Report Two: Fort Hays State Movie Theaters

CSCI 441

Professor: Hieu Vu
Members: Jakob Schaefer, Derek Litke, Andrew Carter, Hanyong Yoon, David Sowles, Ryan Smith

Github - https://github.com/FHSMT/FHSMT_Project
YouTrack - https://fhsmt.youtrack.cloud/dashboard?id=169-0

Member Contribution

Jakob Schaefer – Traceability Matrix, formatting, contribution list & graph, table of contents, concept definition

Hanyong Yoon – Contribution list / table of contents, concept definition, association definition, attribute definition, and traceability matrix

Derek Litke – Systems Operations Contracts, Class Diagram

Andrew Carter – Class Diagram, association definition diagrams, association table

David Sowles – Class Diagram, gantt diagram

Ryan Smith - Data Types and Operation Signatures, Database Schema, Design of Tests

Responsibility Matrix

Sub-Project / Task	Jakob	Hanyong	Derek	Andrew	David	Ryan
Cover Page & Contributions	20%	20%	20%	20%	20%	0%
Work Assignment	16%	16%	16%	16%	16%	16%
Concept Definitions	30%	30%	10%	10%	10%	10%
Association Definitions	10%	40%	10%	20%	10%	10%
Attribute Definitions	15%	40%	15%	10%	10%	10%
Traceability Matrix	15%	40%	15%	10%	10%	10%
System Operation Contracts	10%	15%	40%	15%	10%	10%
Data Model and Persistent Data	15%	10%	15%	10%	10%	40%
Mathematical Model	15%	10%	15%	10%	10%	40%
Interaction Diagrams	10%	15%	15%	40%	10%	10%
Class Diagram	10%	10%	30%	10%	30%	10%
Data Types and Operation Signatures	15%	10%	15%	10%	10%	40%
Traceability Matrix	30%	30%	10%	10%	10%	10%
Project Management	15%	10%	15%	10%	40%	10%
Project Management	15%	10%	15%	10%	40%	10%
References	16%	16%	16%	16%	16%	16%

Table of Contents

Concept Definitions

Account – Stores customer credentials (email, password) and profile details

Authentication – validates login information for secure access

Catalog UI – Interface that displays movie listings

Search Service – allows searching for movies

Movie – official record of film details

Scheduling Service – handles the creation and management of showtimes

Showtime – represents date/time and auditorium for a movie

Seating UI – displays seating charts with availability

Seat Map Template – defines the layout of seats in an auditorium

Seat Inventory – tracks available, sold, or blocked seats

Seat Hold Manager – holds seats temporarily and releases them if not purchased

Pricing Engine – calculates ticket prices, fees, and charges

Order Service – creates orders based on selected seats

Order – stores order details including items, fees, taxes, and status

Ticket Issuer – issues tickets once payment is confirmed

Ticket – stores finalized ticket details

Notification Service – sends confirmation/cancellation emails

Cancellation Service – handles order cancellations

Refund – records and processes returned payments

Reporting Service – generates sales and booking reports

Association Definitions

Association Definitions (Purchasing Context)

Concept A	Concept B	Multiplicity / Role
Customer	Account	1 ⟷ 1
Customer	Order	1 ⟶ 0..* (places)
Order	Payment	1 ⟷ 1 (authorized-by)
Payment	Refund	0..1 (refund)
Order	Ticket	1 ⟶ 1..* (issues)
Ticket	Showtime	1 ⟶ 1 (for)
Ticket	Seat	1 ⟷ 1 (assigned)
Showtime	Movie	* ⟶ 1 (listed-for)
Showtime	Auditorium	* ⟶ 1 (scheduled-in)
Showtime	SeatInventory	1 ⟷ 1 (has layout)

Customer <-> Account

-Customer is linked to a single account once they register. The account stores login information, profile, and purchase history.

Customer <-> Order

-A customer can place as many orders as they want, but each order belongs to the single customer that has placed the order.

Order <-> Ticket

-An order can contain one or more tickets, and each ticket is always tied to a single order.

Ticket <-> Showtime

-Each ticket gives access to a single showtime. Showtime may have multiple tickets tied to it.

Ticket <-> Seat

- Ticket can reserve one seat per showtime. A seat may be associated with many tickets, but it can only be reserved once per showtime.

Movie <-> Showtime

-Movie can have multiple showtimes, showtimes show one movie each.

Theater <-> Auditorium

-Theater contains multiple auditoriums. Each auditorium belongs to one theater.

Showtime <-> Auditorium

-A showtime is scheduled in one auditorium. An auditorium can host many showtimes over time.

Showtime <-> Seat Inventory

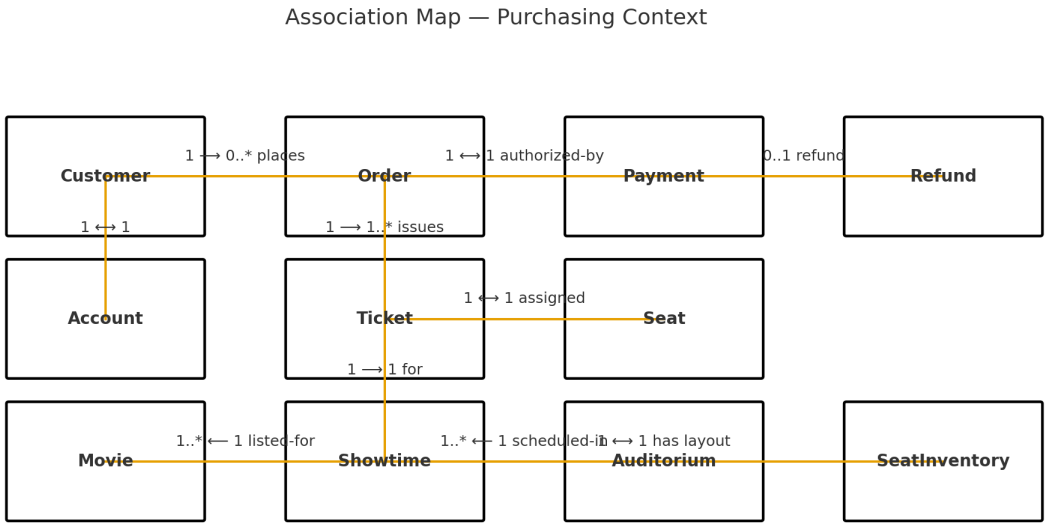
-Each showtime generates seat inventory that keeps which seats are available.

Payment <-> Order

-Payment is tied to one order. An order must have one payment.

Refund <-> Payment

-Refund is linked to original payment. A payment may or not be associated with refunds depending on whether the customer requested it or not.



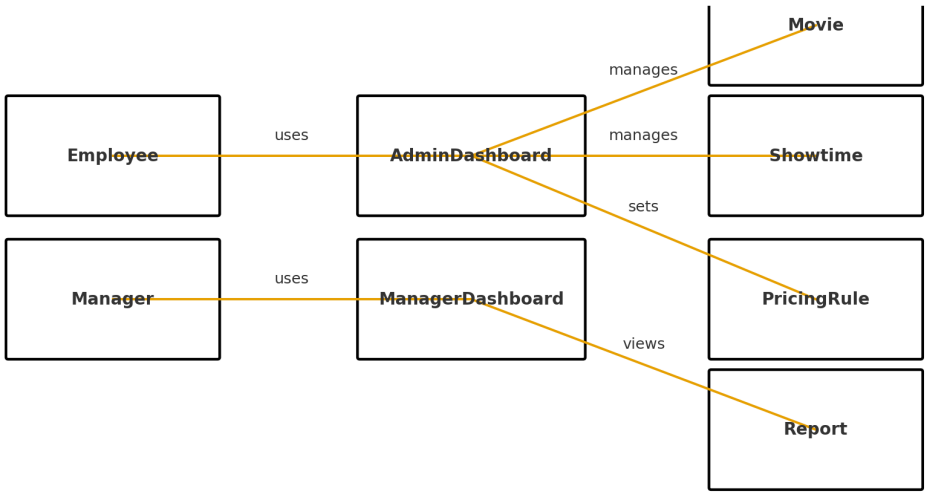
Employee <-> Admin Dashboard

-An employee accesses the admin dashboard to update theater information.

Manager <-> Manager Dashboard.

-A manager uses the manager dashboard to monitor sales and payroll summaries.

