AcmePlex

ENSF 614 - Design Document

Last Updated: Dec 1, 2024

Status: Final

Authors: Team 16

Contributors: Frank Ma, Jaskirat Singh, Saba Soghraty, Samin Hazeri

Project Documentation for Movie Booking System

1. Introduction and Scenarios

1.1 Project Overview

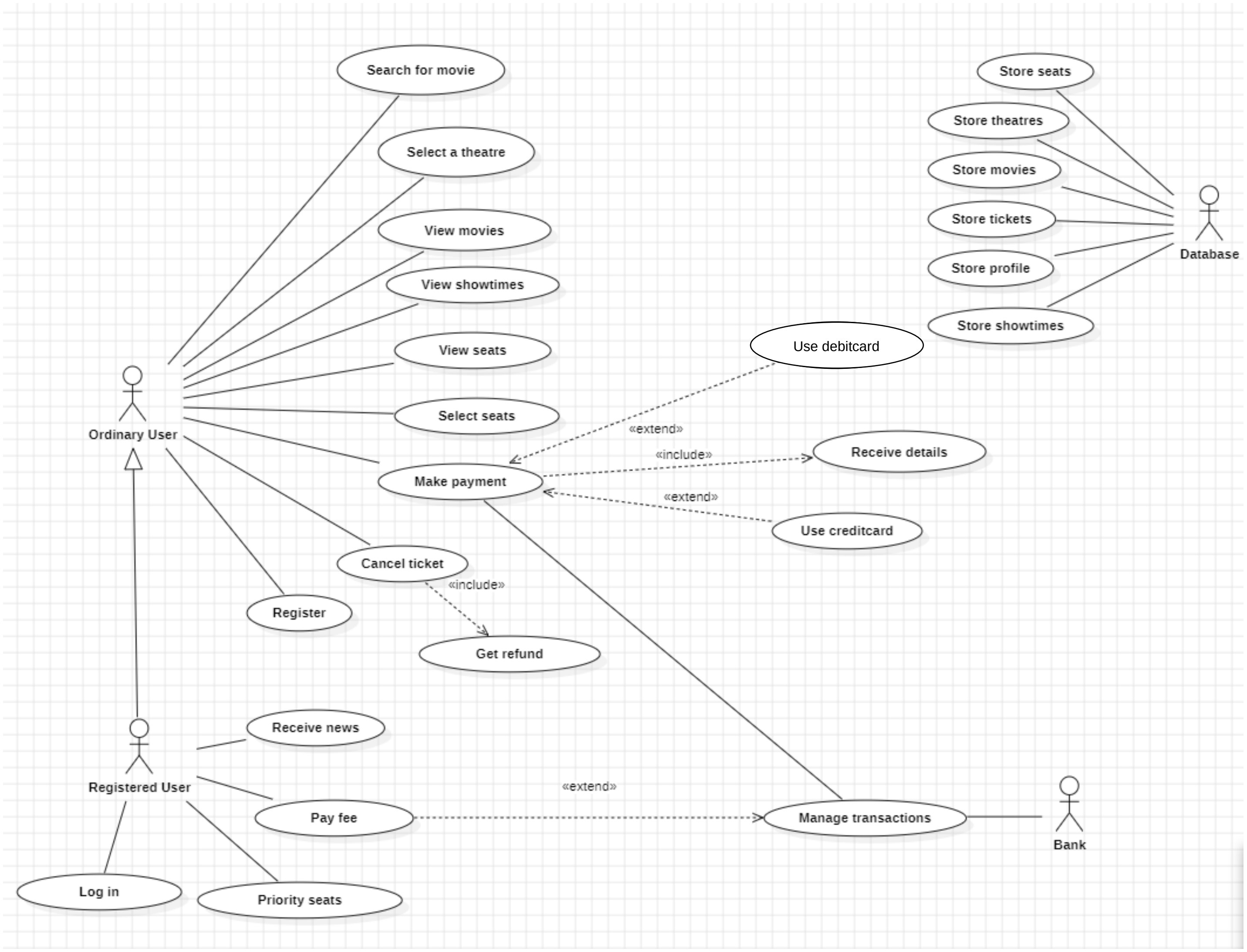
The Movie Booking System is a web-based application designed to simplify the process of booking movie tickets. Users can search for movies, view seat availability, and book tickets. The system also supports features for registered users, such as ticket cancellations and personalized notifications.

1.2 Objectives

Build a functional movie booking system with core features:

- Movie browsing.
- Seat selection and ticket reservation.
- Payment processing.

1.3 Use Case Diagram



1.4 Scenarios

Use Case 1: Ordinary User Searching for a Movie

- Ordinary User searches for a specific movie title through the app's search feature. They enter the name, and the system displays a list of movies matching the title, including details like the release date and genre.

Use Case 2: Ordinary User Viewing Available Seats

- Ordinary User selects a showtime from the available options.
- The system retrieves the SeatingChart with color-coded seats indicating availability, from the associated Theater object.
- Ordinary User selects a Seat.

- The system highlights the selected Seat and passes it onto the next page when user is ready for payment.

Use Case 3: Ordinary User Making a Payment

- Ordinary User makes payment after selecting a movie and seat. They enter their creditor debit card details and submit payment. The system confirms the payment and shows the ticket.
- Ordinary User makes a *payment* but enters incorrect card details. The system displays an error message, prompting them to re-enter valid details before confirming the payment.

Use Case 4: Registered User (RU) Cancelling a Ticket

- Registered User selects the Reservation object they want to cancel.
- Registered User requests to cancel their ticket more than 72 hours before the show. The system confirms eligibility and issues a full credit to their account, valid for one year.
- Registered User requests a cancellation within the 72 hours before showtime, so the system declines the request due to the 72-hour policy, prompting them to review the cancellation policy.

Use Case 5: Registered User Receiving News Announcements

- Registered User receives early access news.

Use Case 6: System Limiting Pre-booked Seats for Registered Users

- ○ The system enforces a 10% limit on seats that registered users can pre-book for a showtime.

---

2. System Design

2.1 Architecture Overview

The system follows a multi-layered architecture:

