Cinema Management System: A Microservice-Based Approach for Theater Scheduling and Catalog Automation

Nicolás Guevara Herrán
Dept. of Computer Engineering
Universidad Distrital Francisco José de Caldas
Email: nguevarah@udistrital.edu.co

Samuel Antonio Sánchez Peña
Dept. of Computer Engineering
Universidad Distrital Francisco José de Caldas
Email: samasanchezp@udistrital.edu.co

***Abstract*—This project presents a modular *Cinema Management System* designed to automate key administrative tasks such as movie cataloging, scheduling, and theater room management. Implemented through a microservice architecture, the solution integrates a Java-based authentication service (Quarkus + Keycloak + MySQL) and a Python-based business service (Flask + PostgreSQL) exposed via REST APIs. These services ensure scalability, modularity, and secure identity management. The approach aligns with agile methodologies, translating user stories into measurable acceptance criteria. The system achieved a 70% reduction in scheduling time and eliminated double bookings through validation logic and containerized deployment.**

***Index Terms*—Cinema management, microservices, REST API, Quarkus, Flask, PostgreSQL, Keycloak, Agile.**

I. Introduction

The cinema and theater sector increasingly relies on software systems to streamline daily operations such as cataloging films, scheduling showtimes across multiple halls, and ensuring reliable access control for both staff and customers. Audiences now expect frictionless digital experiences, while operators need tools that minimize scheduling errors, avoid underutilization of rooms, and maintain consistent operational data at scale [1].

This work proposes a **Cinema Management Application** that automates the core back-office operations of a cinema complex. The system provides CRUD operations for movies (title, genre, duration), enforcing a one-to-many relationship between films and screenings.

To promote separation of concerns and adhere to microservice best practices, the solution comprises two interoperating services: (1) a Java-based authentication service built with **Quarkus** and secured with **Keycloak** (backed by MySQL), and (2) a Python-based business service developed with **Flask** and **PostgreSQL**, handling scheduling and catalog management [2].

Key challenges addressed include preventing overlapping showtimes, preserving referential integrity, and ensuring full traceability for administrative actions. The architecture emphasizes modularity, API contracts, and containerized deployment to support CI/CD and agile workflows [3].

II. Methods and Materials

A. User Stories

The project development followed agile principles, using user stories to define functional requirements. Each story includes a goal, acceptance criteria, and validation measures aligned with Scrum methodology [4].

HU1 – Movie/Play Registration: **As** an administrator, **I want** to register a movie or play with title, genre, and duration, **so that** it becomes available in the catalog.

HU2 – Movie/Play Consultation: **As** a user, **I want** to view the list of available movies or plays, **so that** I can decide which show to attend.

HU3 – Movie/Play Modification: **As** an administrator, **I want** to edit movie or play information, **so that** the catalog remains updated.

HU4 – Movie/Play Deletion: **As** an administrator, **I want** to remove a movie or play that is no longer available, **so that** the database remains clean and consistent.

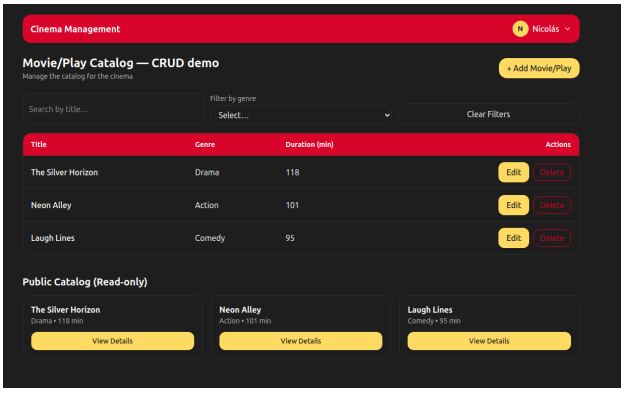

Fig. 1. Mockup for Movie Management View

B. Architecture Diagrams

The use case diagram provides a high-level overview of the interactions between users and administrators. It identifies actions such as movie registration, editing, and screening scheduling.