Association Map — Admin/Backoffice Context



Attribute Definitions

Concept	Attribute	Description
Customer Account	accountEmail	Email address associated with the customer's account. Used for login and notifications.
	accountPassword	Encrypted password for secure login.
	accountHistory	Stores all past ticket purchases linked to the customer.
	accountFavorites	Optional list of saved or preferred movies.
Movie	movieTitle	Official title of the movie.
	movieRating	Rating shown to the customer.
	movieRuntime	Length of the movie.
	moviePoster	Promo image of the movie shown in the listings page.
Showtime	showtimeID	Identifier for the showtime.
	startTime	Date and time the movie starts.
	auditoriumAssigned	Auditorium where the movie showtime starts.
	format	Format the movie is in.
Ticket	ticketID	Identifier for the ticket.
	ticketCode	Code for entry validation.
	seatAssigned	Seats reserved by the ticket.
	ticketStatus	Status of the ticket; canceled, refunded, unauthorized.
Payment	paymentID	Identifier for payment transaction.
	paymentMethod	Method of payment used.
	paymentAmount	Total amount charged.
	paymentStatus	Transaction status; authorized, refunded, unauthorized.

Report	reportID	Identifier for each report.
	reportType	Shows report; sales, bookings, performance.
	dateRange	Time the report covers.
	reportContent	Content generated for the report.
Interface	currentInterface	Shows proper interface depending on who is logged in.
	browseMovies	Allows users to browse movie listings page.
	searchMovies	Allows users to search using keywords related to the movie.
	seatSelectionChart	Interactive seating map showing available, sold, and accessible seats.
Controller	movieList	Manages and shows the set of movies available in the system.
	showtimeSchedule	Display and update the showtime of each movie.
	ticketPurchase	Coordinates purchase flow for each movie.
	ticketCancellation	Allows customers to cancel their purchase and free up the seat.
	adminDashboard	Provides employees access to movies, showtime, and pricing management functions.
	managerDashboard	Provides managers access to high-level sales and payroll summaries.

Traceability Matrix

Legend: D = Doing, K = Knowing

Responsibility	Type	Concept
R1: Create a customer account using email and password.	D	Account
R2: Store customer email, password, and profile.	K	Account
R3: Authenticate customer log-in information.	D	Authentication
R4: Show movie listings with title, rating, and showtimes.	D	Catalog UI
R5: Search movies by titles and keywords.	D	Search service
R6: Store and maintain official details of movies.	K	Movie
R7: Schedule showtimes in an auditorium.	D	Scheduling Service
R8: Store showtime information like date/time and auditorium.	K	Showtime
R9: Show interactive seating charts with availability.	D	Seating UI
R10: Define auditorium seat layout.	K	Seat map Template
R11: Track what seats are available, sold, and blocked.	K	Seat Inventory
R12: Hold selected seats with timeout.	D	Seat Hold Manager
R13: Release expires holds back to being available.	D	Seat Hold Manager
R14: Calculate Ticket prices and other charges.	D	Pricing Engine
R15: Create order about selected seats.	D	Order Service
R16: Store order items, fees, taxes, and status.	K	Order
R17: Issue tickets after payment.	D	Ticket Issuer
R18: Store ticket details.	K	Ticket
R19: Send purchase and cancellation emails.	D	Notification Service
R20: Cancel purchase and orders	D	Cancellation Service
R21: Record and refund payment	K	Refund
R22: Generate sales and booking report by movie/date/auditorium	D	Reporting Service

System Operation Contracts

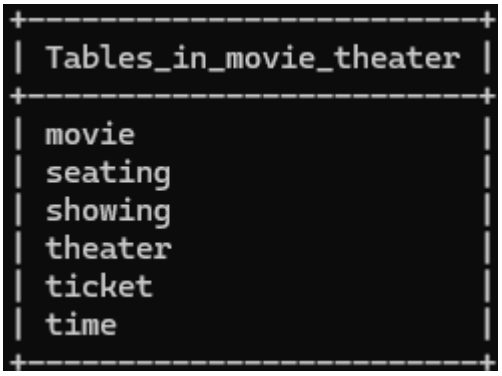
1. Operation: createAccount (email, password)
    a. Preconditions
        i. Email is not already registered. Password meets security rules.
    b. Postconditions
        i. A new account object is created with stored email and encrypted password.
        ii. The customer is linked to the new account
2. Operation: authenticate (email, password)
    a. Precondition
        i. Email exists in the system
    b. Postcondition
        i. If credentials match, system generates authenticated session
        ii. If not, error is returned
3. Operation: searchMovies (keyword)
    a. Preconditons
        i. Catalog contains movies
    b. Postconditions
        i. Returns a list of Movie objects matching keyword by title or metadata
4. Operation: selectSeats (showtime, seatIDs)
    a. Preconditions
        i. Showtime exists and Seat Inventory is available
    b. Postconditions
        i. Selected seats are held by Seat Hold Manager
        ii. Held seats cannot be chosen by other customers until released or confirmed
5. Operation: placeOrder (customer, showtime, seatIDs, paymentInfo)
    a. Preconditions
        i. Seats are held by the customer for that showtime
    b. Postconditions
        i. An Order object is created with selected seats and pricing
        ii. A Payment object is created and linked to the order
        iii. Upon successful payment, Tickets are issued and tied to the order
        iv. Notifications are sent via email
6. Operation: cancelOrder (orderID)
    a. Preconditions

        i. Order exists and is in cancellable state
    b. Postconditions
        i. Order status updated to “Cancelled”
        ii. A Refund object is created and linked to original payment
        iii. Seats are released back to availability
        iv. Notification sent to customer
7. Operation: generateReport (criteria)
    a. Precondition
        i. At least one order exists in the system
    b. Postcondition
        i. Reporting Service generates summary data (sales by movie, date, auditorium, etc)
        ii. Report is returned for display in the Admin/Manager dashboard

Data Model and Persistent Data Storage

Our system will need to save and store data that will outlive one execution of the system. A relational database has been chosen to store our data.

Tables



movie

Field	Type	Null	Key	Default	Extra
movie_id	bigint unsigned	NO	PRI	NULL	auto_increment
name	varchar(30)	NO		NULL	
rating	varchar(10)	NO		NULL	
length_in_min	int	NO		NULL	
description	text	NO		NULL	

seating

Field	Type	Null	Key	Default	Extra
seating_id	bigint unsigned	NO	PRI	NULL	auto_increment
showing_id	bigint unsigned	NO	UNI	NULL	
a1	tinyint(1)	NO		NULL	
a2	tinyint(1)	NO		NULL	
a3	tinyint(1)	NO		NULL	
a4	tinyint(1)	NO		NULL	
a5	tinyint(1)	NO		NULL	
b1	tinyint(1)	NO		NULL	
b2	tinyint(1)	NO		NULL	
b3	tinyint(1)	NO		NULL	
b4	tinyint(1)	NO		NULL	
b5	tinyint(1)	NO		NULL	
c1	tinyint(1)	NO		NULL	
c2	tinyint(1)	NO		NULL	
c3	tinyint(1)	NO		NULL	
c4	tinyint(1)	NO		NULL	
c5	tinyint(1)	NO		NULL	
d1	tinyint(1)	NO		NULL	
d2	tinyint(1)	NO		NULL	
d3	tinyint(1)	NO		NULL	
d4	tinyint(1)	NO		NULL	
d5	tinyint(1)	NO		NULL	

showing

Field	Type	Null	Key	Default	Extra
showing_id	bigint unsigned	NO	PRI	NULL	auto_increment
theater_id	bigint unsigned	NO	MUL	NULL	
movie_id	bigint unsigned	NO	MUL	NULL	
time_id	bigint unsigned	NO	MUL	NULL	
tickets_sold	int unsigned	NO		NULL	
date	date	NO		NULL	

theater

Field	Type	Null	Key	Default	Extra
theater_id	bigint unsigned	NO	PRI	NULL	auto_increment
number	int	NO	UNI	NULL	
seats	int	NO		NULL	
seat_type	varchar(30)	YES		NULL	
assigned_seat	tinyint(1)	NO		NULL	

ticket

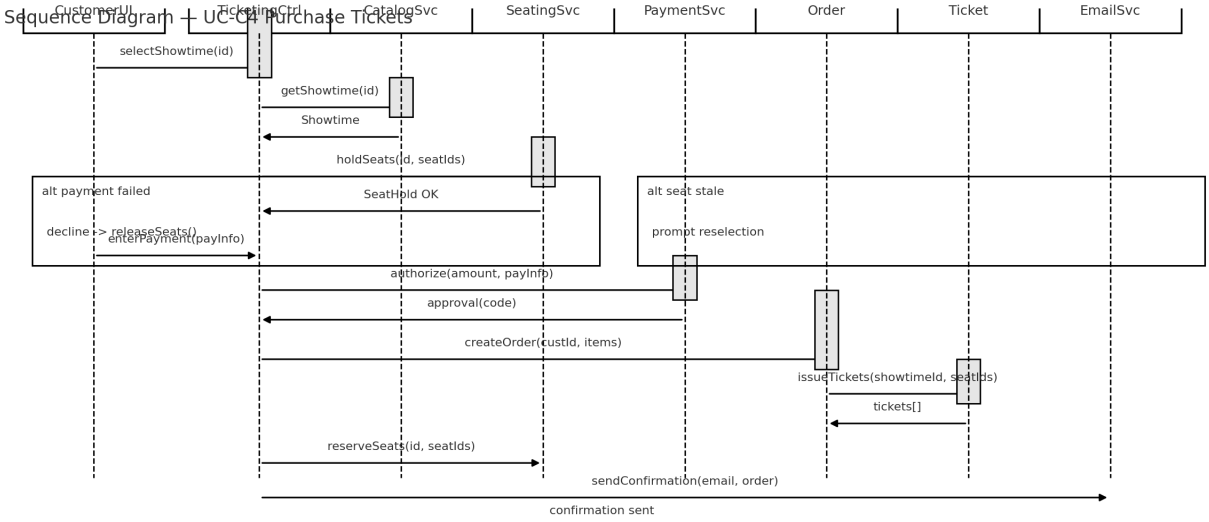
Field	Type	Null	Key	Default	Extra
ticket_id	bigint unsigned	NO	PRI	NULL	auto_increment
child	tinyint(1)	NO		NULL	
showing_id	bigint unsigned	NO	MUL	NULL	