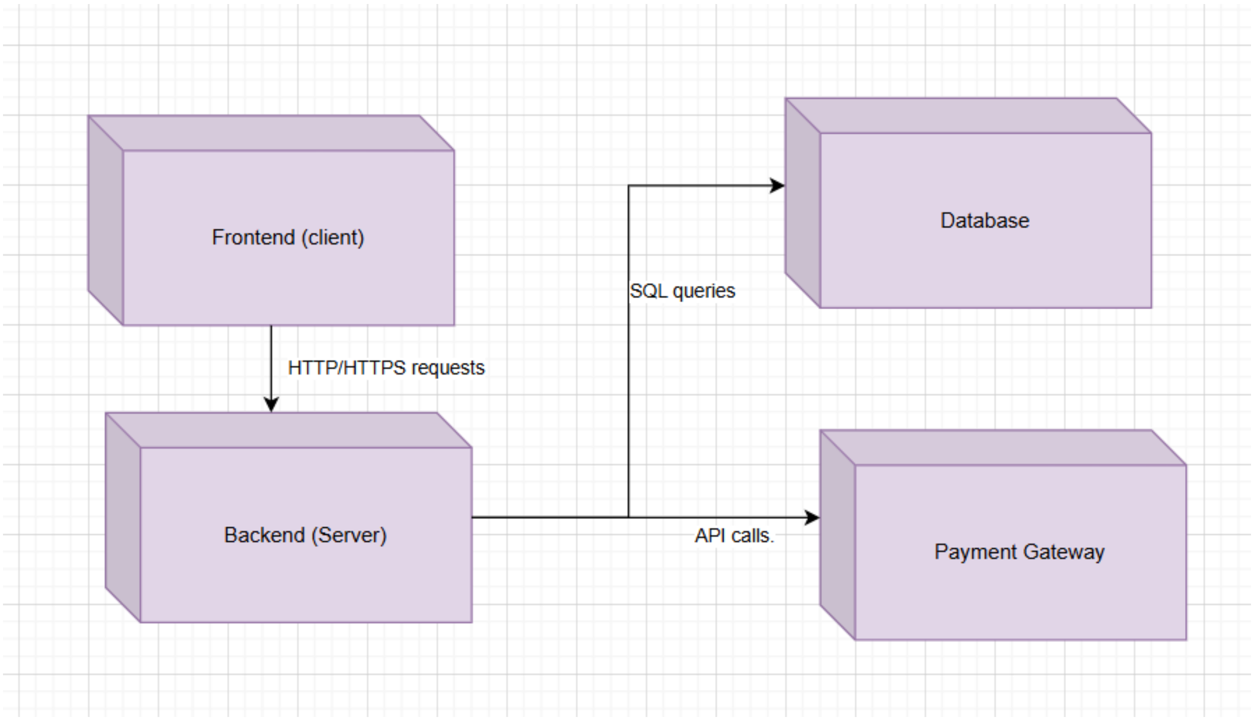
Frontend: Built with a React.js framework, providing an intuitive user interface.
Backend: Developed using Spring Boot, managing business logic and API endpoints.
Database: MySQL is used for data persistence.
External Services: Integrated with a payment gateway for secure transactions.

2.2 Deployment Diagram

The deployment involves:

- **Client:**
  Represents the browser or mobile app interface. Users interact with the system through this node to perform actions like searching for movies, booking tickets, and making payments.
- **Application Server:**
  Hosts the backend services and APIs. Manages business logic, processes requests, and communicates with the database and external services.
- **Database Server:**
  Stores user, movie, ticket, and payment information. Provides persistent data storage for all application entities.
- **Payment Gateway:**
  Handles secure payment processing. Processes transactions and provides payment confirmation to the application server.

Figure: Deployment diagram.



---

3. Class Diagram

3.1 Key Classes

Controllers (`<<boundary>>`)

- MovieController: Handles movie search and details (getAllMovies(), searchMovie()).
- SeatController: Manages seat availability and reservations.
- PaymentController: Processes payments.
- TicketController: Manages ticket creation and cancellations.
- ShowtimeController: Fetches showtime and seat details.
- NotificationController: Sends user notifications.
- HomeController: Provides homepage and admin dashboard access.

Services (`<<control>>`)

- MovieService, SeatService, PaymentService, TicketService, ShowtimeService: Handle business logic for respective domains.

Entities (`<<entity>>`)

- User: Core user class (userID, email, password).
- RegisteredUser: Extends User with features like cancellation without fees.
- Movie: Represents a movie (movieID, title, genre).
- Showtime: Scheduled movie showing (showtimeID, theatreID, seats).
- Seat: Represents a theater seat (seatID, status).
- Ticket: Represents a booked ticket (ticketID, seatNumber, showtime).
- Payment: Tracks payments (paymentID, amount, status).

Repositories (`<<repository>>`)

- MovieRepository, SeatRepository, TicketRepository, UserRepository, ShowtimeRepository, PaymentRepository: Perform CRUD operations for entities.

3.2 Packaging

1. Domain Classes (Entity Layer)

- **Purpose**: Represent core business data like User, Movie, Seat, and Ticket.
- Layer: Entity/Domain.
- **Interactions**: Used by **Control Classes** and persisted via **Repository Classes**.

2. Repository Classes (Data Access Layer)

- **Purpose**: Manage database operations for domain classes (e.g., UserRepository, MovieRepository).

- **Layer**: Data Access.
- **Interactions**: Accessed by **Control Classes** for CRUD operations.

3. Control Classes (Service Layer)

- **Purpose**: Contain business logic (e.g., MovieService, PaymentService).
- **Layer**: Service/Business Logic.
- **Interactions**: Bridge between **Boundary Classes** and **Repository Classes**.

4. Boundary Classes (Controller Layer)

- **Purpose**: Handle user interactions and API endpoints (e.g., MovieController, SeatController).
- **Layer**: Controller/Boundary.
- **Interactions**: Pass user requests to **Control Classes** and return responses.

