Workshop 2 — Cinema Management

Nicolás Guevara Herrán — Samuel Antonio Sánchez Peña — Jorge Enrique Acosta Jiménez
Universidad Distrital Francisco José de Caldas

Introduction

This second workshop builds on the previously defined Cinema Management System and reports concrete design and implementation progress. It consolidates (i) updated class diagrams for the Auth (Java/Quarkus) and Cinema (Python/Flask) services, (ii) a reference architecture with an API gateway and service/data boundaries, (iii) a customer-centric business process model, and (iv) the Web UI status—implemented screens versus pending mockups. Together, these artifacts capture the current scope, interfaces, and next implementation steps.

Class Diagrams

The following class diagrams provide a detailed static view of the system's structure for both the Auth and Cinema microservices. They outline the key classes, their attributes, methods, and the relationships between them. These diagrams serve as a foundational blueprint for development, defining the data models and core business logic that each service will implement.

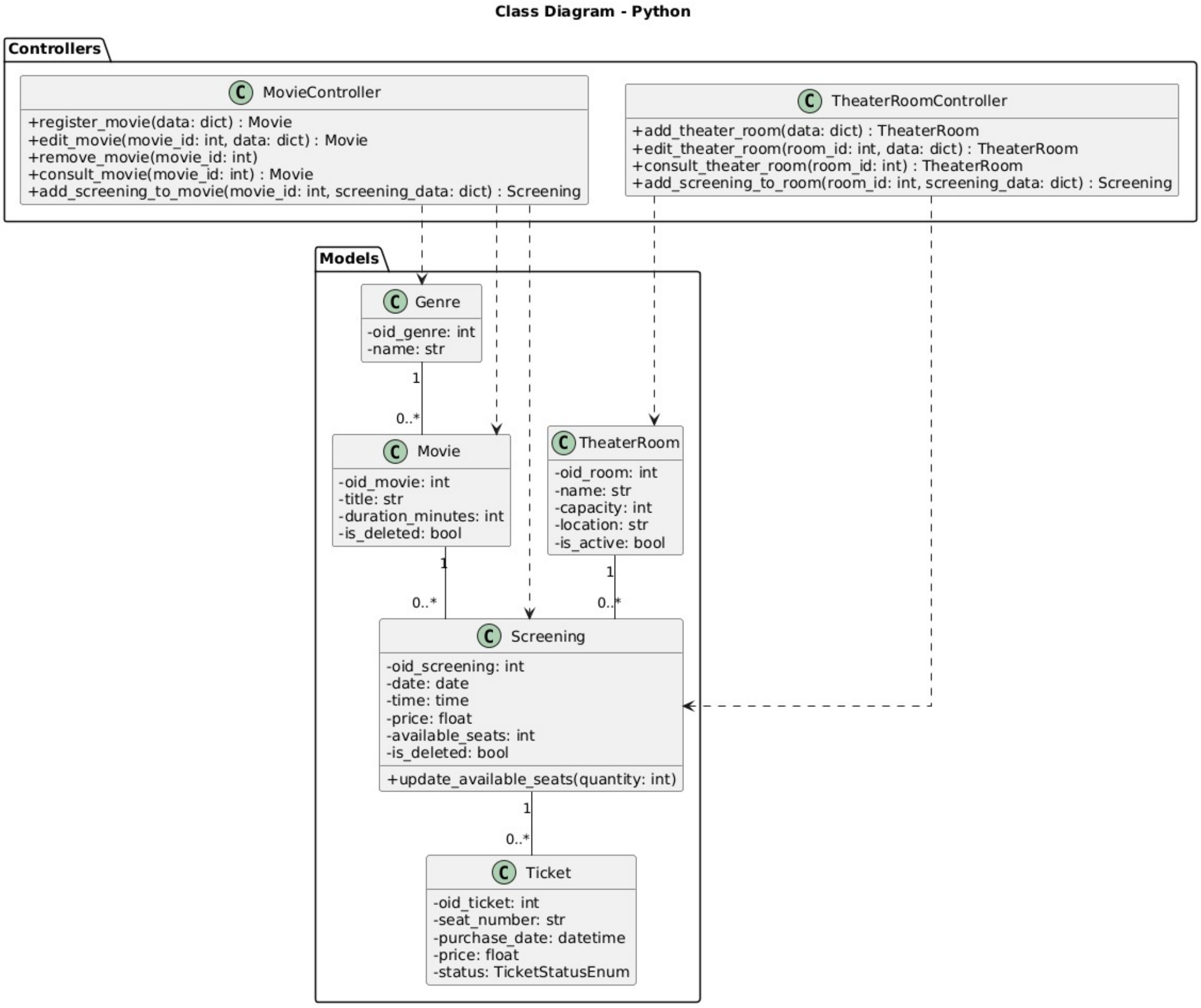

Class Diagram - Python
Controllers
MovieController
+register_movie(data: dict) : Movie
+edit_movie(movie_id: int, data: dict) : Movie
+remove_movie(movie_id: int)
+consult_movie(movie_id: int) : Movie
+add_screening_to_movie(movie_id: int, screening_data: dict) : Screening
TheaterRoomController
+add_theater_room(data: dict) : TheaterRoom
+edit_theater_room(room_id: int, data: dict) : TheaterRoom
+consult_theater_room(room_id: int) : TheaterRoom
+add_screening_to_room(room_id: int, screening_data: dict) : Screening
Models
Genre
-oid_genre: int
-name: str
1
0..*
Movie
-oid_movie: int
-title: str
-duration_minutes: int
-is_deleted: bool
TheaterRoom
-oid_room: int
-name: str
-capacity: int
-location: str
-is_active: bool
1
0..*
1
0..*
Screening
-oid_screening: int
-date: date
-time: time
-price: float
-available_seats: int
-is_deleted: bool
+update_available_seats(quantity: int)
1
0..*
Ticket
-oid_ticket: int
-seat_number: str
-purchase_date: datetime
-price: float
-status: TicketStatusEnum

Figure 1: Class Diagram for Python App

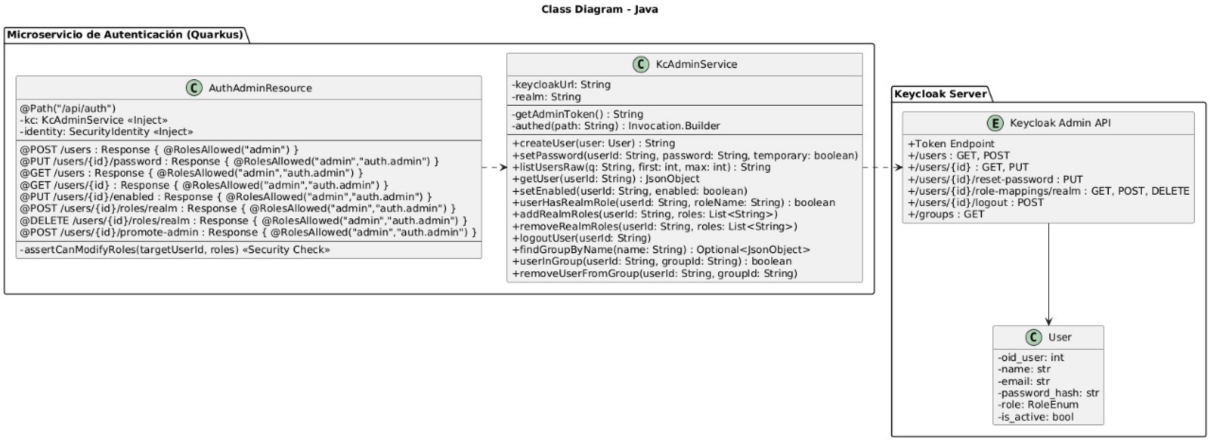

Class Diagram - Java
Microservicio de Autenticación (Quarkus)
C AuthAdminResource
@Path("/api/auth")
-kc: KcAdminService «Inject»
-identity: SecurityIdentity «Inject»
@POST /users : Response { @RolesAllowed("admin") }
@PUT /users/{id}/password : Response { @RolesAllowed("admin","auth.admin") }
@GET /users : Response { @RolesAllowed("admin","auth.admin") }
@GET /users/{id} : Response { @RolesAllowed("admin","auth.admin") }
@PUT /users/{id}/enabled : Response { @RolesAllowed("admin","auth.admin") }
@POST /users/{id}/roles/realm : Response { @RolesAllowed("admin","auth.admin") }
@DELETE /users/{id}/roles/realm : Response { @RolesAllowed("admin","auth.admin") }
@POST /users/{id}/promote-admin : Response { @RolesAllowed("admin","auth.admin") }
-assertCanModifyRoles(targetUserId, roles) «Security Check»
C KcAdminService
-keycloakUrl: String
-realm: String
-getAdminToken() : String
-authed(path: String) : Invocation.Builder
+createUser(user: User) : String
+setPassword(userId: String, password: String, temporary: boolean)
+listUsersRaw(q: String, first: int, max: int) : String
+getUser(userId: String) : JsonObject
+setEnabled(userId: String, enabled: boolean)
+userHasRealmRole(userId: String, roleName: String) : boolean
+addRealmRoles(userId: String, roles: List<String>)
+removeRealmRoles(userId: String, roles: List<String>)
+logoutUser(userId: String)
+findGroupByName(name: String) : Optional<JsonObject>
+userInGroup(userId: String, groupId: String) : boolean
+removeUserFromGroup(userId: String, groupId: String)
Keycloak Server
E Keycloak Admin API
+Token Endpoint
+/users : GET, POST
+/users/{id} : GET, PUT
+/users/{id}/reset-password : PUT
+/users/{id}/role-mappings/realm : GET, POST, DELETE
+/users/{id}/logout : POST
+/groups : GET
C User
-oid_user: int
-name: str
-email: str
-password_hash: str
-role: RoleEnum
-is_active: bool

Figure 2: Class Diagram for Java Authentication

Architecture Diagram

The following diagram illustrates the reference architecture adopted for the Cinema Management System. The system follows a microservices-based approach, separating authentication and cinema-related business logic into independent services to ensure modularity, scalability, and ease of maintenance.

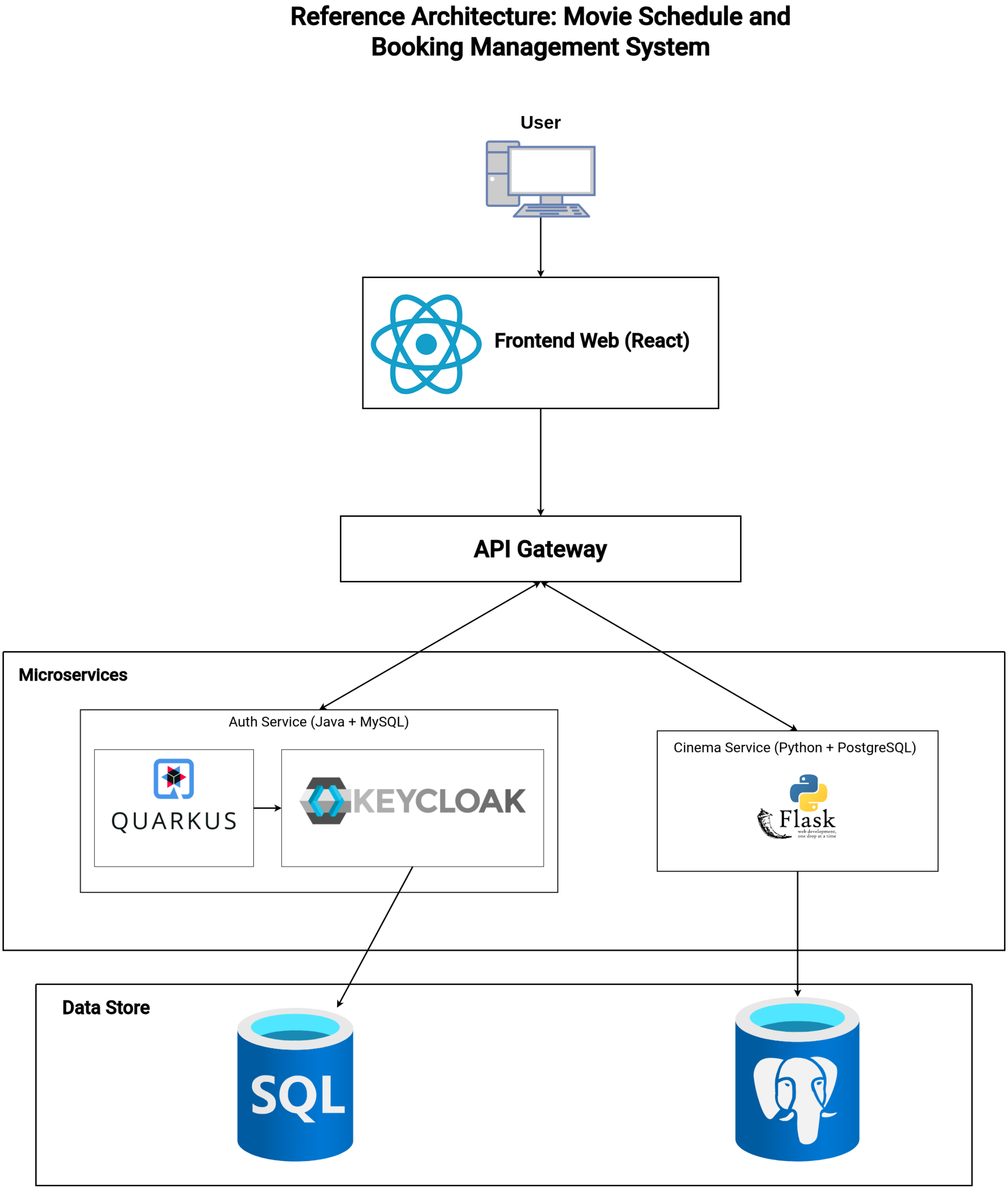

Reference Architecture: Movie Schedule and Booking Management System
User
Frontend Web (React)
API Gateway
Microservices
Auth Service (Java + MySQL)
QUARKUS
KEYCLOAK
Cinema Service (Python + PostgreSQL)
Flask
web development, one drop at a time
Data Store
SQL

Figure 3: Arquitecture Diagram

As shown in the diagram, users interact directly with the React-based frontend, which communicates with the backend services through an API Gateway. The system is composed of two microservices:

- **Auth Service (Java + Quarkus + Keycloak + MySQL):** This service handles user authentication, authorization, and identity management using Keycloak as the IAM provider, while MySQL stores user-related data.
- **Cinema Service (Python + Flask + PostgreSQL):** This service is responsible for

managing movies, schedules, and bookings. PostgreSQL serves as the primary data store for cinema-related business information.

Deployment Diagram

The deployment diagram in Figure 4 illustrates the system fully deployed on **Microsoft Azure** using **Container Apps** for all services. The solution employs a **GitHub Actions** pipeline that builds and pushes Docker images to the **Azure Container Registry (ACR)**, which are then deployed automatically. The **React.js frontend** runs in an **Azure Web App** and connects via **HTTPS** to an **API Gateway** that routes requests to two microservices: **Auth** (Java + Quarkus + Keycloak) and **Cinema** (Python + Flask), linked to managed **MySQL** and **PostgreSQL** databases respectively. This container-based deployment ensures scalability, security, and cost efficiency through Azure's consumption-based pricing model.

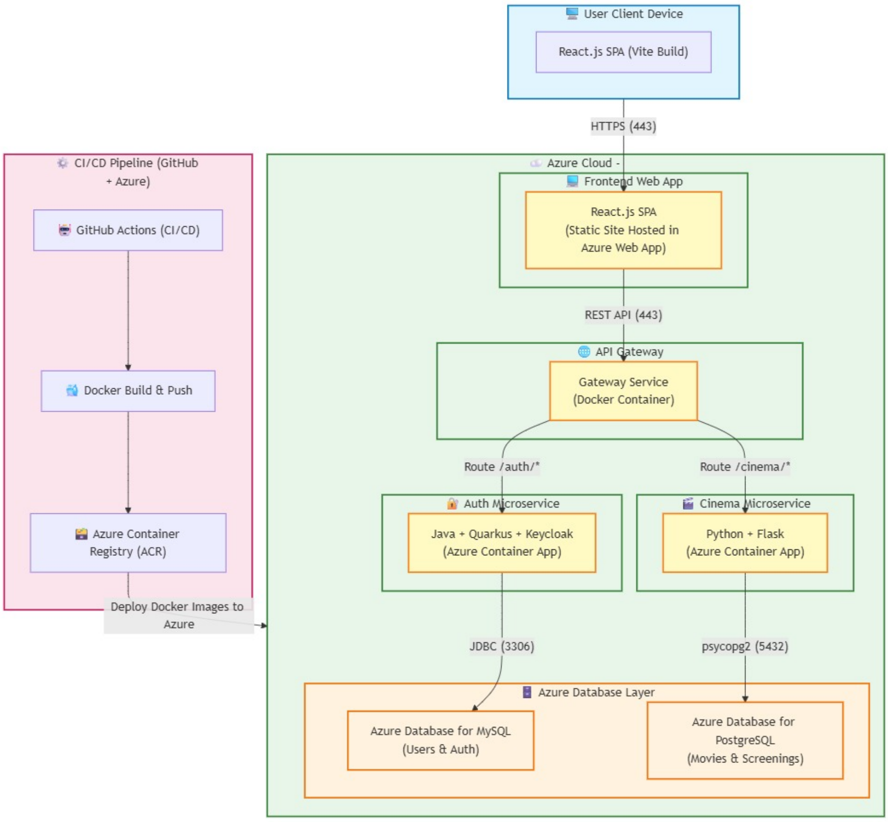

User Client Device
React.js SPA (Vite Build)
HTTPS (443)
CI/CD Pipeline (GitHub + Azure)
GitHub Actions (CI/CD)
Docker Build & Push
Azure Container Registry (ACR)
Deploy Docker Images to Azure
Azure Cloud -
Frontend Web App
React.js SPA (Static Site Hosted in Azure Web App)
REST API (443)
API Gateway
Gateway Service (Docker Container)
Route /auth/*
Route /cinema/*
Auth Microservice
Java + Quarkus + Keycloak (Azure Container App)
Cinema Microservice
Python + Flask (Azure Container App)
JDBC (3306)
psycopg2 (5432)
Azure Database Layer
Azure Database for MySQL (Users & Auth)
Azure Database for PostgreSQL (Movies & Screenings)

Figure 4: Deployment Diagram

Business Process Documentation

The following diagram represents the business process flow from the perspective of the customer interacting with the Cinema Management System. It outlines how an already authenticated user who has the "customer" role initiates a movie search, requests schedule information, and receives confirmation from the system based on availability and internal coordination with the cinema and content providers.

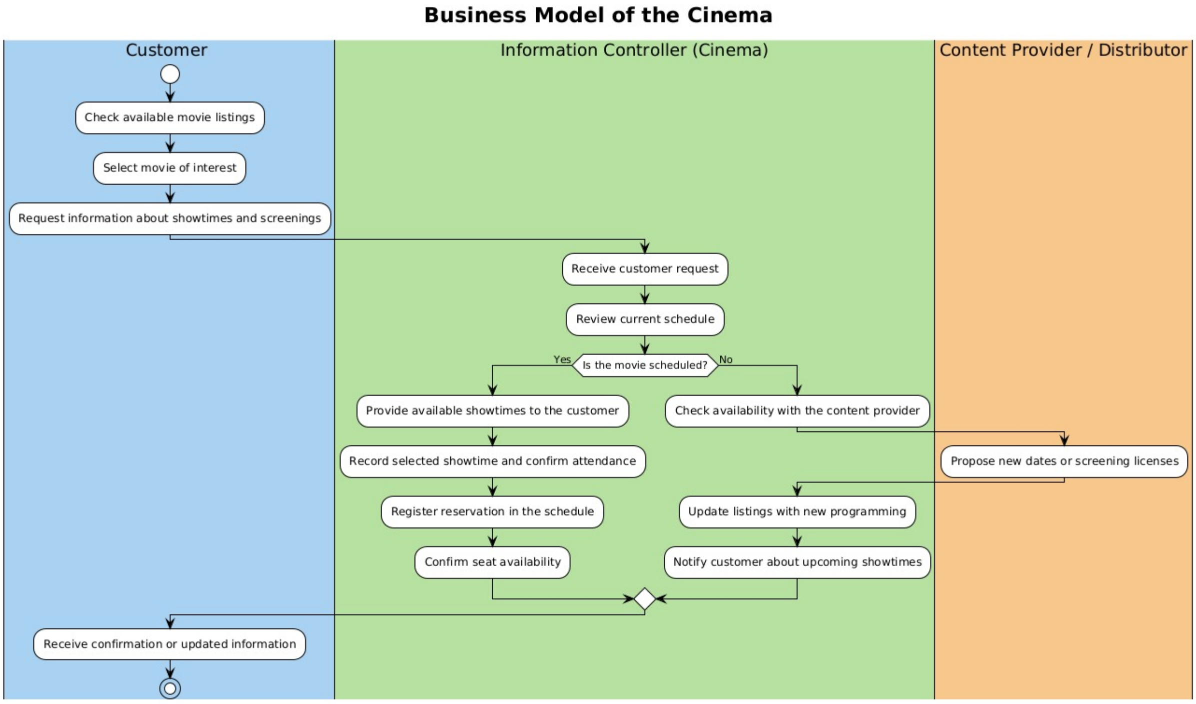

Business Model of the Cinema
Customer
Information Controller (Cinema)
Content Provider / Distributor
Check available movie listings
Select movie of interest
Request information about showtimes and screenings
Receive customer request
Review current schedule
Yes
Is the movie scheduled?
No
Provide available showtimes to the customer
Check availability with the content provider
Record selected showtime and confirm attendance
Propose new dates or screening licenses
Register reservation in the schedule
Update listings with new programming
Confirm seat availability
Notify customer about upcoming showtimes
Receive confirmation or updated information

Figure 5: Business Model Process for Customer

As illustrated in the process flow, the customer begins by browsing the available movie listings and selecting a title of interest. Once a movie is chosen, the user requests information regarding upcoming showtimes. The cinema system then evaluates the current schedule and, if the movie is already programmed, returns the available time slots to the customer.

If the movie is not yet scheduled, the system communicates with the content provider or distributor to check availability and potentially request new screening slots. Based on the response, the system updates its programming and notifies the customer about available or upcoming showtimes. The user then confirms interest, and the system registers the reservation and seating availability. Finally, the customer receives confirmation or updated information about their request.

Web UI Screenshots

Implemented Web UI (Screenshots)

To demonstrate the current progress of the frontend implementation, this subsection presents screenshots of the features that are already developed and fully functional in the Web UI. These interfaces are connected to the backend and allow real interaction from end users, reflecting the actual progress of the cinema management system.

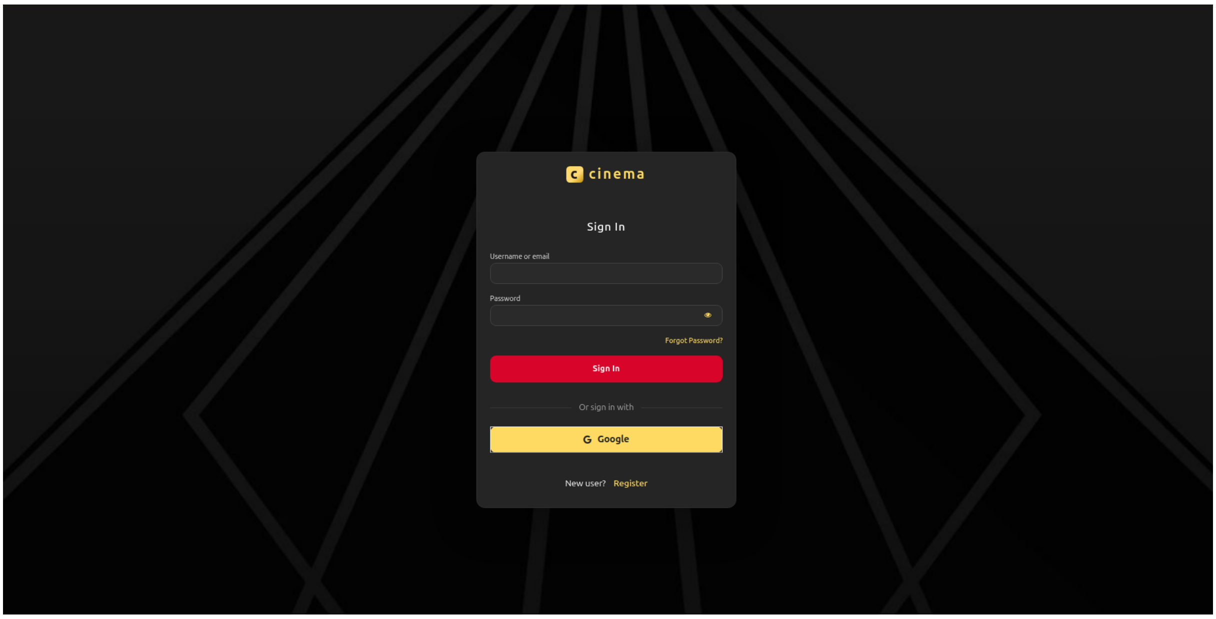


Figure 6: Login

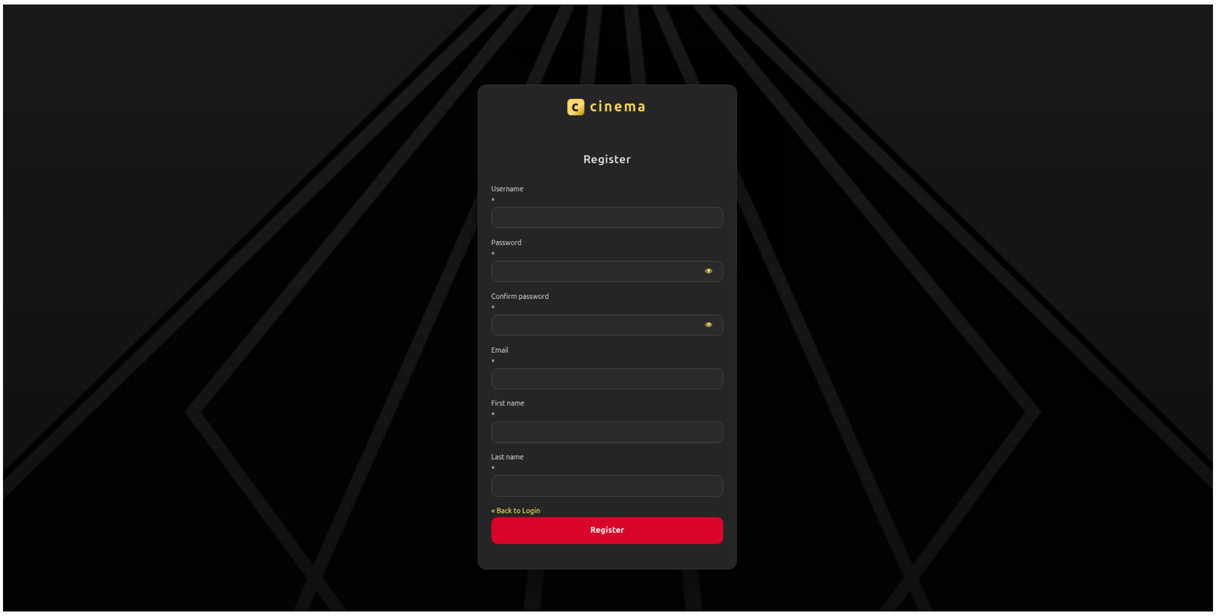
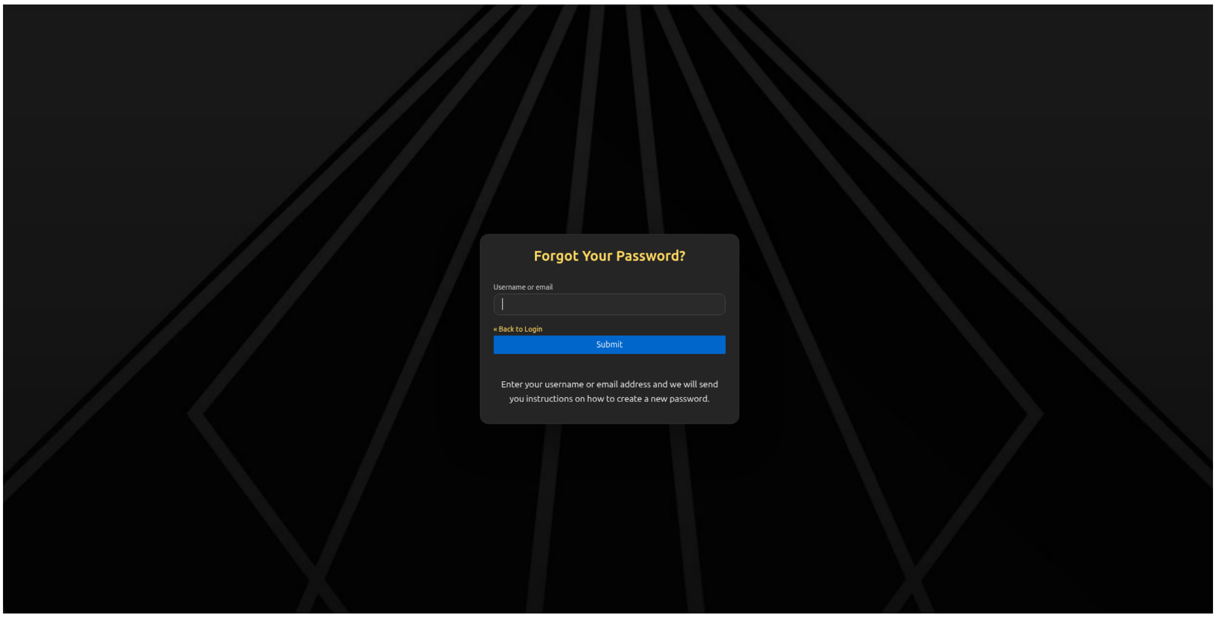


Figure 7: Register



Forgot Your Password?

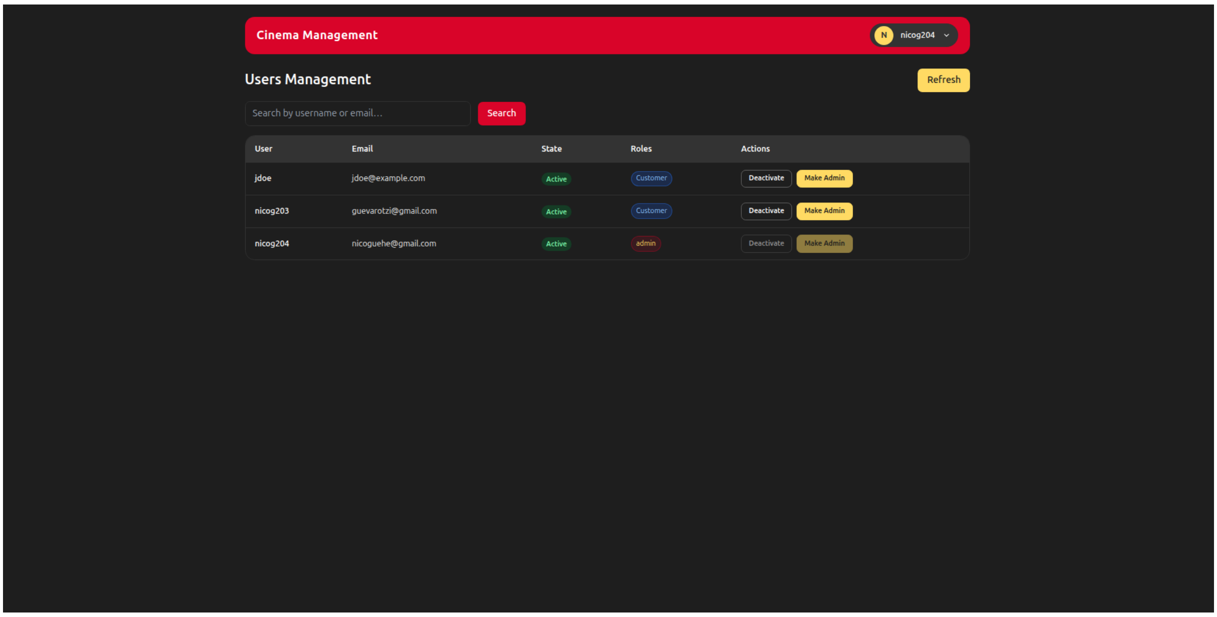
Username or email

« Back to Login

Submit

Enter your username or email address and we will send you instructions on how to create a new password.

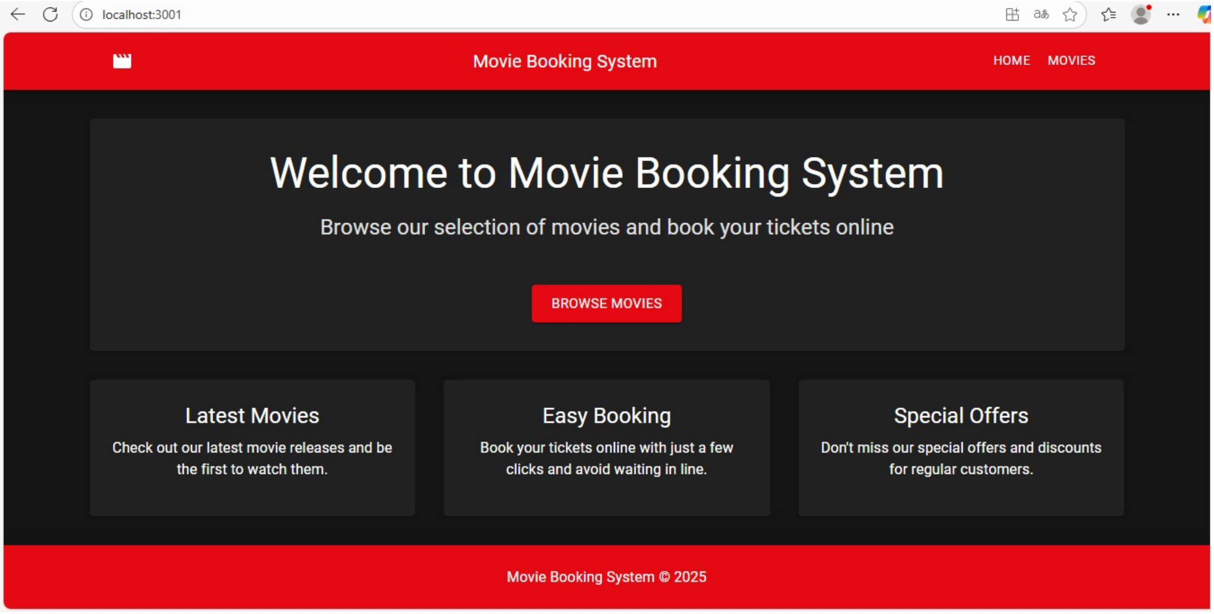
Figure 8: Forgot Password



Users Management

User	Email	State	Roles	Actions
jdoe	jdoe@example.com	Active	Customer	Deactivate Make Admin
nicog203	guevarotzi@gmail.com	Active	Customer	Deactivate Make Admin
nicog204	nicoguehe@gmail.com	Active	admin	Deactivate Make Admin

Figure 9: Users Management


localhost:3001

Movie Booking System

HOME MOVIES

Welcome to Movie Booking System

Browse our selection of movies and book your tickets online

BROWSE MOVIES

Latest Movies

Check out our latest movie releases and be the first to watch them.

Easy Booking

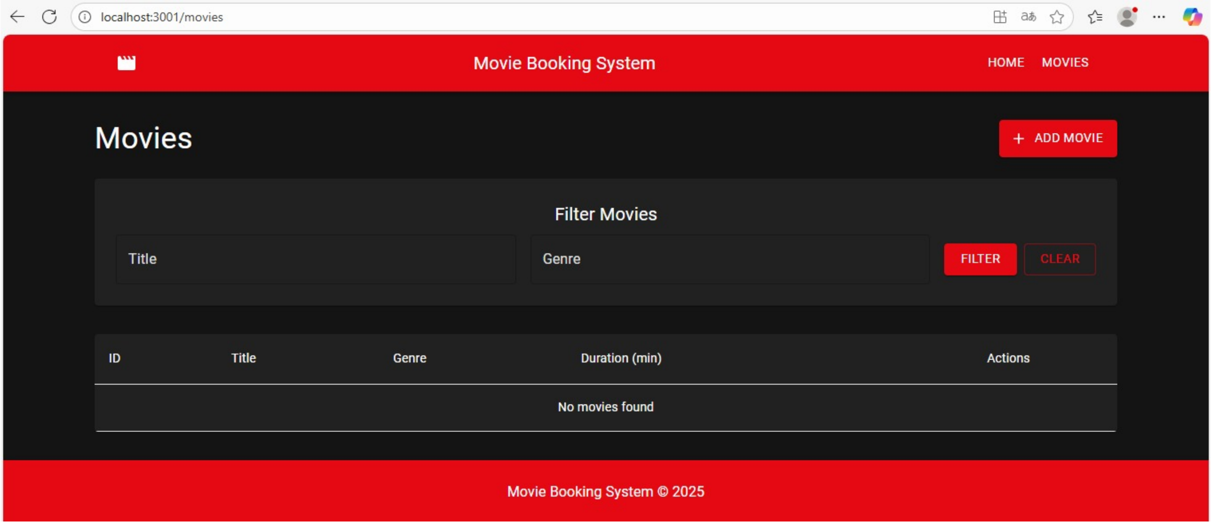
Book your tickets online with just a few clicks and avoid waiting in line.

Special Offers

Don't miss our special offers and discounts for regular customers.

Movie Booking System © 2025

Figure 10: Movie Catalog


localhost:3001/movies

Movie Booking System

HOME MOVIES

Movies

+ ADD MOVIE

Filter Movies

Title

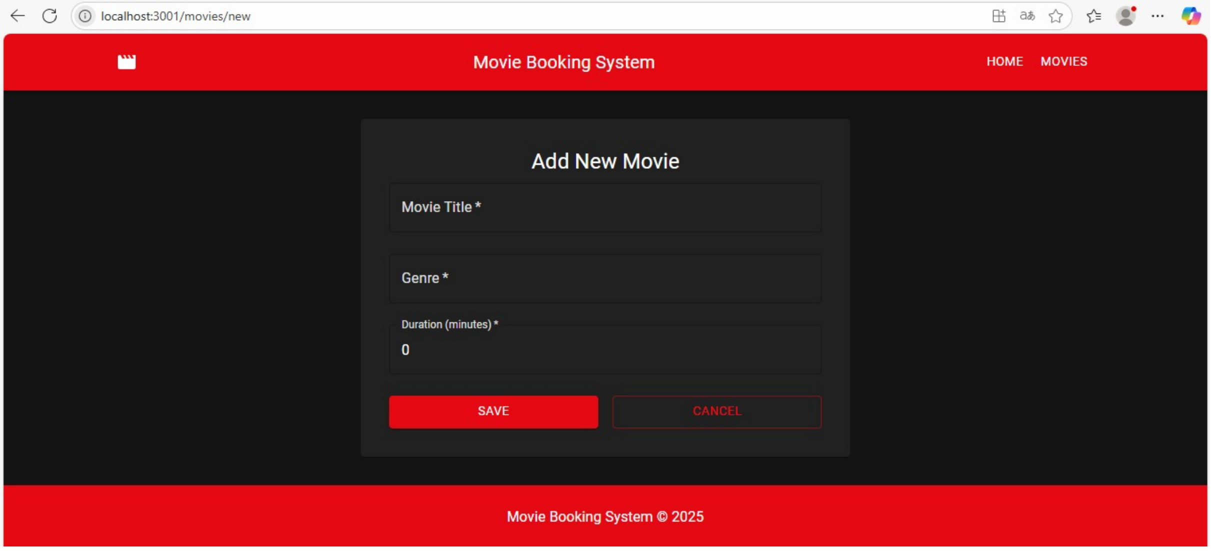
Genre

FILTER CLEAR

ID	Title	Genre	Duration (min)	Actions
		No movies found		

Movie Booking System © 2025

Figure 11: Movie Management



localhost:3001/movies/new

Movie Booking System

Add New Movie

Movie Title *

Genre *

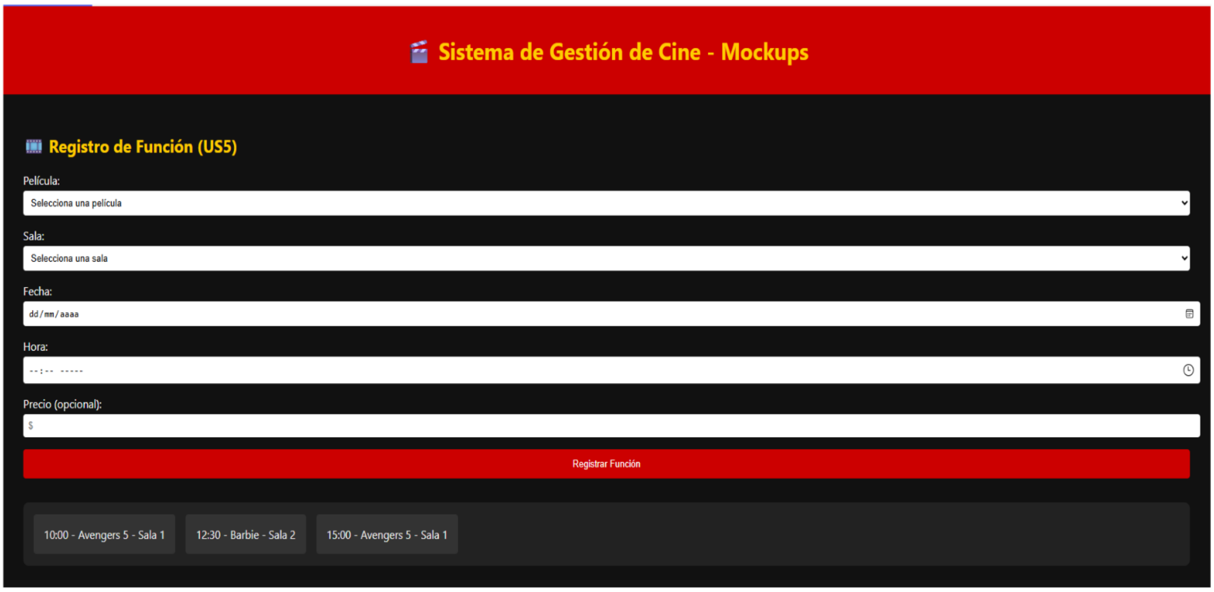
Duration (minutes) *
0

Movie Booking System © 2025

Figure 12: Add Movie

Pending UI (Mockups)

The following mockups represent the user interface components that are still under development. They serve as visual guidelines for the remaining features and illustrate the intended design and interaction flow before their implementation in the functional Web UI.



🎬 Sistema de Gestión de Cine - Mockups

Registro de Función (US5)

Película:
Selecciona una película

Sala:
Selecciona una sala

Fecha:
dd/mm/aaaa

Hora:
--:-- -----

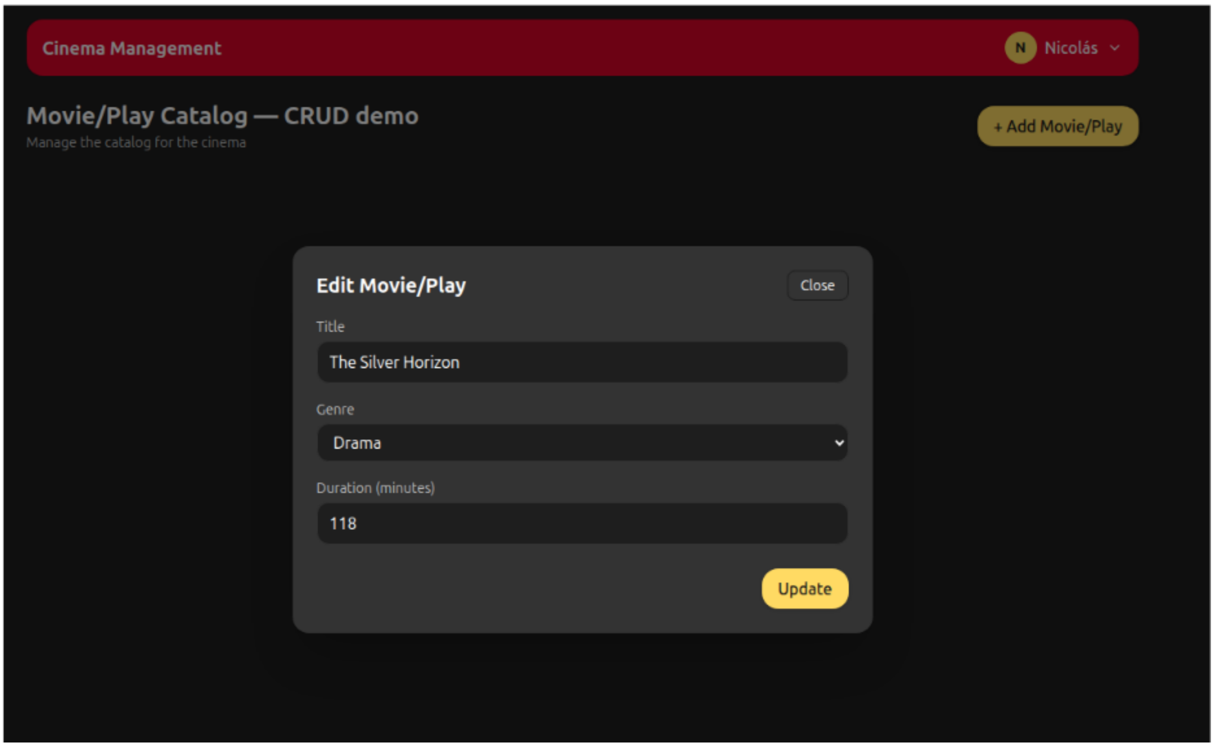
Precio (opcional):
$

10:00 - Avengers 5 - Sala 1

12:30 - Barbie - Sala 2

15:00 - Avengers 5 - Sala 1

Figure 13: Add Function (Mockup)

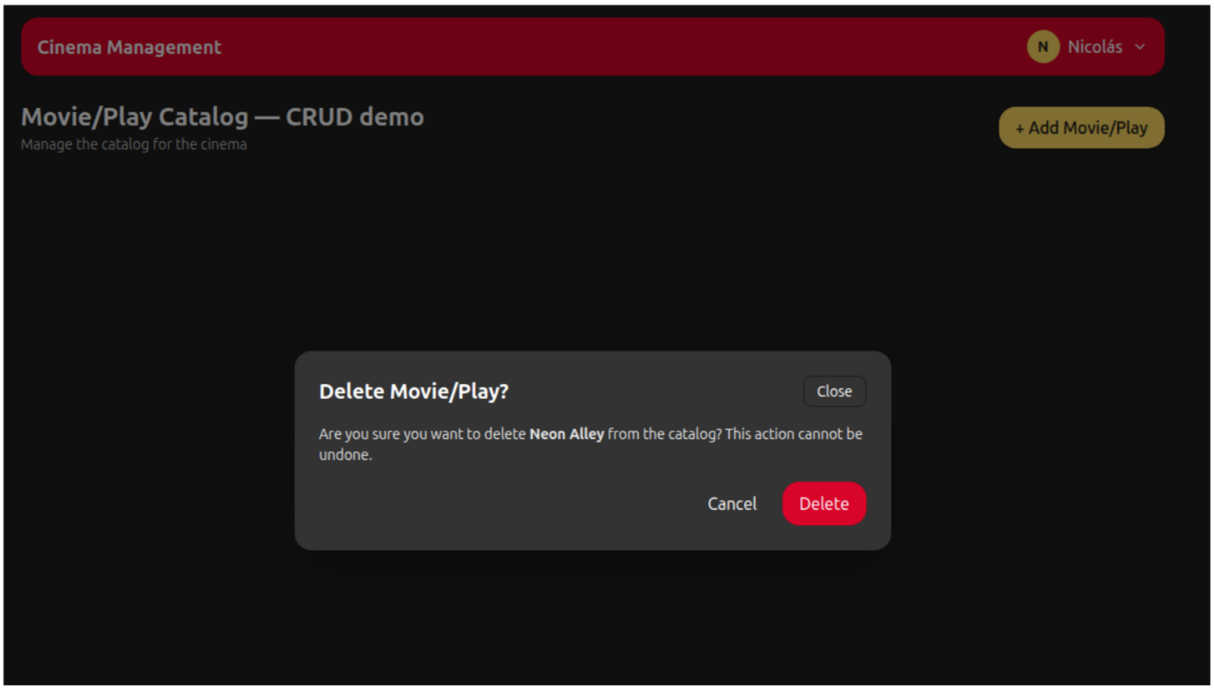


Title: The Silver Horizon

Genre: Drama

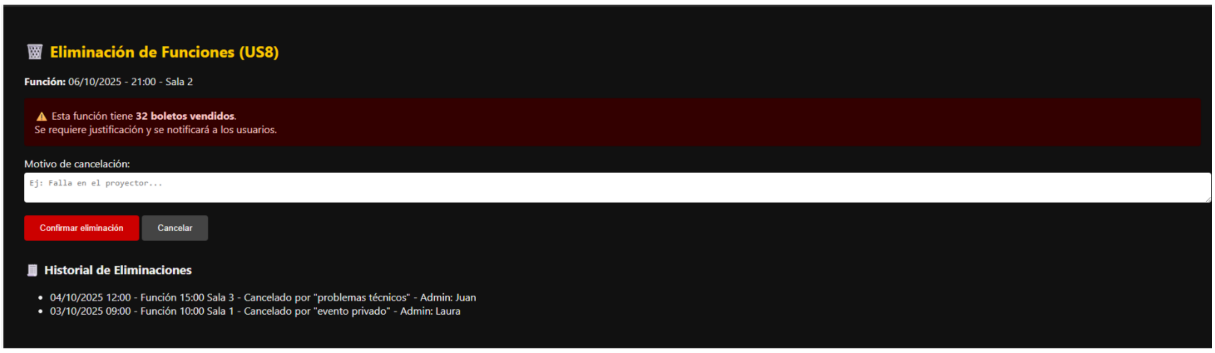
Duration (minutes): 118

Figure 14: Edit Movie (Mockup)



Are you sure you want to delete **Neon Alley** from the catalog? This action cannot be undone.

Figure 15: Delete Movie (Mockup)



Eliminación de Funciones (US8)

Función: 06/10/2025 - 21:00 - Sala 2

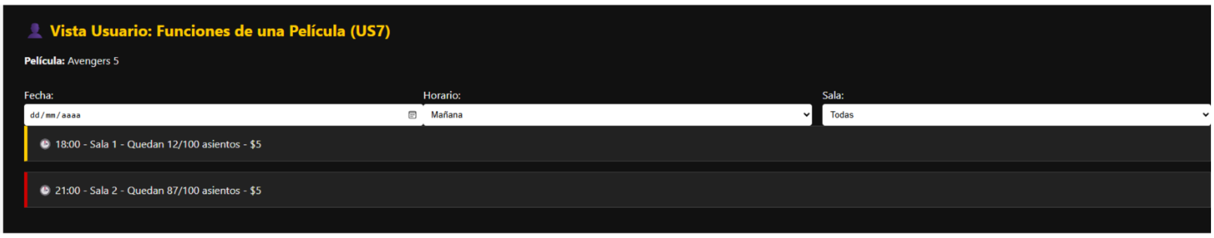
⚠ Esta función tiene **32 boletos vendidos**.
Se requiere justificación y se notificará a los usuarios.

Motivo de cancelación: Ej: Falla en el proyector...

Historial de Eliminaciones

- 04/10/2025 12:00 - Función 15:00 Sala 3 - Cancelado por "problemas técnicos" - Admin: Juan
- 03/10/2025 09:00 - Función 10:00 Sala 1 - Cancelado por "evento privado" - Admin: Laura

Figure 16: Delete Function (Mockup)



Vista Usuario: Funciones de una Película (US7)

Película: Avengers 5

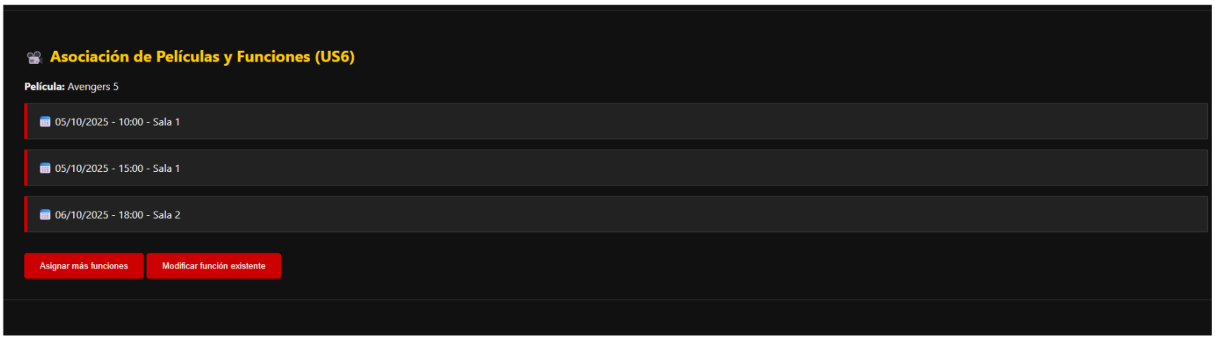
Fecha: dd/mm/aaaa

Horario: Mañana

Sala: Todas

- 18:00 - Sala 1 - Quedan 12/100 asientos - $5
- 21:00 - Sala 2 - Quedan 87/100 asientos - $5

Figure 17: Functions of a Movie (Mockup)



Asociación de Películas y Funciones (US6)

Película: Avengers 5

- 05/10/2025 - 10:00 - Sala 1
- 05/10/2025 - 15:00 - Sala 1
- 06/10/2025 - 18:00 - Sala 2

Figure 18: Movie–Function Association (Mockup)