time

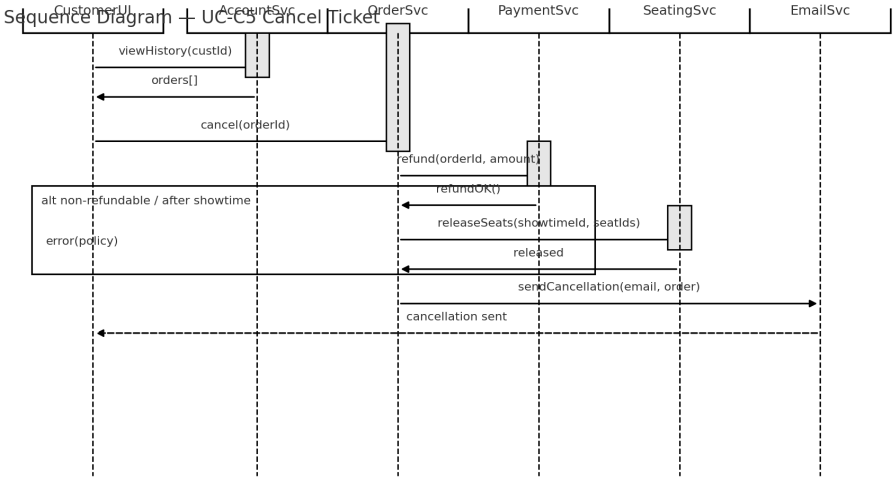
Field	Type	Null	Key	Default	Extra
time_id	bigint unsigned	NO	PRI	NULL	auto_increment
matinee	tinyint(1)	NO		NULL	
showing_time	time	NO		NULL	

Interaction Diagrams

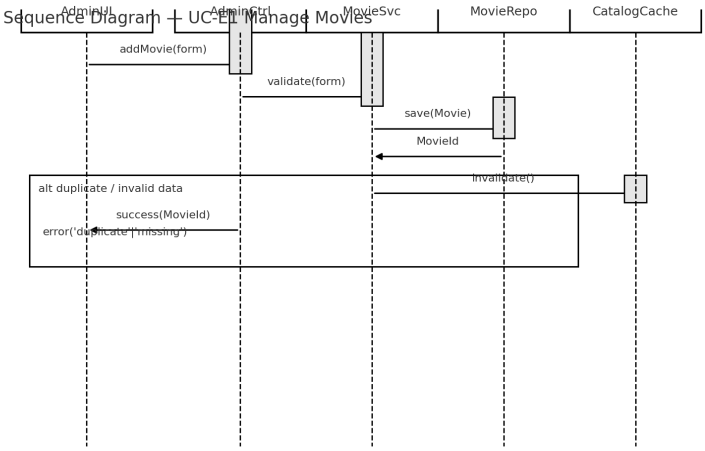
Sequence Diagram — UC-C4 Purchase Tickets



Sequence Diagram — UC-C5 Cancel Ticket



Sequence Diagram — UC-E1 Manage Movies



Class Diagram and Interface Specification

Class Diagram

sequenceDiagram

actor Customer

participant :CustomerUI

participant :OrderService

participant :SeatHoldManager

participant :PricingEngine

participant :PaymentGateway participant :Order

participant :TicketIssuer

participant :NotificationService

Customer->>:CustomerUI: selectSeats(IDs)
:CustomerUI->>:OrderService: holdSeats(IDs)
:OrderService->>:SeatHoldManager: holdSeats(IDs)
note right of :SeatHoldManager: **Information Expert**: Tracks and locks seats.
:SeatHoldManager-->>:OrderService: holdConfirmed()

Customer->>:CustomerUI: placeOrder(paymentInfo)
:CustomerUI->>:OrderService: placeOrder(IDs, paymentInfo)

:OrderService->>:PricingEngine: calculateTotal(IDs)
note right of :PricingEngine: **Low Coupling**: Encapsulates pricing rules.
:PricingEngine-->>:OrderService: totalAmount

:OrderService->>:PaymentGateway: authorize(totalAmount, info)
:PaymentGateway-->>:OrderService: paymentApproved()

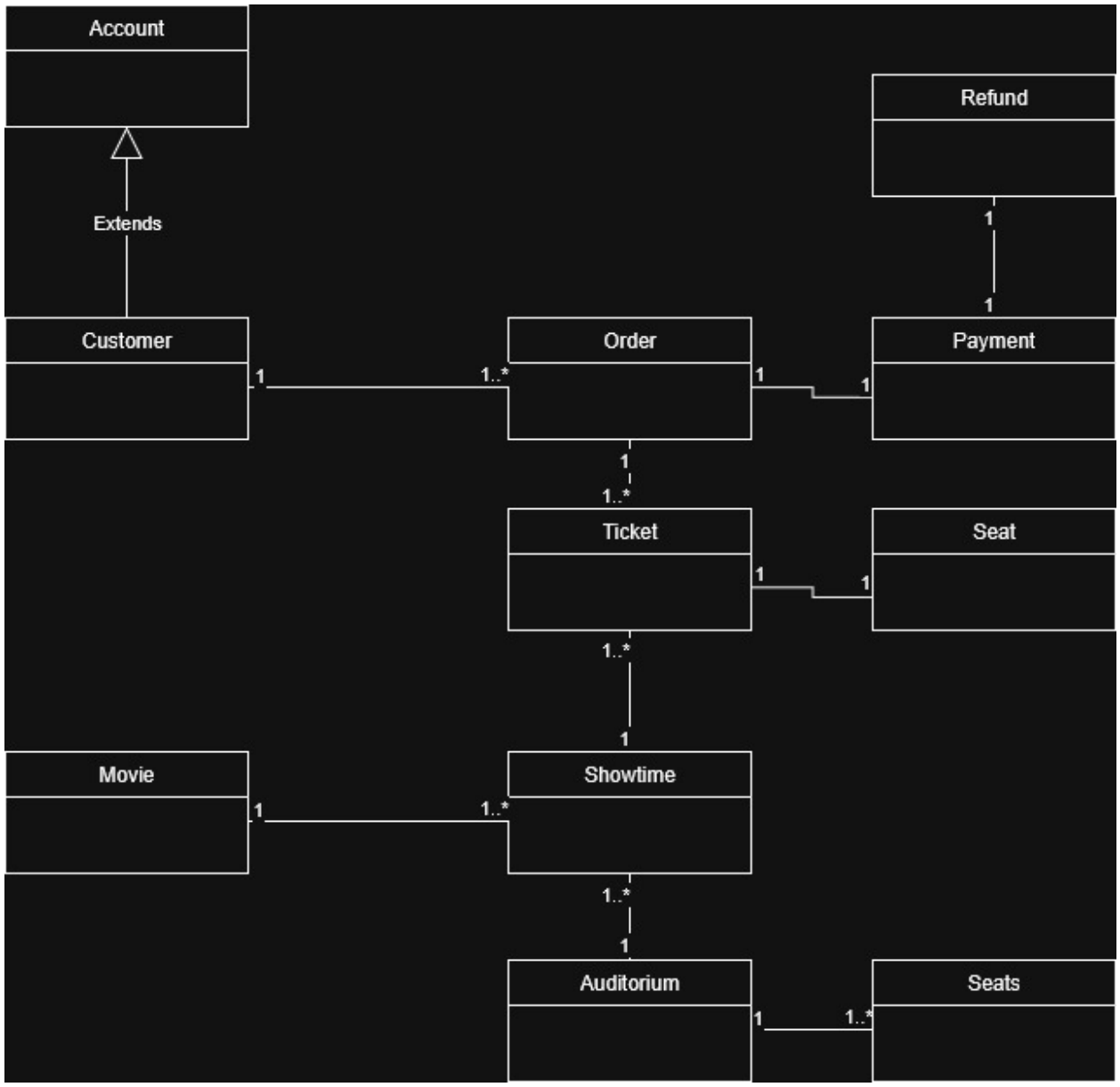
alt Payment Approved
  :OrderService->>:Order: **[Creation]** new(IDs, total)
  note right of :Order: **Creator**: OrderService creates the Order.

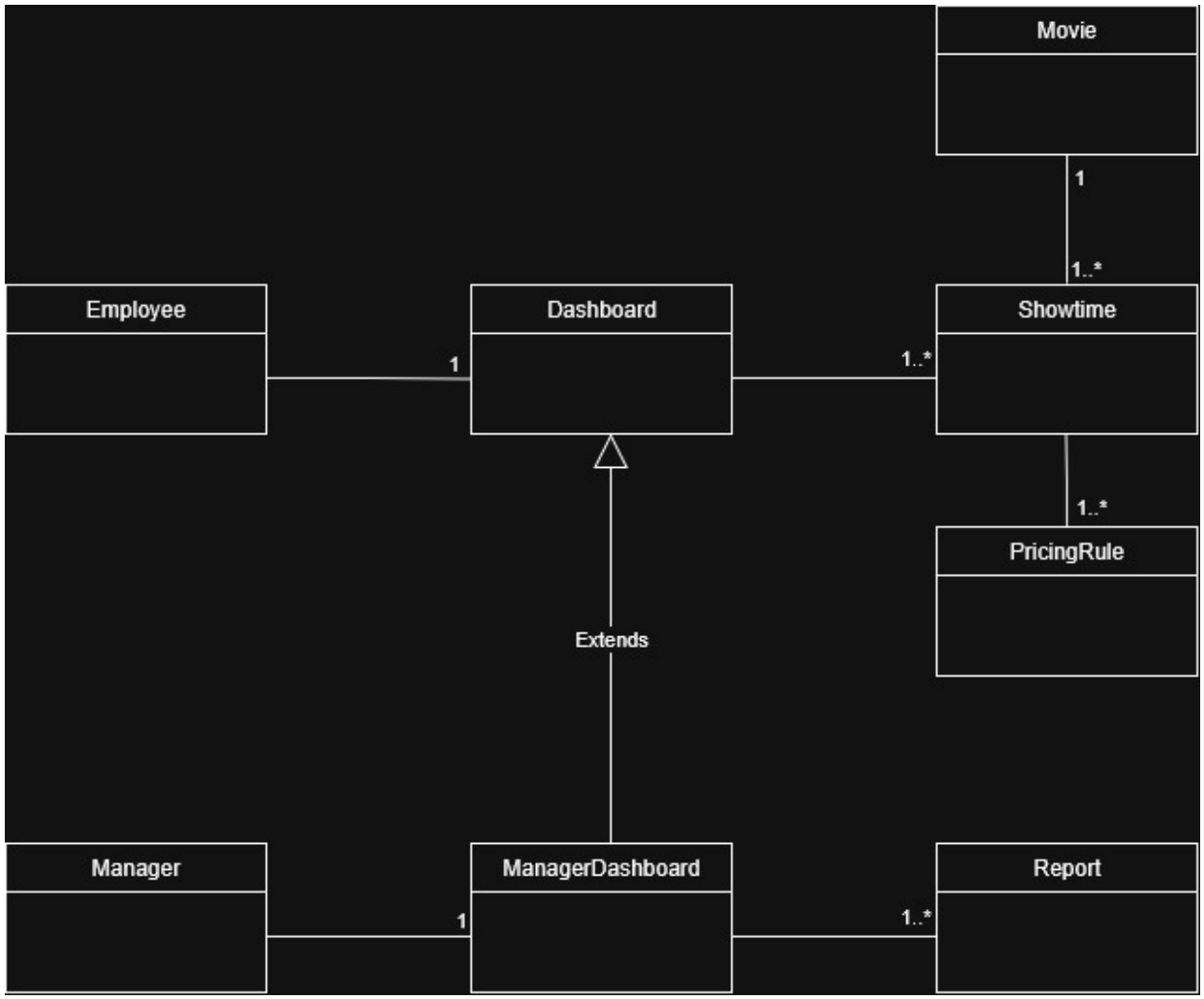
```
    :OrderService->>:SeatHoldManager: confirmHold(IDs)

    :OrderService->>:TicketIssuer: issueTickets(Order)
    note right of :TicketIssuer: **Creator/Low Coupling**: Creates Ticket objects and finalizes records.
    :TicketIssuer-->>:OrderService: ticketsIssued()

    :OrderService->>:NotificationService: sendConfirmation(Order)
    :NotificationService-->>:OrderService: notificationSent()

    :OrderService-->>:CustomerUI: success(Order, Tickets)
    :CustomerUI-->>Customer: showConfirmation()
else Payment Denied / Seats Unavailable
    :OrderService->>:SeatHoldManager: releaseHold(IDs)
    :OrderService-->>:CustomerUI: error(message)
end

sequenceDiagram

actor Customer

participant :CustomerUI

participant :OrderService

participant :SeatHoldManager

participant :PricingEngine

participant :PaymentGateway

participant :Order

participant :TicketIssuer

participant :NotificationService

Customer->>:CustomerUI: selectSeats(IDs)
:CustomerUI->>:OrderService: holdSeats(IDs)
:OrderService->>:SeatHoldManager: holdSeats(IDs)
note right of :SeatHoldManager: **Information Expert**: Tracks and locks seats.
```

```
:SeatHoldManager-->>:OrderService: holdConfirmed()

Customer->>:CustomerUI: placeOrder(paymentInfo)
:CustomerUI->>:OrderService: placeOrder(IDs, paymentInfo)

:OrderService->>:PricingEngine: calculateTotal(IDs)
note right of :PricingEngine: **Low Coupling**: Encapsulates pricing rules.
:PricingEngine-->>:OrderService: totalAmount

:OrderService->>:PaymentGateway: authorize(totalAmount, info)
:PaymentGateway-->>:OrderService: paymentApproved()

alt Payment Approved
  :OrderService->>:Order: **[Creation]** new(IDs, total)
  note right of :Order: **Creator**: OrderService creates the Order.
  :OrderService->>:SeatHoldManager: confirmHold(IDs)

  :OrderService->>:TicketIssuer: issueTickets(Order)
  note right of :TicketIssuer: **Creator/Low Coupling**: Creates Ticket objects and
finalizes records.
  :TicketIssuer-->>:OrderService: ticketsIssued()

  :OrderService->>:NotificationService: sendConfirmation(Order)
  :NotificationService-->>:OrderService: notificationSent()

  :OrderService-->>:CustomerUI: success(Order, Tickets)
  :CustomerUI-->>Customer: showConfirmation()
else Payment Denied / Seats Unavailable
  :OrderService->>:SeatHoldManager: releaseHold(IDs)
  :OrderService-->>:CustomerUI: error(message)
end
sequenceDiagram

actor Customer

participant :CustomerUI
participant :CancellationService

participant :Order
```

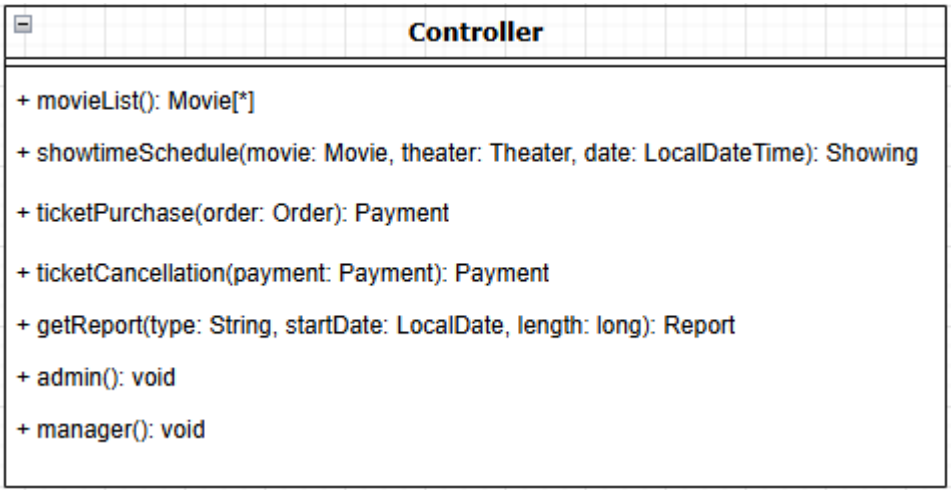
```
participant :SeatInventory

participant :PaymentGateway

participant :Refund

participant :NotificationService

Customer->>:CustomerUI: selectCancel(orderID)
:CustomerUI->>:CancellationService: cancelOrder(orderID)
note right of :CancellationService: **Controller**: Coordinates all steps for cancellation.

:CancellationService->>:Order: isCancellable()
note right of :Order: **Information Expert**: Knows its current status and refund policy.
:Order-->>:CancellationService: true

:CancellationService->>:Order: getPaymentID()
:Order-->>:CancellationService: paymentID

:CancellationService->>:PaymentGateway: processRefund(paymentID)
alt Refund Processed
  :PaymentGateway-->>:CancellationService: refundSuccess()
  :CancellationService->>:Refund: **[Creation]** new(orderID, amount)
  note right of :Refund: **Creator**: CancellationService creates the Refund record.

  :CancellationService->>:Order: updateStatus(CANCELED)
  :CancellationService->>:SeatInventory: releaseSeats(orderID)
  note right of :SeatInventory: **Information Expert**: Updates seat status back to
AVAILABLE.

  :CancellationService->>:NotificationService: sendCancellation(Order)
  :CancellationService-->>:CustomerUI: success(message)
else Refund Failed
  :CancellationService-->>:CustomerUI: error(refundFailure)
end
:CustomerUI-->>Customer: showResult()

sequenceDiagram
```

```
actor Employee

participant :AdminUI

participant :AdminController

participant :Movie

participant :MovieCatalog

Employee->>:AdminUI: enterMovieDetails(details)
:AdminUI->>:AdminController: addMovie(details)
note right of :AdminController: **Controller**: Receives admin request and manages creation.

:AdminController->>:MovieCatalog: isDuplicate(title)
:MovieCatalog-->>:AdminController: false

:AdminController->>:Movie: **[Creation]** new(details)
note right of :Movie: **Creator**: AdminController creates the Movie object.

:AdminController->>:MovieCatalog: saveMovie(Movie)
note right of :MovieCatalog: **Information Expert**: Stores and manages the collection of Movie objects.
:MovieCatalog-->>:AdminController: saveConfirmed()

:AdminController-->>:AdminUI: success(Movie)
:AdminUI-->>Employee: displayNewMovie()
```