Figure: Package Diagram

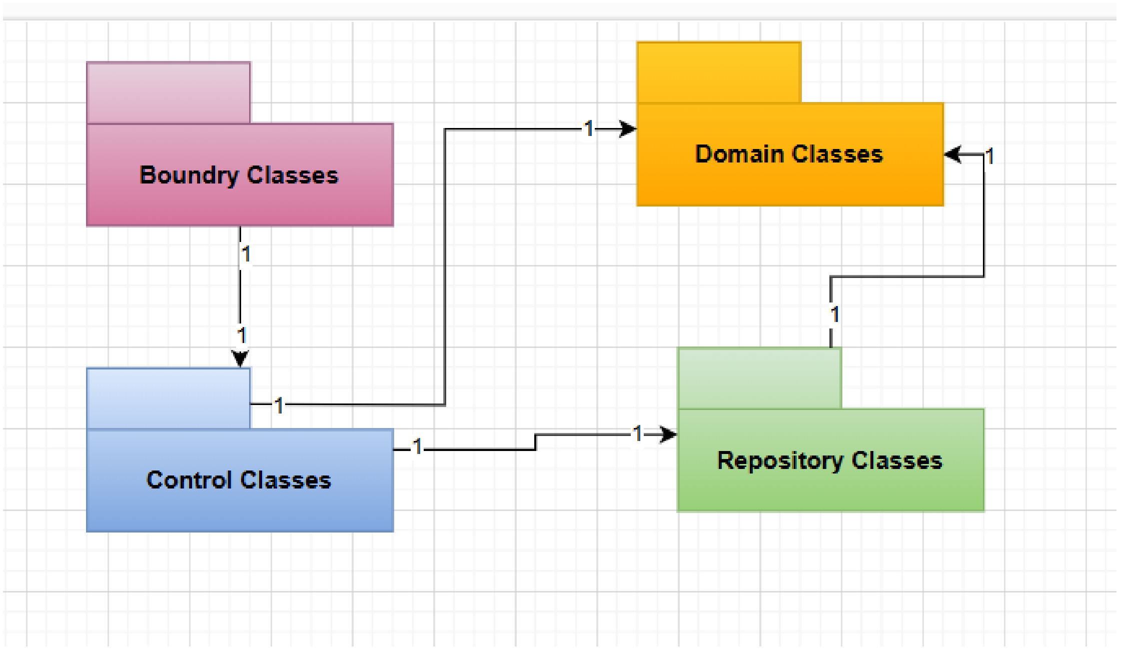
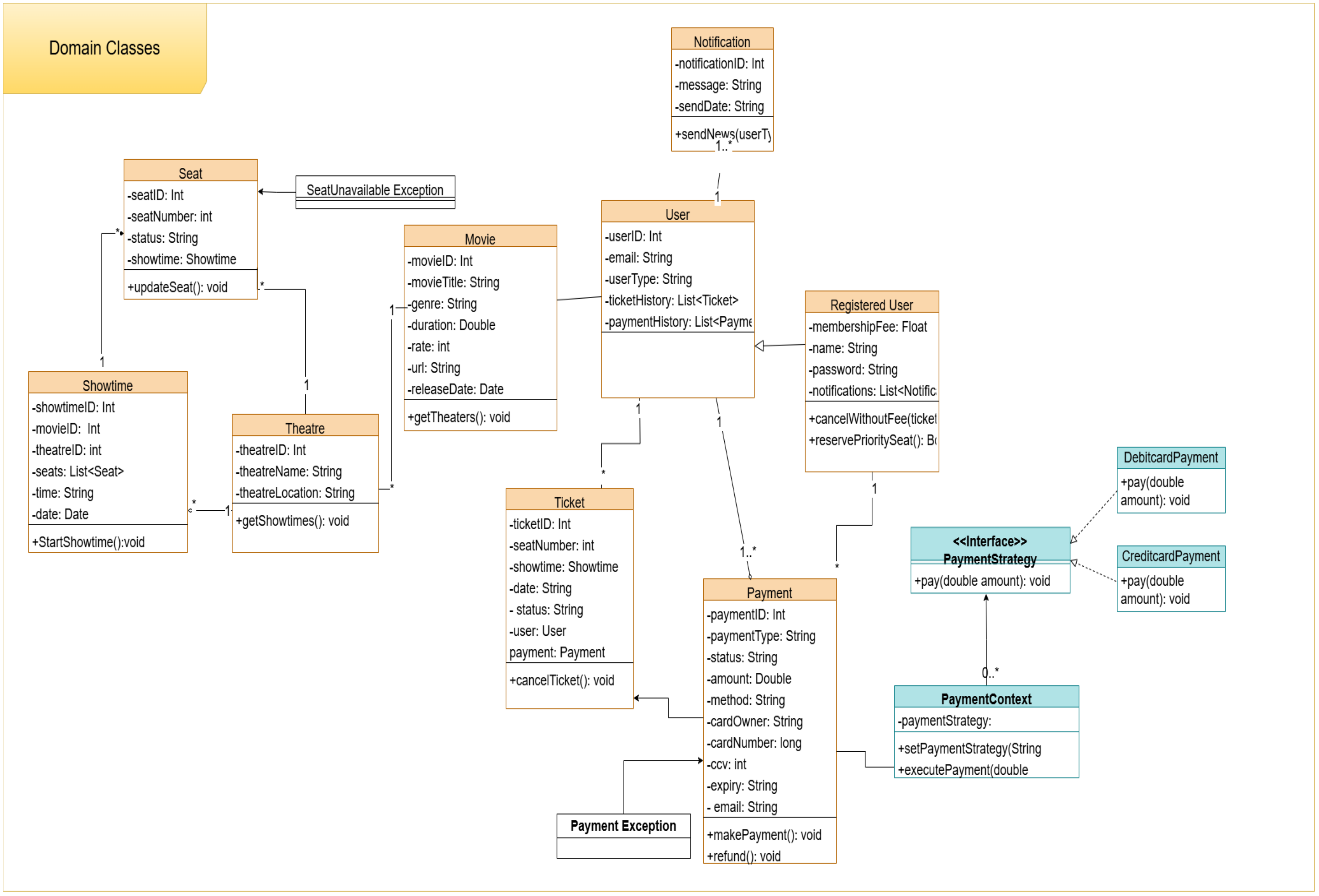