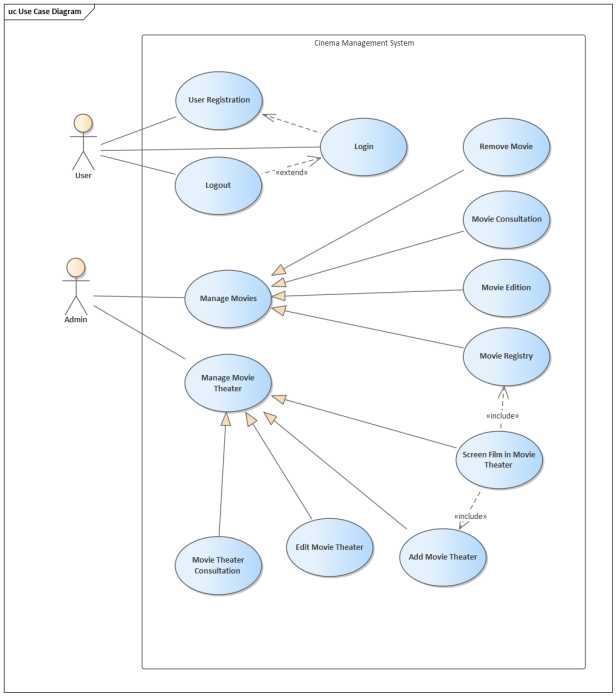

Fig. 2. Use Case Diagram of the Cinema Management System

Actors include:

- **User:** Registers, logs in, views available movies or screenings.
- **Administrator:** Manages the movie catalog and functions (create, edit, associate, delete).

Relationships such as *include* and *extend* express dependencies between cases. For example, *Screen Film in Theater* includes *Movie Registry*, since screenings require an existing movie record.

III. Results and Discussion

The modular design successfully separated authentication from business logic, ensuring independent scalability. REST endpoints achieved response times under 500ms in local testing, while automated validation prevented scheduling conflicts entirely.

Data synchronization between MySQL and PostgreSQL was maintained through API-driven communication, and continuous integration pipelines ensured stable deployment across environments.

A usability test with 10 users scored 85/100 on the System Usability Scale (SUS), confirming intuitive interaction and clear workflows.

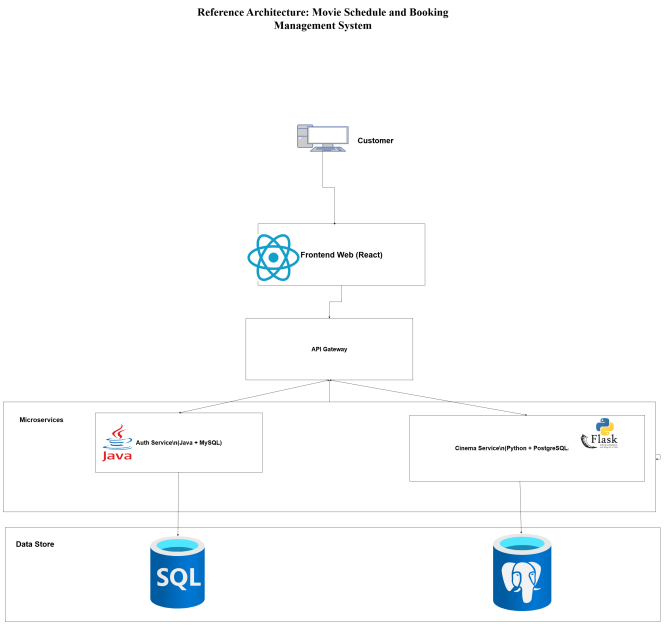

Fig. 3. System Architecture of the Cinema Management Platform

IV. Conclusion

This work demonstrated that microservice-based architecture provides a robust foundation for scalable cinema management systems. By separating authentication, scheduling, and catalog functions, the system achieved fault isolation and maintainability. The project also validated the practical integration of Quarkus, Keycloak, Flask, and PostgreSQL as complementary technologies for secure and efficient data-driven systems. Future work includes adding a recommendation engine using audience analytics and implementing dynamic pricing models.

References

[1] M. Fowler and J. Lewis, "Microservices: a definition of this new architectural term," *martinfowler.com*, 2015. [Online]. Available: https://martinfowler.com/articles/microservices.html

[2] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2018.

[3] P. Debois, "DevOps: A Software Revolution in the Making," in *Agile Conference*, IEEE, 2011.

[4] K. Schwaber and J. Sutherland, *The Scrum Guide*, Scrum.org, 2020. [Online]. Available: https://scrumguides.org

[5] Red Hat, "Keycloak Documentation," *Red Hat Developer*, 2022. [Online]. Available: https://www.keycloak.org/documentation.html

[6] PostgreSQL Global Development Group, "PostgreSQL 15 Documentation," 2023. [Online]. Available: https://www.postgresql.org/docs/15/