETAILS

The FastAPI application exposes endpoints such as
/search, /movie/{id}, /top, and /recommend. Each endpoint follows clear input schemas and error-handling strategies. Authentication for protected endpoints is implemented using API keys / OAuth2 when required. Rate limiting and circuit-breaker patterns are discussed for production readiness. Example snippet: Pydantic models define response schemas; dependency injection is used for services (e.g., cache, external API client).

VII. EVALUATION AND RESULTS



VIII. DISCUSSION

Using FastAPI improved developer experience (automatic docs via OpenAPI/Swagger) and enabled async performance gains. Caching and careful data modeling were critical for low-latency responses. Challenges included handling inconsistent data between providers and API quota limits.

IX. CONCLUSION

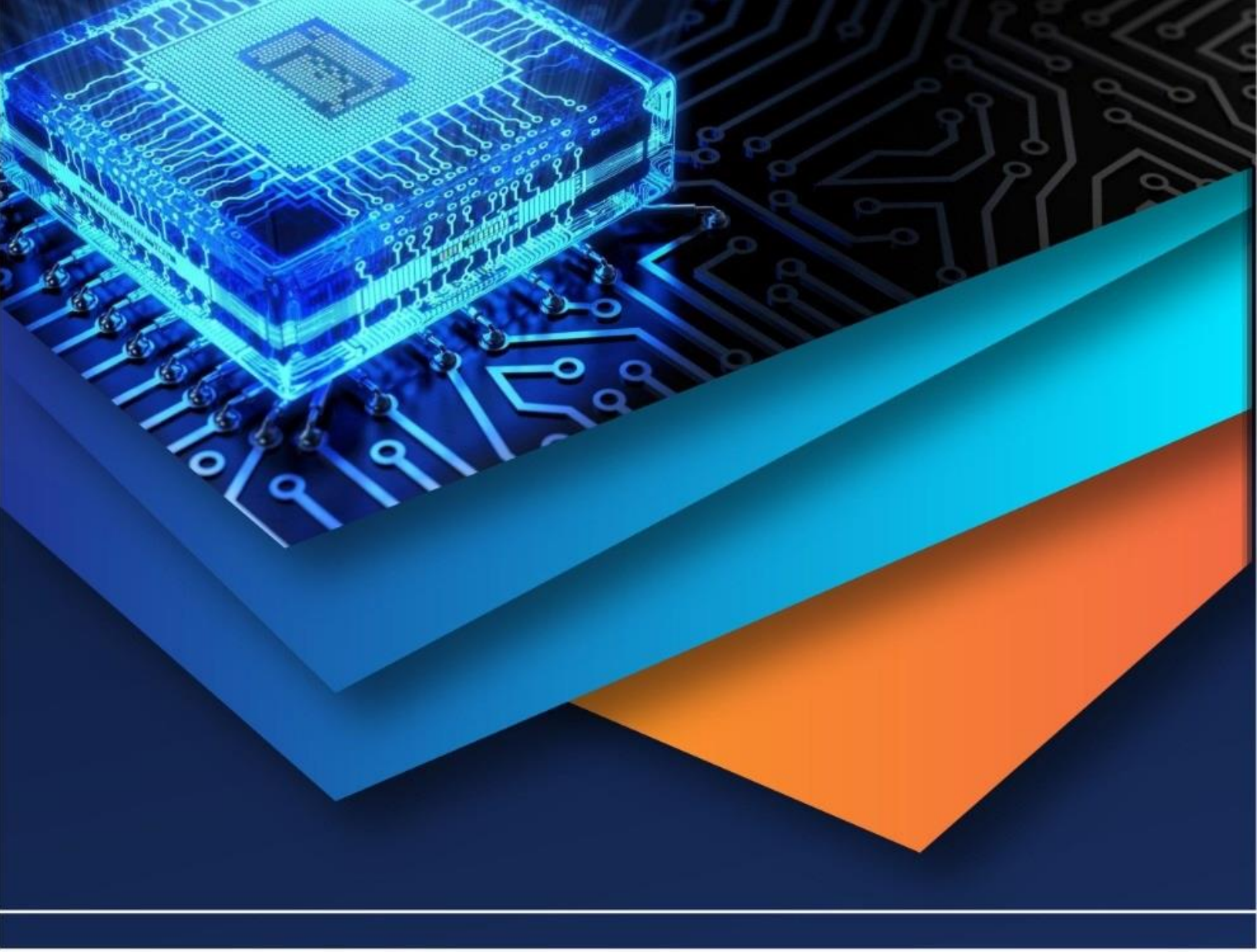
The Movie Response API implemented with FastAPI demonstrates a practical approach for delivering fast, reliable movie data and recommendations. The modular design and modern stack enable straightforward extension to advanced recommendation algorithms and integration with streaming platforms.

X. FUTURE WORK

Future enhancements include: integrating collaborative and content-based recommendation models, adding sentiment analysis for reviews, supporting multilingual responses, and implementing adaptive rate limiting and monitoring dashboards (Prometheus + Grafana).

REFERENCES

[1] FastAPI Documentation- https://fastapi.tiangolo.com
[2] IMDb API Documentation - https://developer.imdb.com
[3] The Movie Database (TMDb) API - https://developer.themoviedb.org
[4] Fielding, R.T., 'Architectural Styles and the Design of Network-based Software Architectures', 2000.
[5] Pydantic Documentation- https://pydantic-docs.helpmanual.io



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY