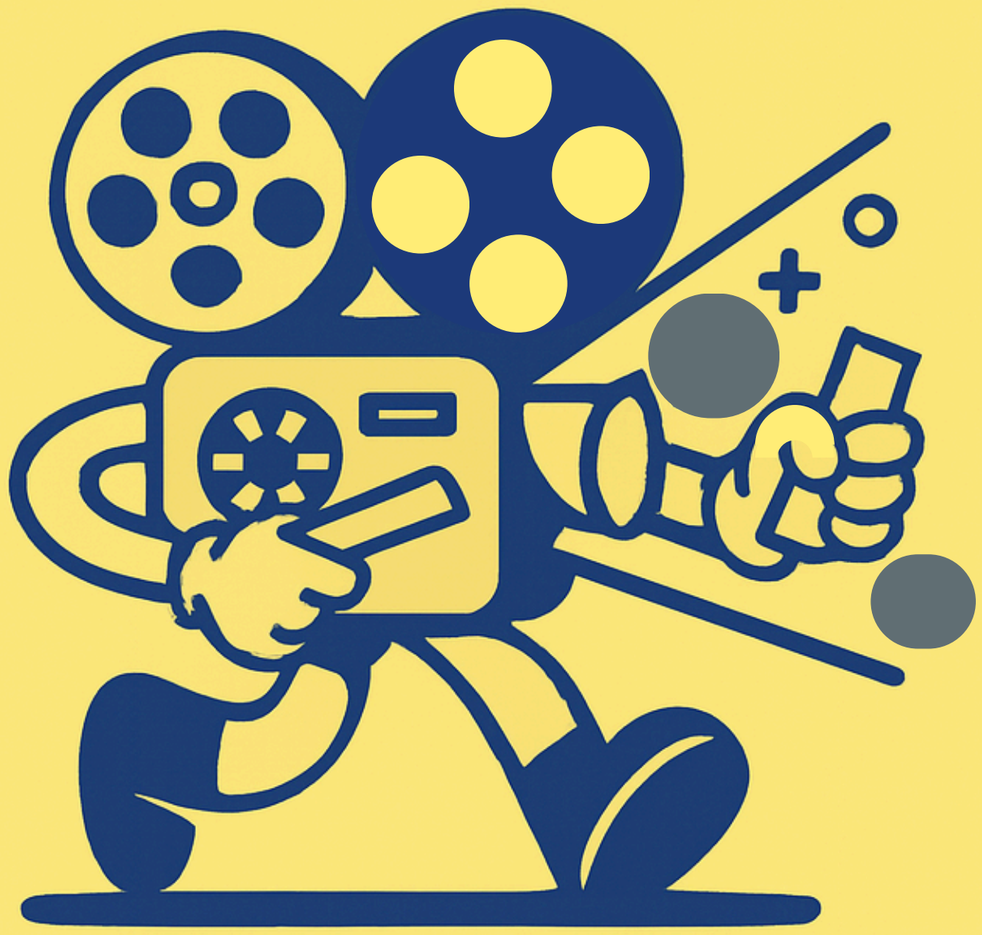


CINEMA SYSTEM

UNIVERSIDAD DISTRITAL FRANCISCO JOSÉ DE CALDAS
DEPARTMENT OF SYSTEMS ENGINEERING
TECHNICAL REPORT

NICOLÁS GUEVARA HERRÁN
SAMUEL ANTONIO SÁNCHEZ PEÑA
JORGE ENRIQUE ACOSTA JIMÉNEZ

SUPERVISOR: CARLOS ANDRÉS SIERRA

2. Introduction

Problem

The management of cinema scheduling often relies on manual methods (such as data entry in uncoordinated spreadsheets) or monolithic systems. This leads to a high incidence of human errors, results in data duplication, and causes programming inconsistency (overlapping showtimes, incorrect room assignment, outdated data). Furthermore, these approaches lack secure integration with user management.

Previous Solution

Legacy systems operated in a decentralized and on-premise manner, with showtime programming treated as an isolated, manual process. This lack of modular architecture drastically limited the real-time visibility of functions and made it impossible to ensure the uniqueness and integrity of programming data.

Challenge

Systematize and centralize the management of schedules and rooms through a modular and decoupled (microservices) architecture. The goal is to eliminate data duplication, automate consistency validations (preventing schedule/room conflicts), and provide a secure and scalable API base for the visualization and administration of showtimes.

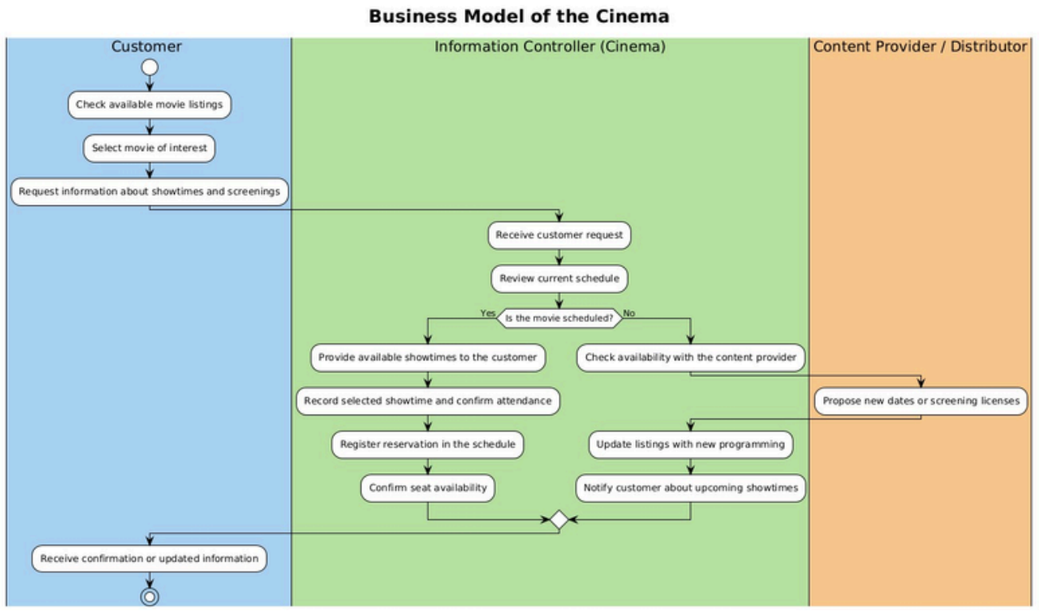


FIG:BUSINESS-PROCESS

3. Goal

Central Goal

To develop a modular and secure cinema management system based on microservices that enables efficient catalog and scheduling management.

Final Product

To build a scalable and secure architecture capable of reducing manual errors and improving operational efficiency in cinema administration.

4 .Proposed Solution

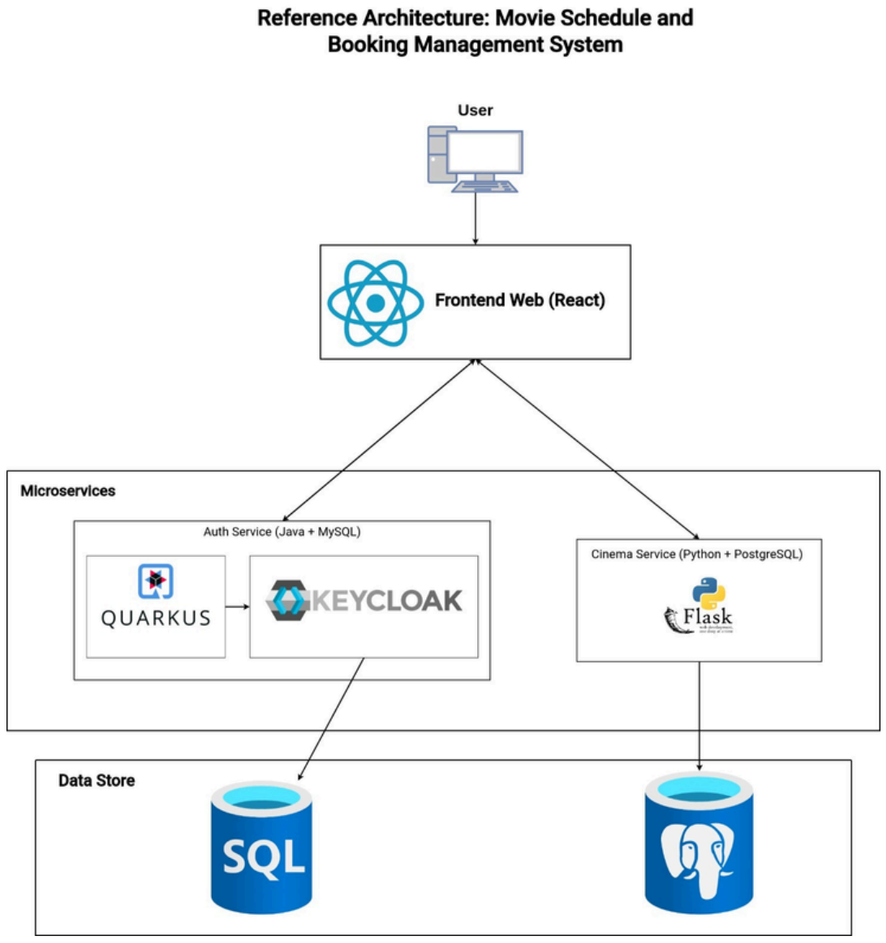


fig:architecture-diagram

The system follows a microservice architecture composed of two main, independent yet interoperable services: 1. Auth Service (Java/Quarkus + Keycloak + MySQL): Handles authentication, authorization, and secure access control. 2. Cinema Service (Python/Flask + PostgreSQL): Manages the core business logic, including movies, rooms, and screenings. Communication occurs through RESTful APIs. The system is containerized (Docker) and deployed via CI/CD pipelines (GitHub Actions to Google Cloud Run).

5. Experiments

Development Approach

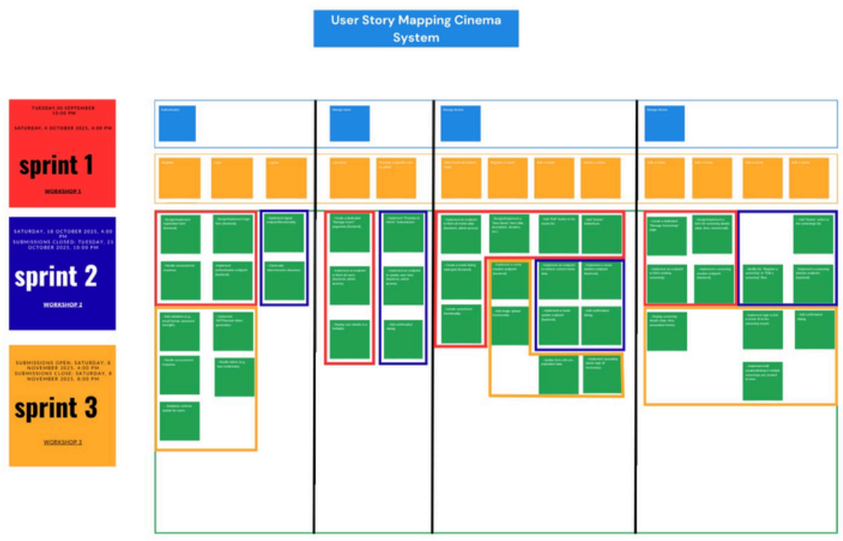
Agile methodology based on Scrum principles, with functional requirements organized using a User Story Map.

Testing and Validation Plan

A comprehensive testing strategy is defined to ensure robustness and correctness: - Unit Testing: Using pytest (Python) and JUnit (Java). - Integration Testing: To validate RESTful communication between containerized services (using Postman/cURL). - Performance Testing: Using JMeter to simulate concurrent requests.

Current Progress

The architectural design is complete, both services are functional in containerized environments, and integration between Keycloak and Flask has been successfully validated.



User Story Mapping Cinema System

6. Results

Strengths

- **Modularity & Maintainability: Clear separation of concerns (Auth vs. Cinema Service).**
- **Security: Use of Keycloak for IAM, reflecting modern best practices.**
- **Automation: CI/CD pipelines ensure consistent and rapid deployment.**

Limitations/Weaknesses

- **The project focused strictly on backend service orchestration; excluded features are payment processing, seat reservation, and advanced reporting.**
- **Development was within a fixed academic timeframe, limiting the depth of certain features.**

Key Outcome

- **Functional system foundation with integrated services that ensure autonomy, fault isolation, and clear separation of concerns, aligned with cloud-native best practices.**

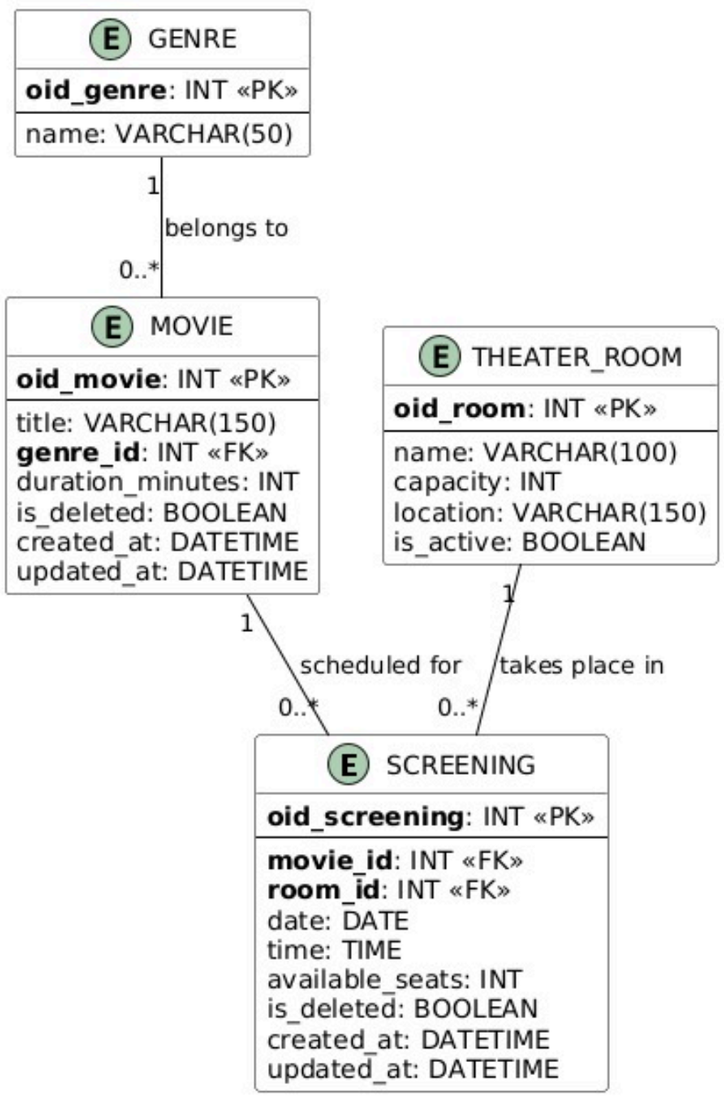


FIG:BUSINESS-PROCESS

7. Conclusions

Achievement

- **The project successfully established an architecture aligned with best practices for modularity, automation, and security, justifying the design decisions to enhance system maintainability and scalability.**

Future Work

- **Future stages will include load testing, usability analysis, and full deployment in the Azure Cloud environment (or finalize deployment in Google Cloud Run).**

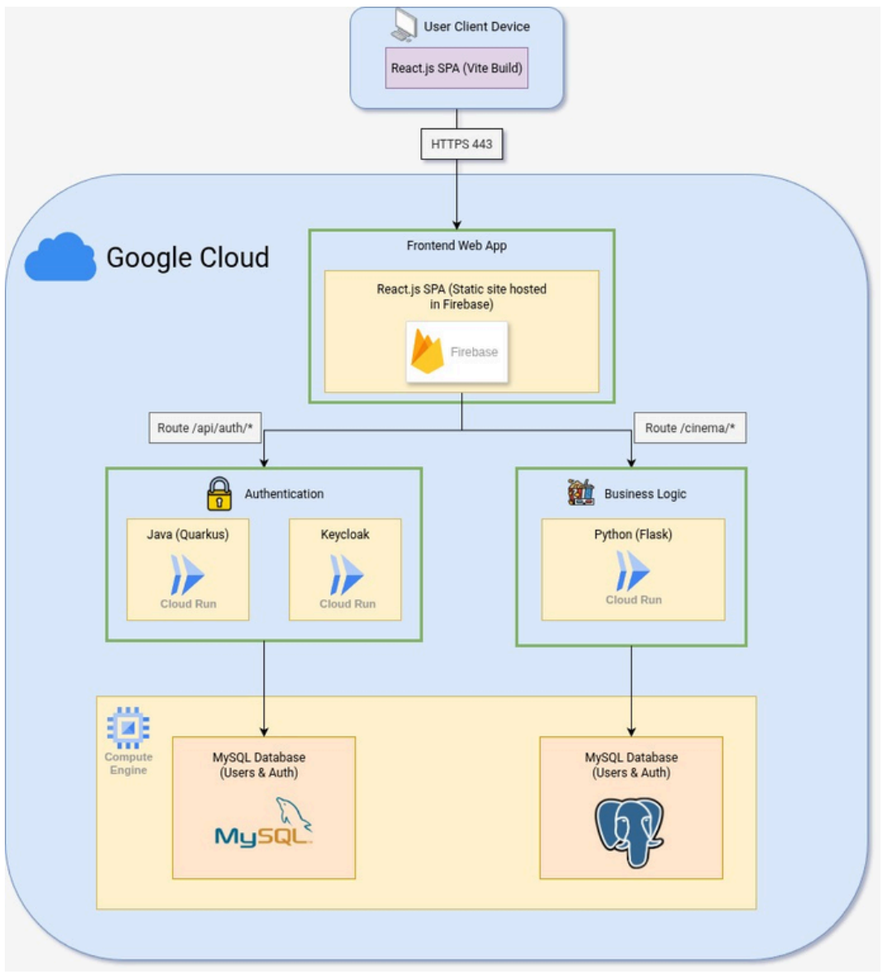


FIG:DEPLOYMENT DIAGRAM

8. Bibliography

- **Forsgren, N., Humble, J., & Kim, G. (2018). Accelerate: The Science of Lean Software and DevOps. IT Revolution Press.**
- **Fowler, M. (2003). Patterns of Enterprise Application Architecture. Addison-Wesley.**
- **Fowler, M. y Lewis, J. (2014). Microservices: a definition of this new architectural term. ThoughtWorks.**
- **Google Cloud. (2023). Building scalable and resilient web applications on Google Cloud.**
- **Google Cloud Architecture Center. (2023). Deployment and monitoring overview.**