Account
Extends
Customer
Refund
1
1
Payment
Order
1
1..*
1
1
1..*
Ticket
Seat
1
1
1..*
1
Movie
Showtime
1
1..*
1..*
1
Auditorium
Seats
1
1..*

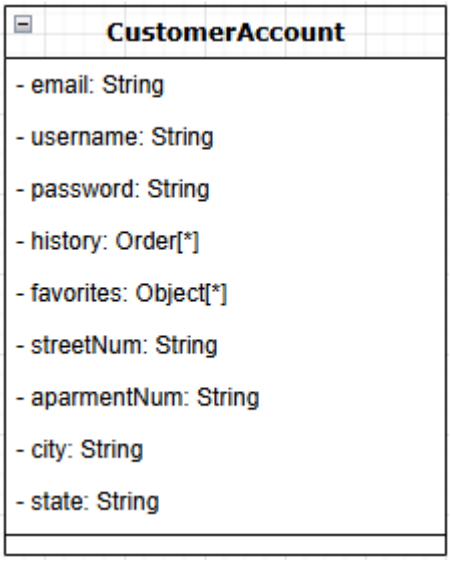

Movie
1
1..*
Employee
Dashboard
Showtime
1
1..*
1..*
PricingRule
Extends
Manager
ManagerDashboard
Report
1
1..*

Data Types and Operation Signatures



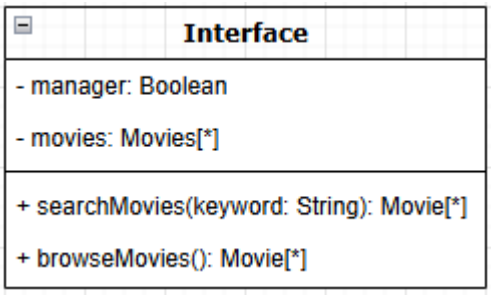
Controller

- **movieList - This put all the current or upcoming movies that the theater will show into an array and return the array.**
- **showtimeScheduler – will create a new showing when given a movie, theater, date, and time.**
- **ticketPurchase – given an order this will process the purchase and return a payment.**
- **ticketCancellation – given a payment this will process a refund and return a payment object.**
- **getReport – given the type, startDate, and length this will compile and return the information from the given period.**
- **admin – will set the user to admin privileges.**
- **manager – will set the user to manager privileges.**



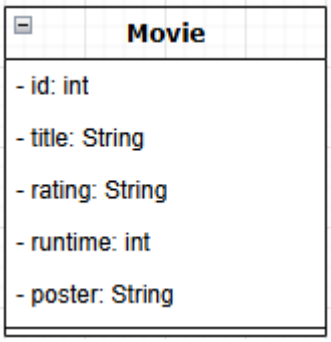
CustomerAccount

- **email - This is the email associated with the customer account.**
- **username - The username associated with the customer account. Used to login.**
- **password – The password hash from our password hash generator. To be compared against the password entered when user is logging in.**
- **history - The order history of the customer account.**
- **favorites - Contains favorites for the user account from dfferent areas.**
- **streetNum – The number and street of the mailing address for the user account.**
- **apartmentNum - The apartment number of the mailing address for the user account.**
- **city – The city of the mailing address for the user account.**
- **state – The state of the mailing address for the user account.**



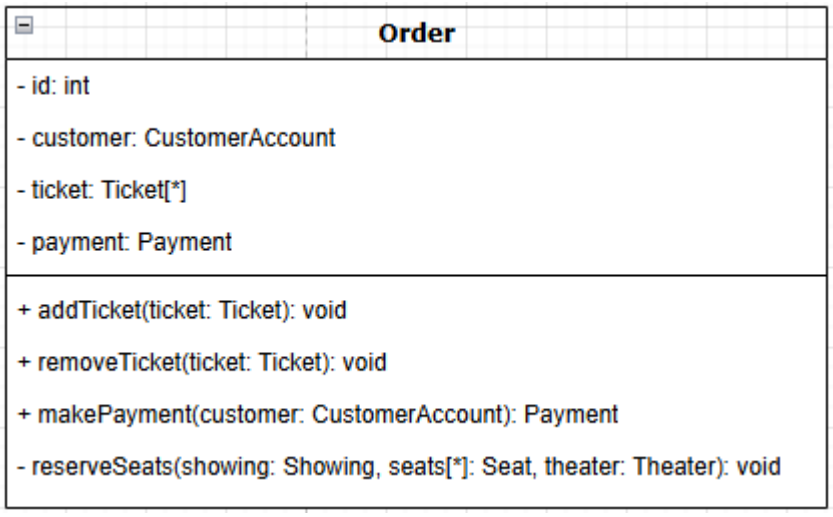
Interface

- **manager - Is the interface in the manager state.**
- **movies – Is an array of all the current or upcoming movies that the theater will show.**
- **searchMovies – Will search through 'movies' with the keyword passed and return an array of matching or similarly titled movies.**
- **browseMovies – return an array of movies that have showtimes and tickets can be purchased for.**



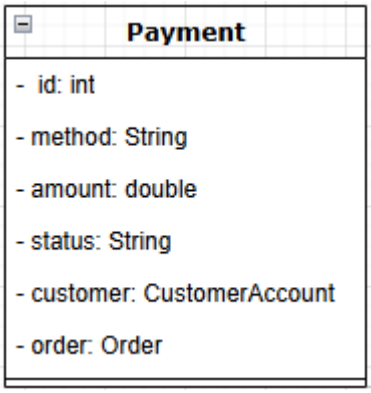
Movie

- **id – This is the unique ID of the movie so it can be tracked if other movies have a matching title.**
- **title - The title of the movie.**
- **rating – The rating given to the movie by the Motion Picture Association.**
- **runtime – The length of the movie in minutes.**
- **poster – link to location of poster for the movie.**



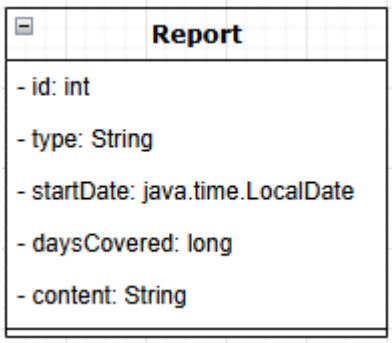
Order

- **id - Unique id so order can be tracked.**
- **customer – The 'CustomerAccount' of the user that is making the order.**
- **ticket – An array of 'Ticket' that have been purchased with this order.**
- **payment – 'Payment' that is associated with this order.**
- **addTicket – adds the given ticket to the order. Can only be used before the payment is processed.**
- **removeTicket – removes the given ticket from the order. Can only be used before payment is processed.**
- **makePayment – Uses the given 'CustomerAccount' to make the process the payment for the order.**
- **reserveSeats – When tickets are added to an order the seats are temporarily reserved to prevent others from selecting the same seats. Once the order is processed the reservation becomes permanent.**