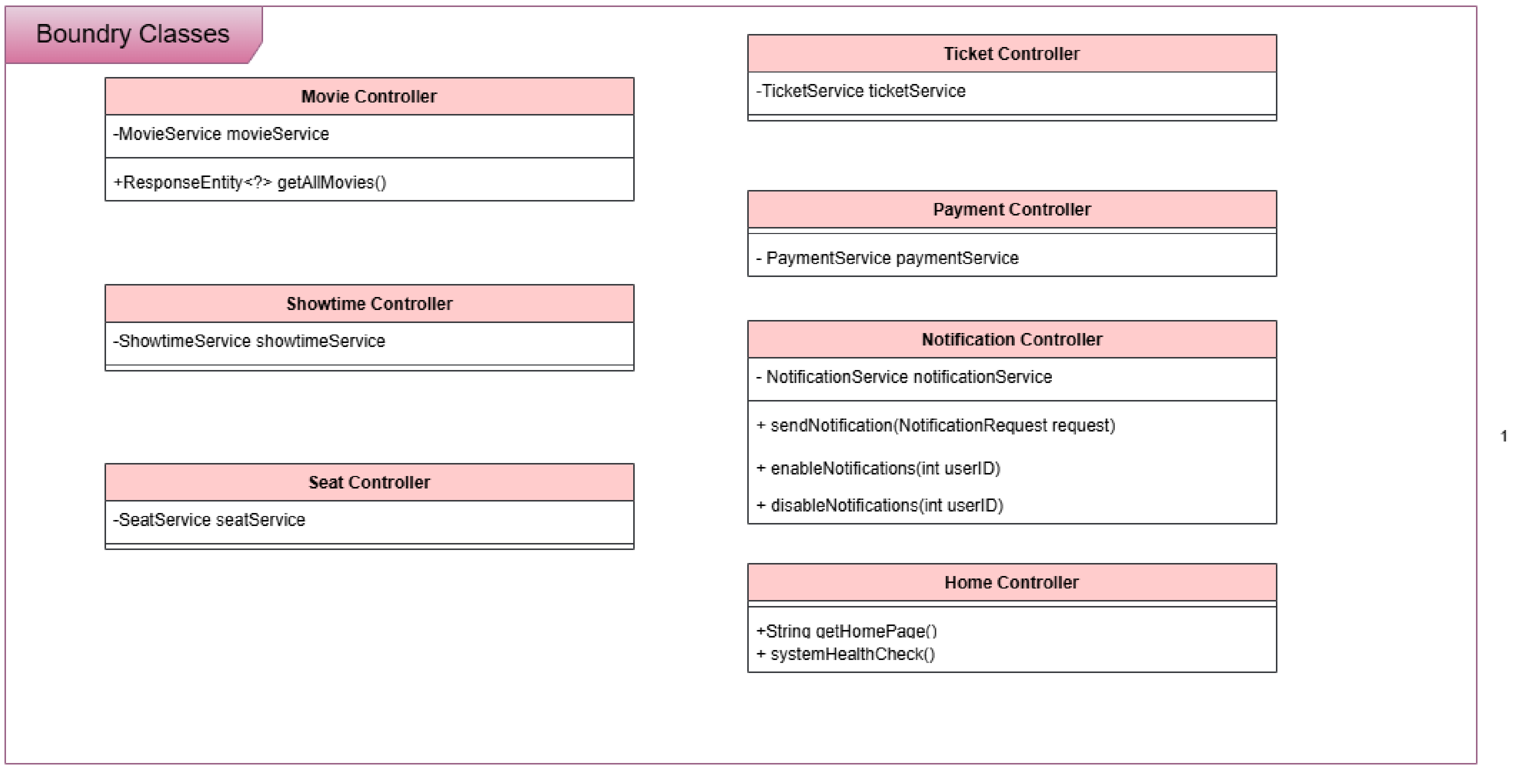
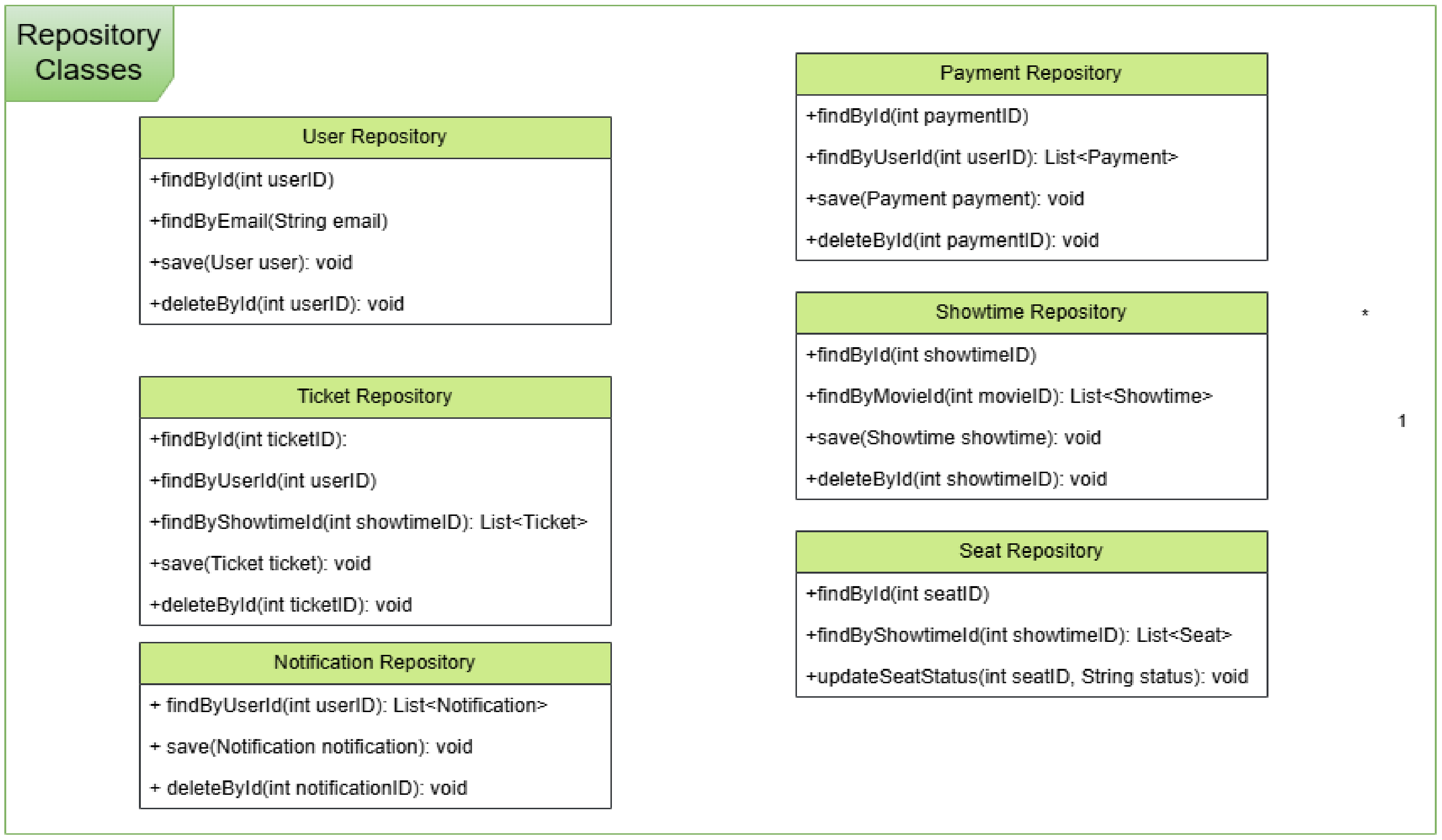
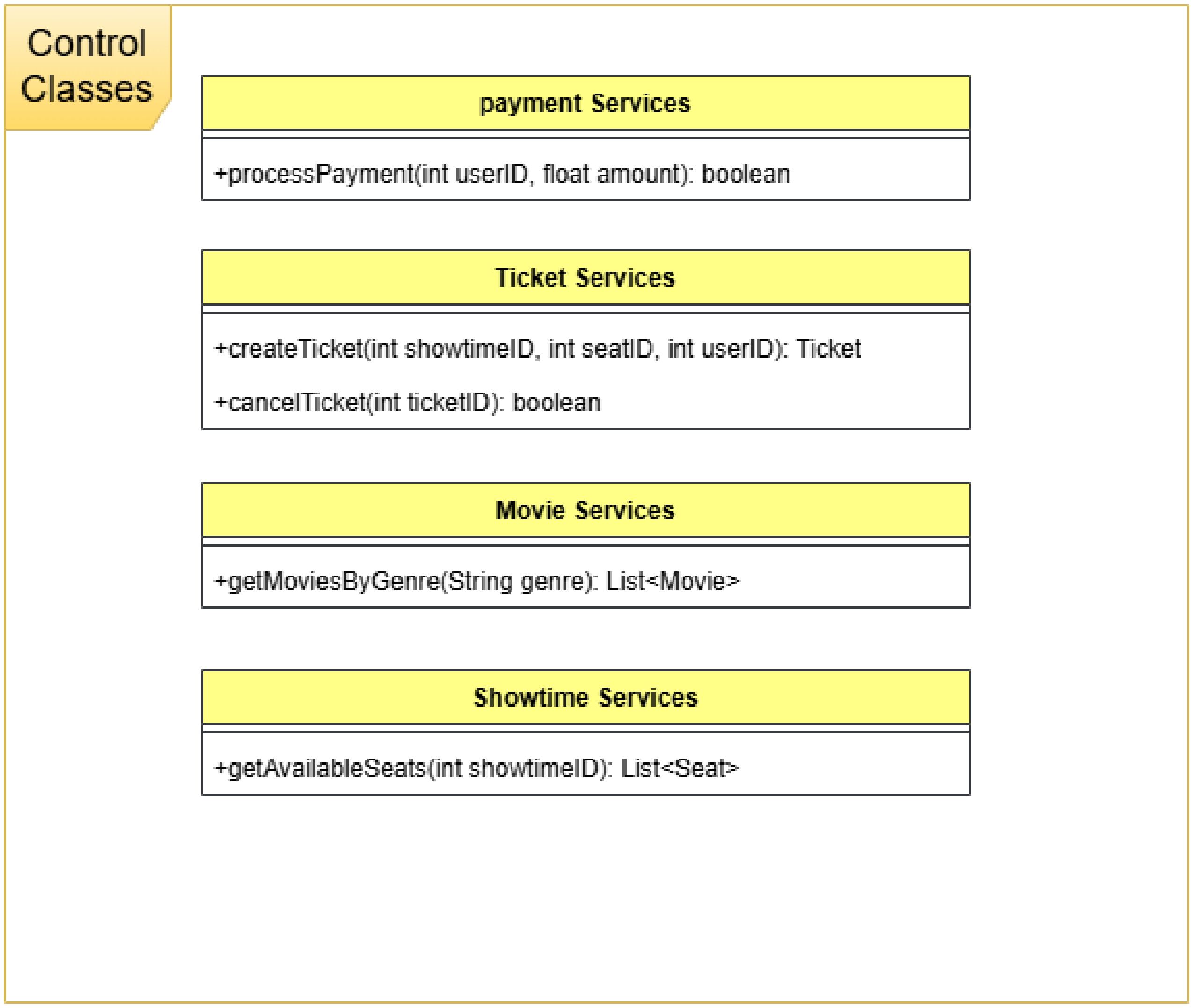


Figure: Class diagram. (a picture of the class diagram is attached to this file Horizontally)


Domain Classes
Notification
-notificationID: Int
-message: String
-sendDate: String
+sendNews(userTy
1..*
1
Seat
-seatID: Int
-seatNumber: int
-status: String
-showtime: Showtime
+updateSeat(): void
SeatUnavailable Exception
Movie
-movieID: Int
-movieTitle: String
-genre: String
-duration: Double
-rate: int
-url: String
-releaseDate: Date
+getTheaters(): void
User
-userID: Int
-email: String
-userType: String
-ticketHistory: List<Ticket>
-paymentHistory: List<Payme
Registered User
-membershipFee: Float
-name: String
-password: String
-notifications: List<Notific
+cancelWithoutFee(ticket
+reservePrioritySeat(): Bo
Showtime
-showtimeID: Int
-movieID: Int
-theatreID: int
-seats: List<Seat>
-time: String
-date: Date
+StartShowtime():void
Theatre
-theatreID: Int
-theatreName: String
-theatreLocation: String
+getShowtimes(): void
Ticket
-ticketID: Int
-seatNumber: int
-showtime: Showtime
-date: String
- status: String
-user: User
payment: Payment
+cancelTicket(): void
Payment
-paymentID: Int
-paymentType: String
-status: String
-amount: Double
-method: String
-cardOwner: String
-cardNumber: long
-ccv: int
-expiry: String
- email: String
+makePayment(): void
+refund(): void
Payment Exception
<<Interface>>
PaymentStrategy
+pay(double amount): void
DebitcardPayment
+pay(double amount): void
CreditcardPayment
+pay(double amount): void
PaymentContext
-paymentStrategy:
+setPaymentStrategy(String
+executePayment(double
1
*
1
*
1
*
1
*
1
*
1
1..*
1
*
0..*


Boundry Classes
Movie Controller
-MovieService movieService
+ResponseEntity<?> getAllMovies()
Showtime Controller
-ShowtimeService showtimeService
Seat Controller
-SeatService seatService
Ticket Controller
-TicketService ticketService
Payment Controller
- PaymentService paymentService
Notification Controller
- NotificationService notificationService
+ sendNotification(NotificationRequest request)
+ enableNotifications(int userID)
+ disableNotifications(int userID)
Home Controller
+String getHomePage()
+ systemHealthCheck()
1


Repository Classes
User Repository
+findById(int userID)
+findByEmail(String email)
+save(User user): void
+deleteById(int userID): void
Ticket Repository
+findById(int ticketID):
+findByUserId(int userID)
+findByShowtimeId(int showtimeID): List<Ticket>
+save(Ticket ticket): void
+deleteById(int ticketID): void
Notification Repository
+ findByUserId(int userID): List<Notification>
+ save(Notification notification): void
+ deleteById(int notificationID): void
Payment Repository
+findById(int paymentID)
+findByUserId(int userID): List<Payment>
+save(Payment payment): void
+deleteById(int paymentID): void
Showtime Repository
+findById(int showtimeID)
+findByMovieId(int movieID): List<Showtime>
+save(Showtime showtime): void
+deleteById(int showtimeID): void
Seat Repository
+findById(int seatID)
+findByShowtimeId(int showtimeID): List<Seat>
+updateSeatStatus(int seatID, String status): void
*
1


Control Classes
payment Services
+processPayment(int userID, float amount): boolean
Ticket Services
+createTicket(int showtimeID, int seatID, int userID): Ticket
+cancelTicket(int ticketID): boolean
Movie Services
+getMoviesByGenre(String genre): List<Movie>
Showtime Services
+getAvailableSeats(int showtimeID): List<Seat>

---

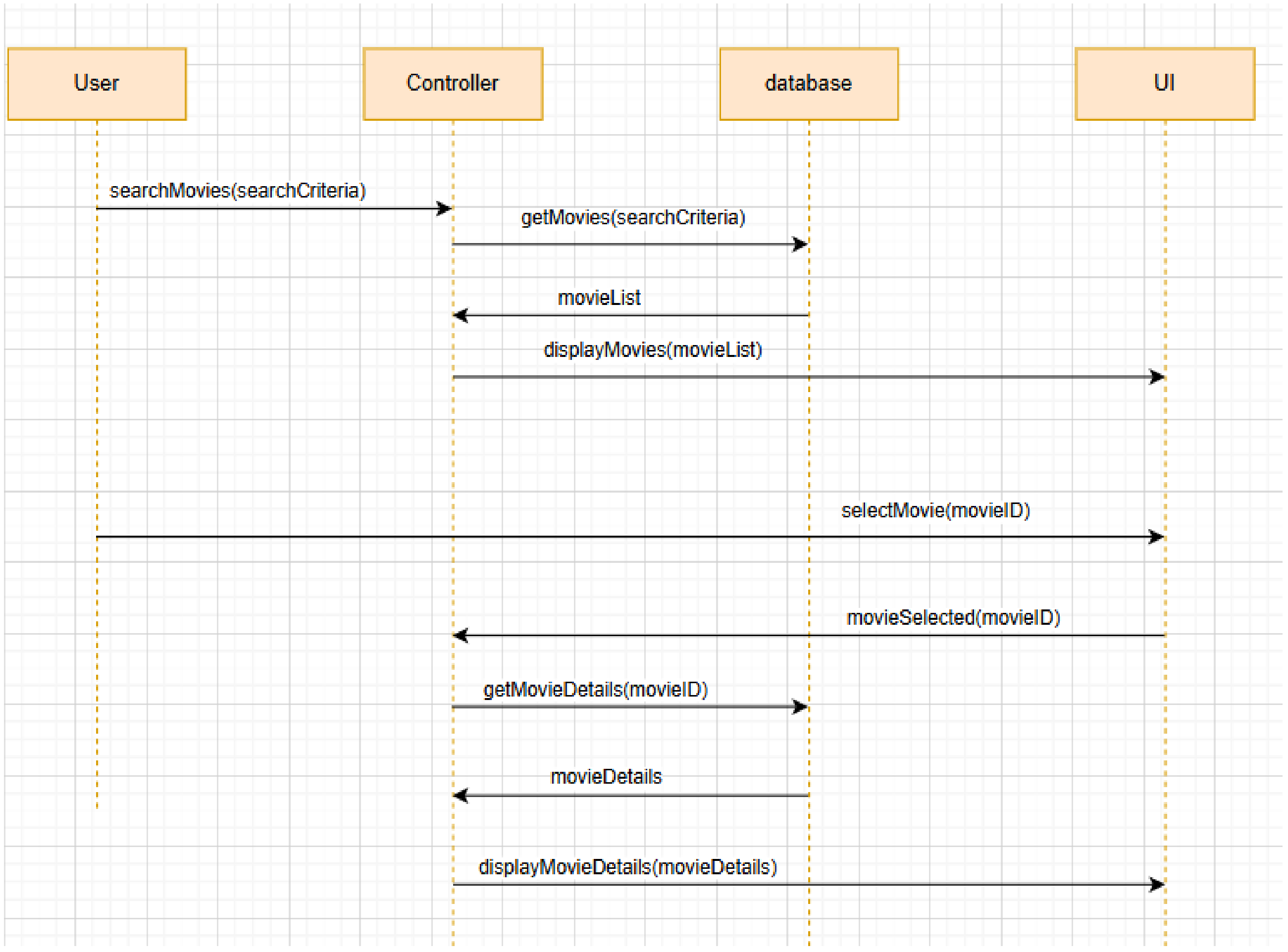
5. Sequence Diagrams

5.1 Searching for a Movie

- Participants:
    1. **User**: Initiates the search.
    2. **Controller**: Handles search logic.
    3. **Database**: Retrieves movie details.
    4. **UI**: Displays results to the user.
- Steps:
    1. User inputs search criteria.
    2. Controller requests movie data from the database.
    3. Database returns a list of movies matching the criteria.
    4. Controller displays the movies to the user via the UI.

5. User selects a movie to view details.
6. Controller fetches and displays movie details.

- **Visual Reference:** Sequence Diagram 1. Done by Frank.

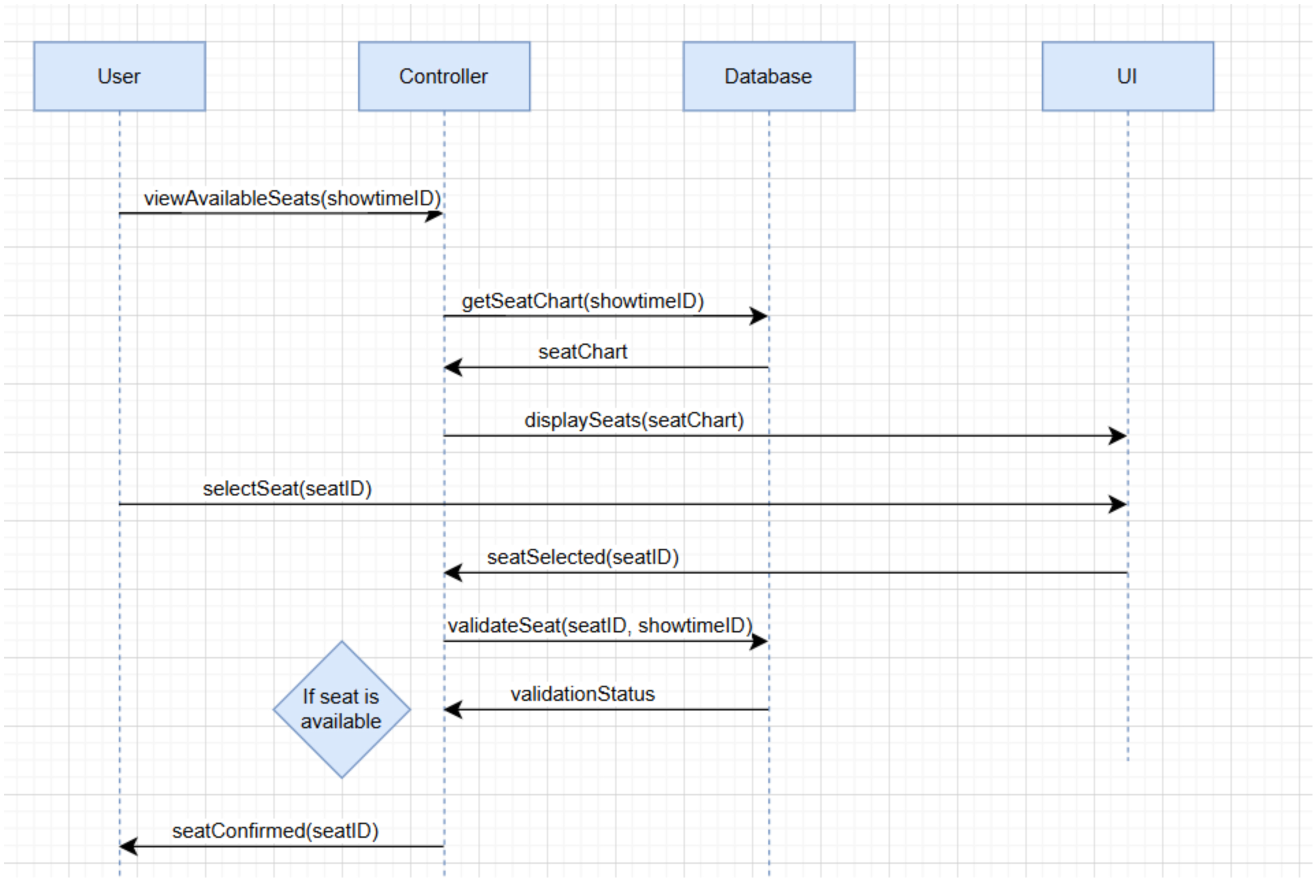


5.2 Viewing Available Seats

- **Participants:**
  1. **User:** Views available seats for a selected showtime.
  2. **Controller:** Manages seat data.
  3. **Database:** Stores seat information.
  4. **UI:** Displays the seat chart.
- **Steps:**
  1. User requests to view available seats for a specific showtime.
  2. Controller retrieves seat information from the database.
  3. Database sends the seat chart to the controller.
  4. Controller displays the seat chart to the user via the UI.
  5. User selects a seat, and the controller validates its availability.

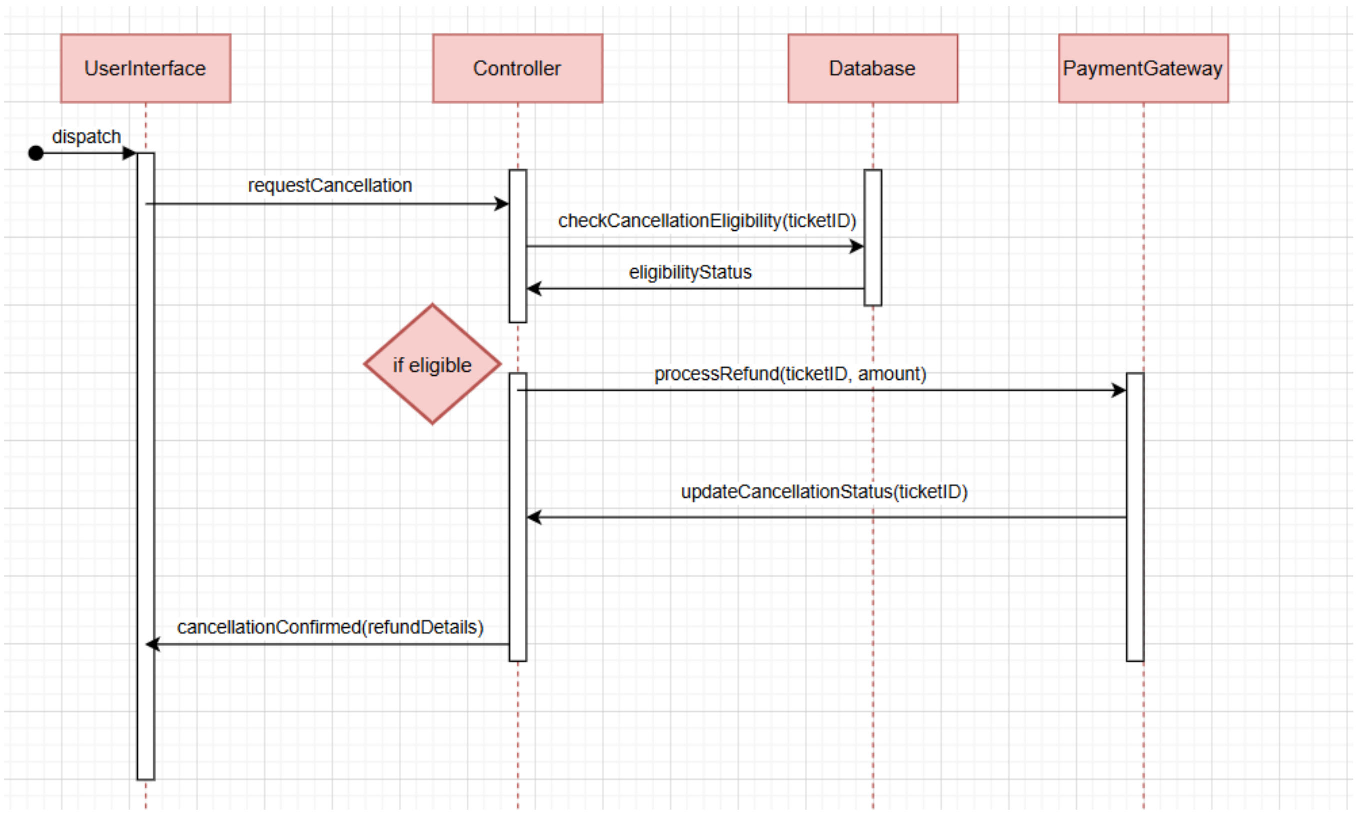
6. If the seat is available, the controller confirms the selection.

- **Visual Reference**: Sequence Diagram 2. Done by Jaskirat.



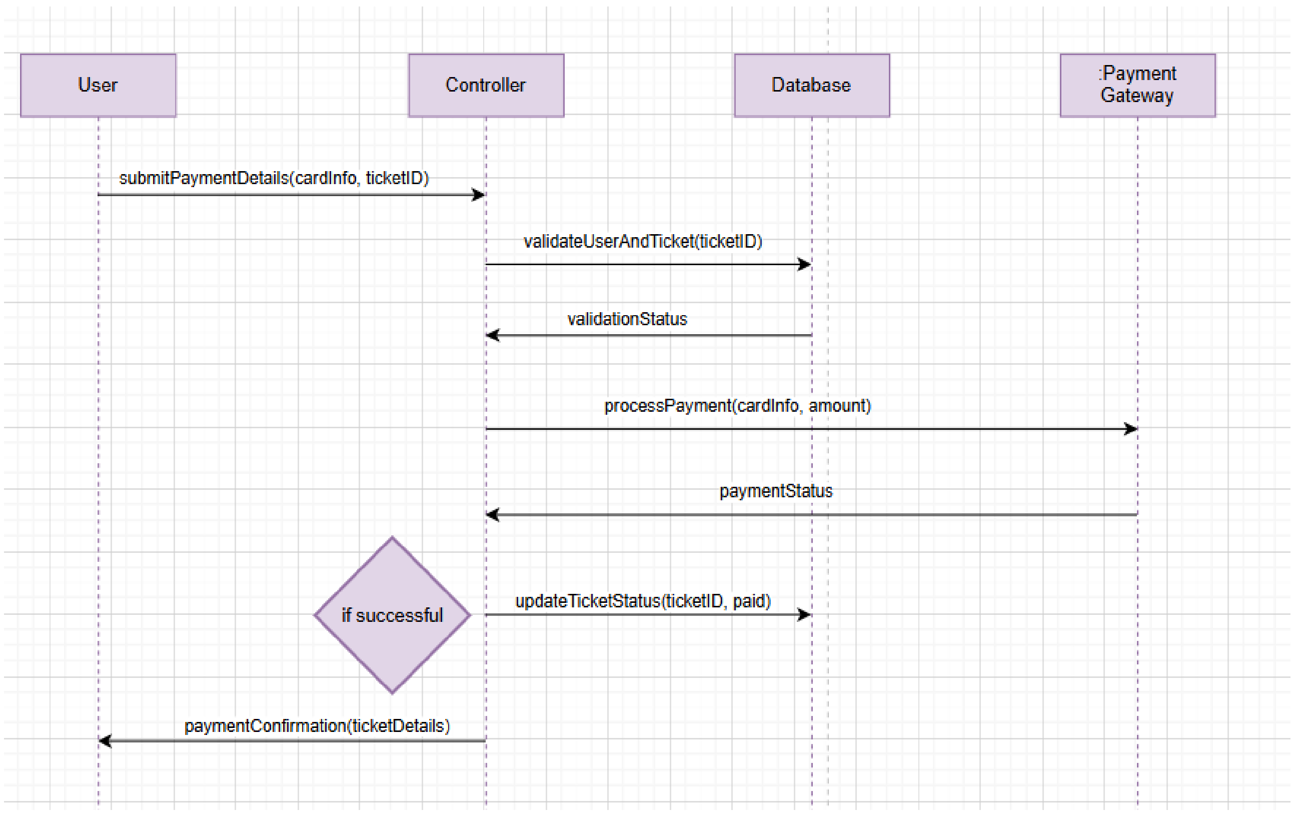
5.3 Cancelling a Ticket

- **Participants**:
  1. **UserInterface**: Initiates the cancellation.
  2. **Controller**: Validates and processes the cancellation request.
  3. **Database**: Updates ticket status.
  4. **Payment Gateway**: Processes refunds if applicable.
- **Steps**:
  1. User requests a ticket cancellation.
  2. Controller checks eligibility for cancellation with the database.
  3. If eligible, the controller processes the refund via the payment gateway.
  4. Controller updates the cancellation status in the database.
  5. User receives confirmation of cancellation and refund details.
- **Visual Reference**: Sequence Diagram 3. Done by Samin.


UserInterface
Controller
Database
PaymentGateway
dispatch
requestCancellation
checkCancellationEligibility(ticketID)
eligibilityStatus
if eligible
processRefund(ticketID, amount)
updateCancellationStatus(ticketID)
cancellationConfirmed(refundDetails)

5.4 Making a Payment

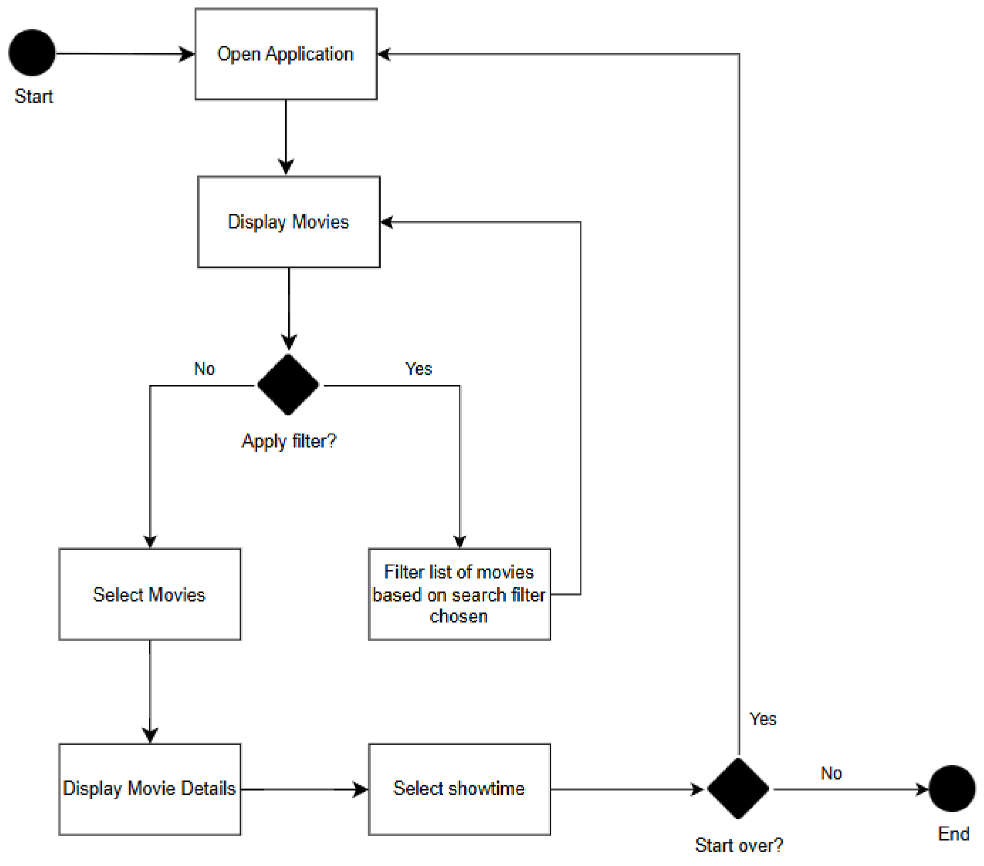
- **Participants:**
  1. **User**: Provides payment details.
  2. **Controller**: Validates and processes the payment.
  3. **Database**: Verifies user and ticket data.
  4. **Payment Gateway**: Handles payment transactions.
- **Steps:**
  1. User submits payment details (card information, ticket ID).
  2. Controller validates the ticket and user information with the database.
  3. If valid, the controller sends the payment details to the payment gateway.
  4. Payment gateway processes the payment and returns the status.
  5. Controller updates the ticket status to "paid" in the database.
  6. User receives payment confirmation.
- **Visual Reference**: Sequence Diagram. Done by Saba.