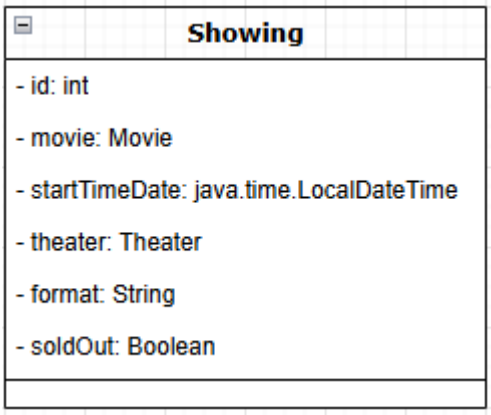
Payment

- **id - Unique id to track the payment by.**
- **method – The payment method used for thsi payment.**
- **amount – The total amount of the payment.**
- **status – The status of the payment. (Completed, pending, cancelled)**
- **customer – The 'CustomerAccount' associated with this payment.**
- **order – The 'Order' associated with the payment.**



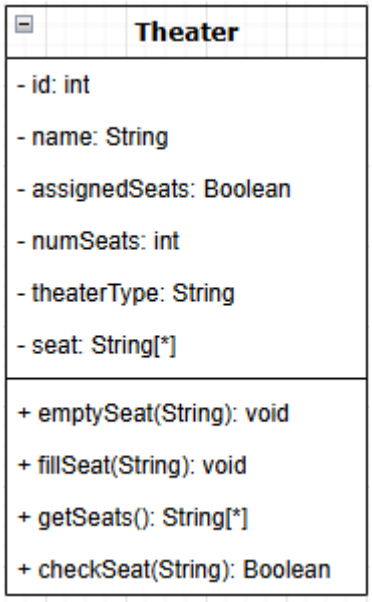
Report

- **id - Unique id used to track the report.**
- **type – The type of report. (unsure but possible select from dropdown or enter a String)**
- **startDate – The date that you want to start getting data for the report.**
- **daysCovered – The number of days that the report will cover from the startDate.**



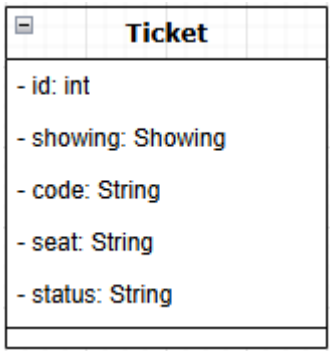
Showing

- **id - Unique id to track the showing by.**
- **movie – The 'Movie' that will be played at the showing.**
- **startTimeDate – The date and time of the showing.**
- **theater – The 'Theater' that the showing will take place in.**
- **format – The format of the movie being shown.**
- **soldOut – Is the showing sold out.**



Theater

- **id - Unique id that the theater can be tracked by.**
- **name – The name of the theater.**
- **assignedSeats – Does the theater have assigned seats.**
- **numSeats – the number of seats that the theater contains.**
- **theaterType – The type of theater. (normal, IMAX)**
- **seat – An array that contains all seat numbers if the theater has assigned seating.**
- **emptySeat – Marks the given seat as open.**
- **fillSeat – Marks the given seat as occupied.**
- **getSeats – Returns and array of all the seats and their status in the theater.**
- **checkSeat – Checks to see if the given seat is occupied. Return true if occupied and false if open.**



Ticket

- **id - Unique id to be used to track the ticket.**
- **showing - The showing that the ticket is associated with.**
- **code – String value of the barcode for the ticket.**
- **seat - If the theater has assigned seating this is the assigned seat.**
- **status – The status of the ticket. (cancelled, refunded, paid)**

Traceability Matrix

Domain Concept	Corresponding Classes	Explaination of Maping
Customer	Customer, CustomerUI	Customer initates actions, UI provides interface to interact with the system
Seat	Seat, SeatHoldManager, SeatInventory	Core object in domain – SeatHoldManager handles temp locks, seatInventory manages seat state
Seat Hold	SeatHoldManager	Controls the holding and releasing of seats during selection and timeouts
Order	Order, OrderService	OrderService orchestrates creation and validation; order stores purchases info
Payment	PaymentGateway, Refund	PaymentGateway authorizes/charges cards; Refund issues reimbursement on cancellations
Ticket	Ticket, TicketIssuer	TicketIssuer generates ticket objects for an order, ticket represents entry to an event
Movie	Movie, MovieCatalog	Movie is core data; MovieCatalog manages list of movies
Admin	AdminUI, AdminController	Staff-side interation for managing theater inventory and schedules
Notification	NotificationService	Sends Confirmations, alerts for orders, and cancellations
Cancellation	CancellationService	Manages refund logic and updating order/seat status

Algorithms and Data Structures

Algorithms

- Seat Selection and Hold Algorithm
    - Ensure that customers can select seats in real time without conflicts
    - Check the requested seat IDs against the SeatInventory for availability
    - If all seats are available, temporarily hold the seats in SeatHoldManager and set a timeout
    - If payment is not complete within the timeout period, automatically release held seats
- Order Processing Algorithm
    - Convert held seats into a confirmed order and associated tickets
    - Calculate total price using the PricingEngine
    - Authorize payment through PaymentGateway
    - On successful payment, created an Order object and issue corresponding ticket objects
    - Update seat status in SeatInventory from HELD to RESERVED
- Cancellation and Refund Algorithm
    - Cancel ordders and release seats while issuing refunds
    - Check if the order is eligible for cancellation
    - Update order status to “Cancelled”
    - Process refund using Refund service
    - Update seat availability in SeatInventory
    - Notify the customer vis NoticationService
- Reporting Algorithm
    - Generate sales and booking reports efficiently
    - Query orders and ticket information for the specified date range and filters
    - Aggregate data by movie, auditorium, or date
    - Format and return report object for display on manager/admin dashboard

Data Structures

- Arrays/Lists
    - Used for storing movies (MovieCatalog), showtimes (ShowtimeSchedule), tickets in an order (Order.ticket), and seat lists (Theater.seat)
- Hash Maps/Dictionaries
    - Used for mapping seat IDs to seat objects in SeatInventory and mapping movie IDs to Movie objects in MovieCatalog
- Queues
    - Used for managing seat hold timeouts in SeatHoldManager
- Objects/Classes
    - Core domain objects include CustomerAccount, Move, Showtime, Ticket, Order, Payment, and Seat

Concurrency

- Our system has limited concurrency concers, primarily in handling multiple users reserving seats simultaneously
    - Threads
        - Seat reservation and timeout release could potentially be handled in separate threads for efficiency
        - SeatHoldManager runs a background thread to release expired seat holds
- Synchronization
    - SeatInventory updates are synchronized to prevent race conditions when multiple customers attempt to reserve the same seat
    - Locks or atomic operations are used to ensure that each seat can only be held or reserved by one customer at a time

Design of Tests

Test Cases for Unit and Integration Testing

Customer Scenario – Create Account

1) Navigate to sign-up page
2) Enter all information requested
3) Submit sign-up
4) Receive email notification of account creation

Tests the creation of an account.

Customer Scenario – Login

1. Select “Login”
2. Enter username and password
3. Click “Login”
4. Directed to “Home Page”

This test case will cover one of the most common usage scenarios. Testing this will allow all our other customer use cases to start from the home page instead of making the user log in for every test.

Customer Scenario – Purchase Tickets

1) Browses available movies
2) Selects movie

3) Look at movie's showtimes
4) Select a showtime
5) View available seats in theater
6) Select seats
7) Enter payment info
8) Confirm purchase
9) Receive confirmation, receipt, and tickets

Purchasing tickets at a location other than the movie theater is one of the most important functions of our software. This will evaluate the processes of navigating and selecting movies, theaters, and seats as well as testing our payment process.

Customer Scenario – Refund Tickets

1) Go to purchase history
2) Select purchase to refund
3) Confirm to refund purchase
4) Receive confirmation and receipt of refund

This will test our purchase history and our payment return process.

Employee Scenario – Login

1. Select "Login"
2. Enter username and password
3. Click "Submit"
4. Directed to "Admin Dashboard"

Will test login for user will elevated permissions.

Employee Scenario – Add Movie

1) Select "Add Movie"
2) Enter movie information and upload image(s)
3) Submit new movie

Tests that an employee can add a movie, and a normal user can't add a movie.

Employee Scenario – Add Showing

1) Select movie
2) Select "Add Showing"
3) Input showtime, date, and theater
4) Submit showing

Tests that an employee can add a showing, and a normal user can't add a showing.

Employee Scenario – Modify Showing

1. Select showing
2. Click "Modify"
3. Make changes to theater, movie, and showtime
4. Submit changes to showing

Tests that an employee can modify a showing, and a normal user can't modify a showing.

Employee Scenario – Delete Showing

1. Select showing
2. Click “Modify”
3. Click “Delete Showing”
4. Confirm the deletion of showing

Tests that an employee can delete a showing, and a normal user can’t delete a showing.

Our coverage for testing will be at a high level. For our unit testing, we will primarily be using white box testing to develop our classes and switch to black box testing, once the class passes white box testing, to verify that the class functions as intended. Our testing process will extensively cover the specifications required for our software. This will be accomplished because of our selected test cases that will evaluate many of our classes numerous times throughout the process.

For our integration strategy we will use vertical integration testing as it aligns with our agile development approach. The test cases (user stories) listed above will be how we will break down our development approach. These user stories have been selected because they are typical usage scenarios, and their inner workings overlap to cover the testing for our entire system.

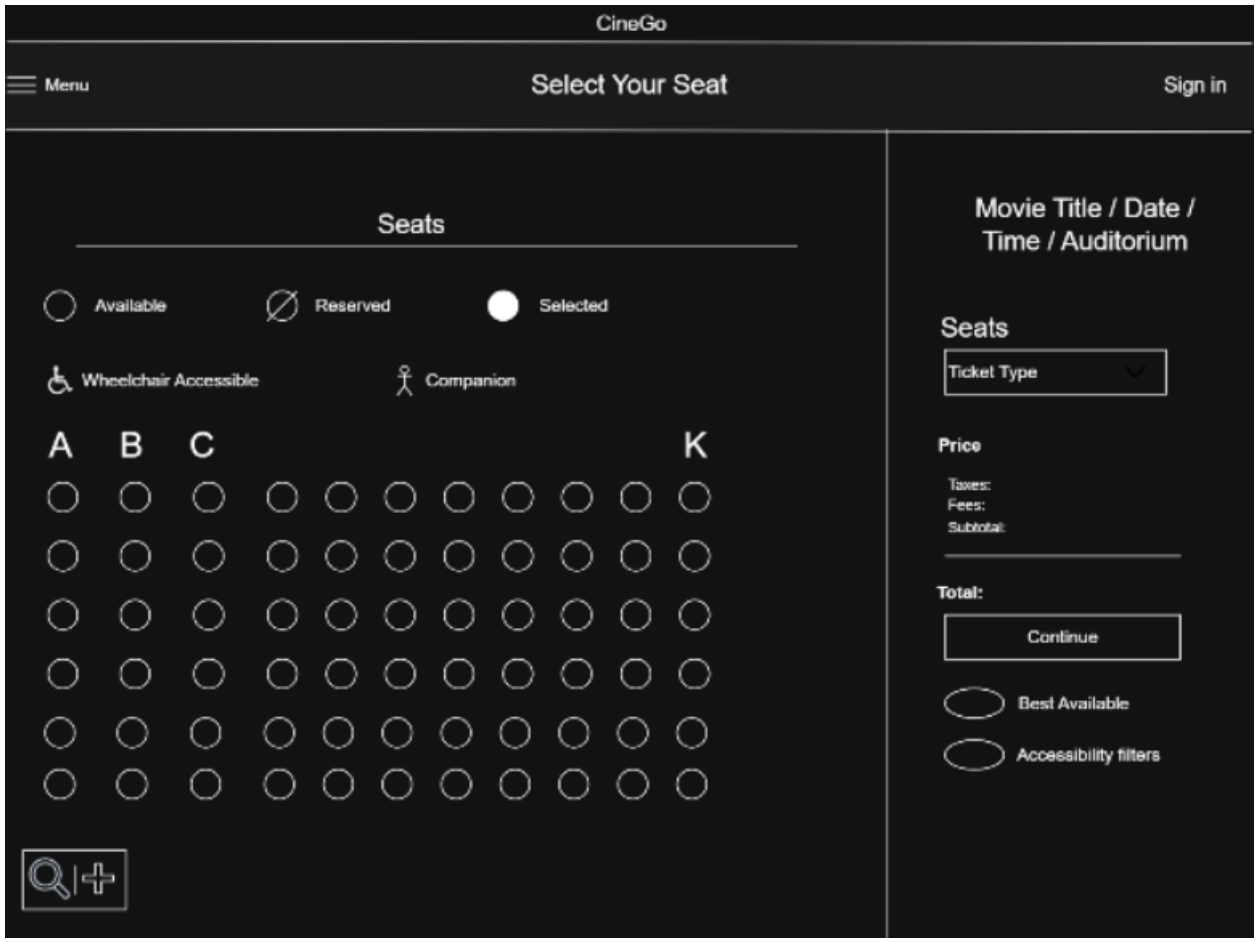
User Interface Design and Implementation

<u>Customer Homepage:</u>

The customer can see what movie is showing on the front page. Homepage contains the option for the customers to search for the app for movies and give options for customers to sign in to their account and view their purchased tickets and purchase history.

Seat Map Selection Interface:

The user can select available seats for a chosen movie through the seat map selection interface. The interface displays the theater layout, and shows which seats are available, reserved, and taken using clear legends. The design prevents double booking using real-time seat availability updates and provides seats that are accessibility friendly.



Admin Dashboard:

The Admin Dashboard Interface provides theater admins/managers with a user-friendly platform to manage all core operations of the theater system. The interface offers quick access to management tools where the admin can change pricing, add movies and showtimes, and see reports of sales and revenue.

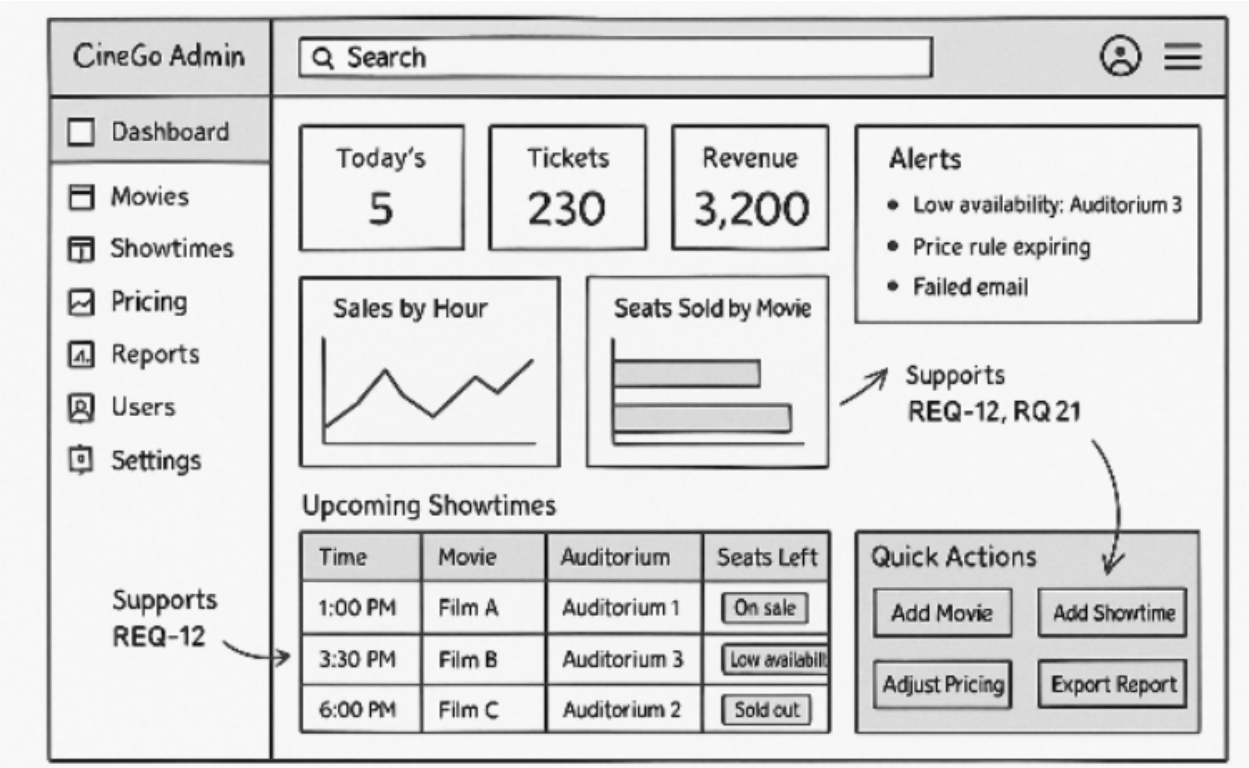


Figure X. Admin Dashboard (Overview)

Movie Details Interface:

The Movie Details Interface shows information about the selected movies within the app. The interface shows available date, showtimes, auditorium, movie length, poster, rating, genre, casts, director, and if the movie supports accessibility issues. Users can conveniently select preferred showtime to purchase their tickets, and it will bring them to the seat selection interface for their seats.

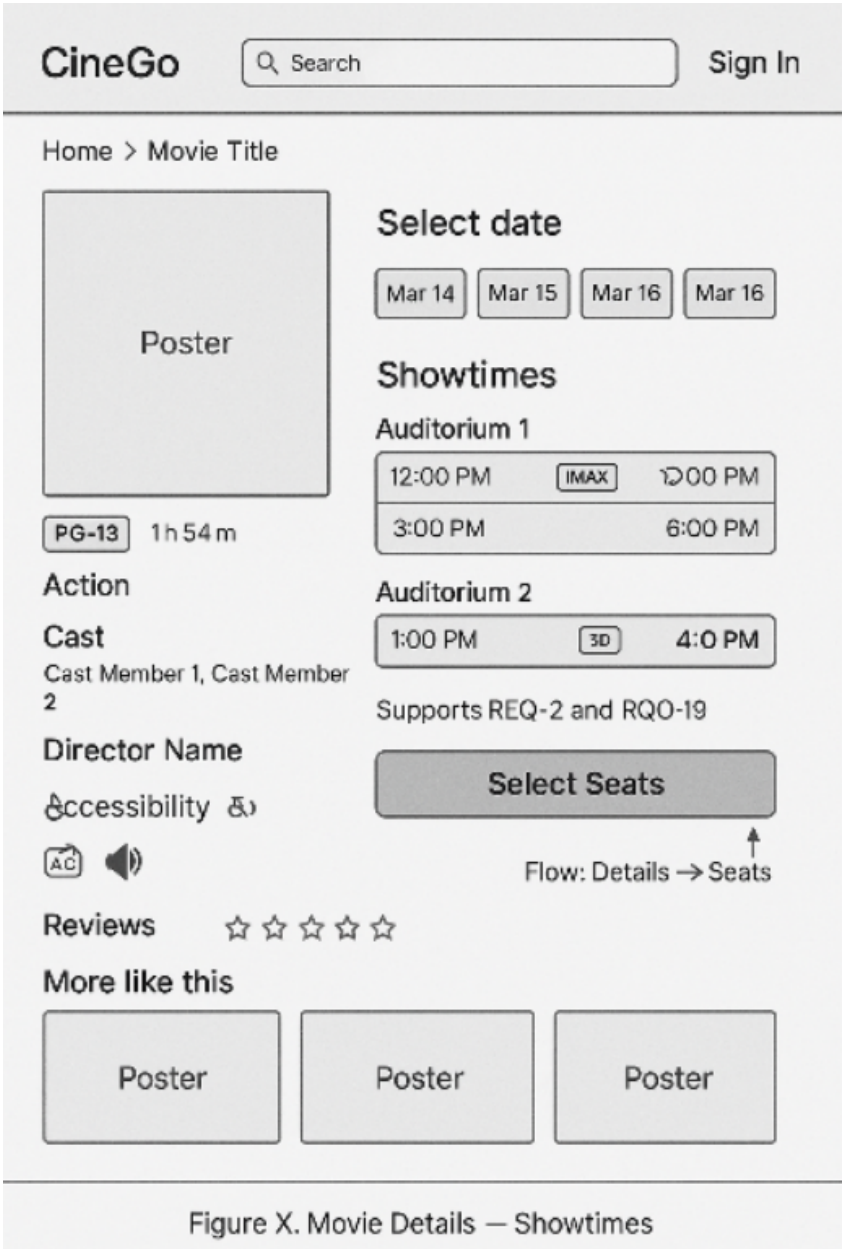


Figure X. Movie Details — Showtimes

Checkout Interface:

Checkout Interface is the final step in the ticket purchasing process, User can review their tickets to make sure everything is correct before purchasing it. This page shows the summary of the user's booking details, showing movie title, seat, showtime, auditorium id, and total cost. Users can choose their preferred method to pay and apply a discount code if available.

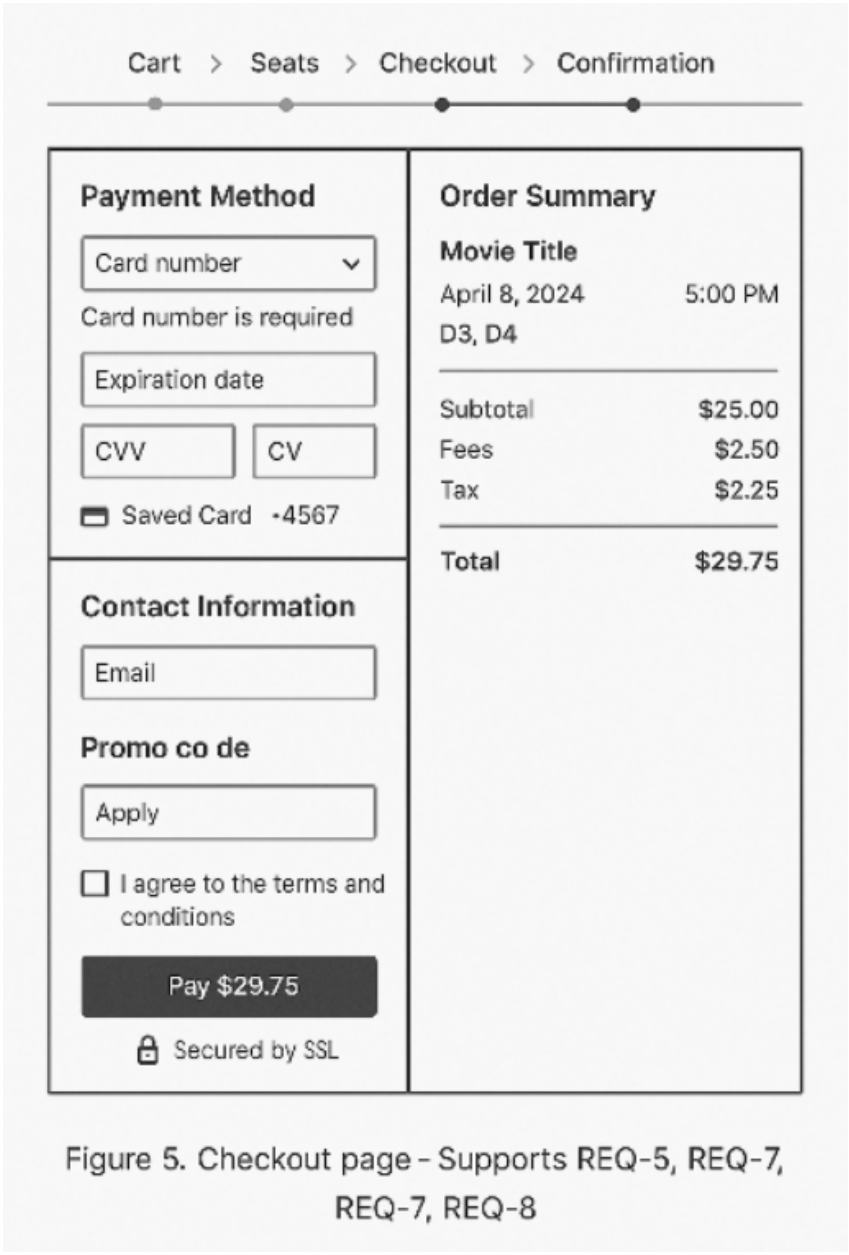
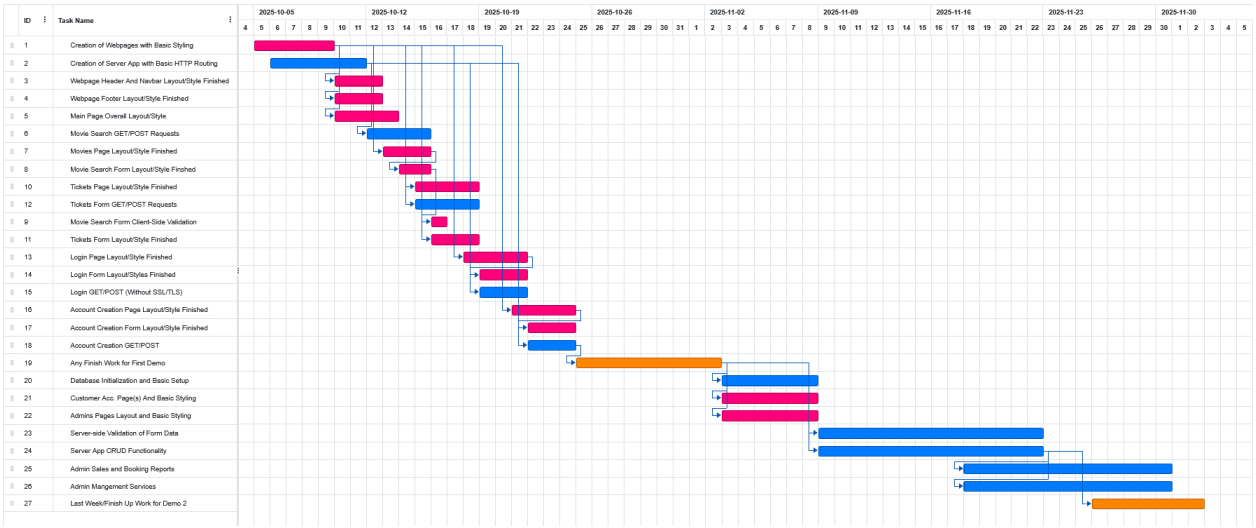


Figure 5. Checkout page - Supports REQ-5, REQ-7, REQ-7, REQ-8

Plan of Work



Breakdown of Coding Responsibilities – To be determined when starting to develop the program. Starting 10/17.

References

https://eceweb1.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf - textbook