User
Controller
Database
:Payment Gateway
submitPaymentDetails(cardInfo, ticketID)
validateUserAndTicket(ticketID)
validationStatus
processPayment(cardInfo, amount)
paymentStatus
if successful
updateTicketStatus(ticketID, paid)
paymentConfirmation(ticketDetails)

6. Activity Diagrams

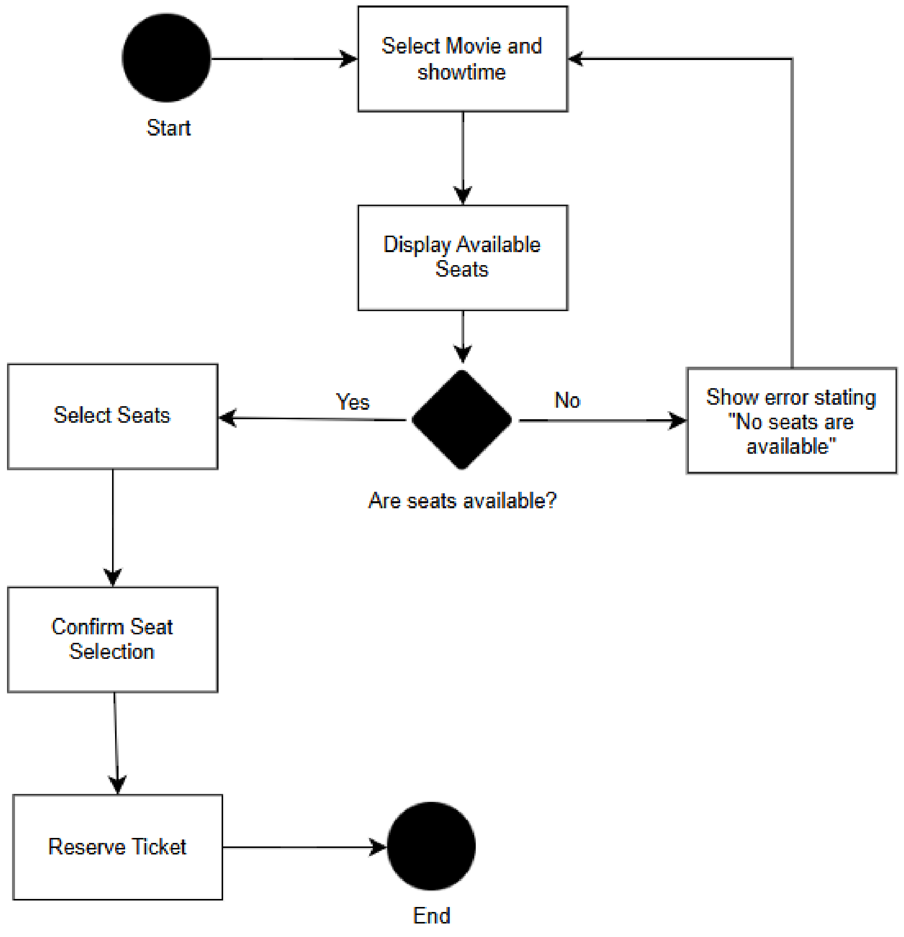
6.1 Browsing Movies

- **Description:**
    1. This diagram outlines the process of browsing movies within the application.
- **Steps:**
    1. The user opens the application.
    2. The system displays a list of available movies.
    3. The user decides whether to apply filters to refine the movie list.
        - If **No**, the user selects a movie directly.
        - If **Yes**, the system filters the list based on the chosen criteria.
    4. The user views the details of the selected movie and chooses a showtime.
    5. The user can either continue to book the showtime or restart the process.
- **Visual Reference:** Activity Diagram 1.



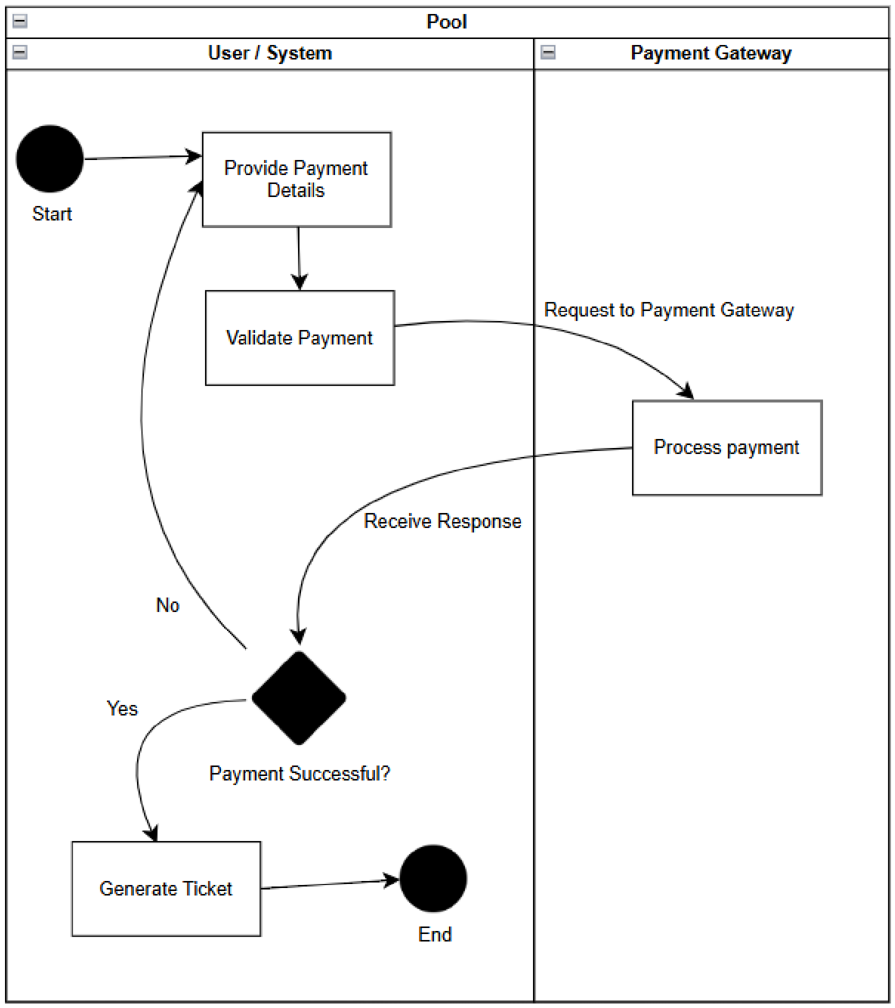
6.2 Viewing and Reserving Seats

- Description:
    1. This diagram depicts the process of selecting and reserving seats for a specific showtime.
- Steps:
    1. The user selects a movie and showtime.
    2. The system displays the available seats.
    3. The user checks whether there are available seats:
        - If **Yes**, the user selects seats and confirms the selection.
        - If **No**, the system displays an error message, and the user returns to seat selection.
    4. Upon confirmation, the ticket is reserved.
- **Visual Reference**: Activity Diagram 2.



6.3 Making a Payment

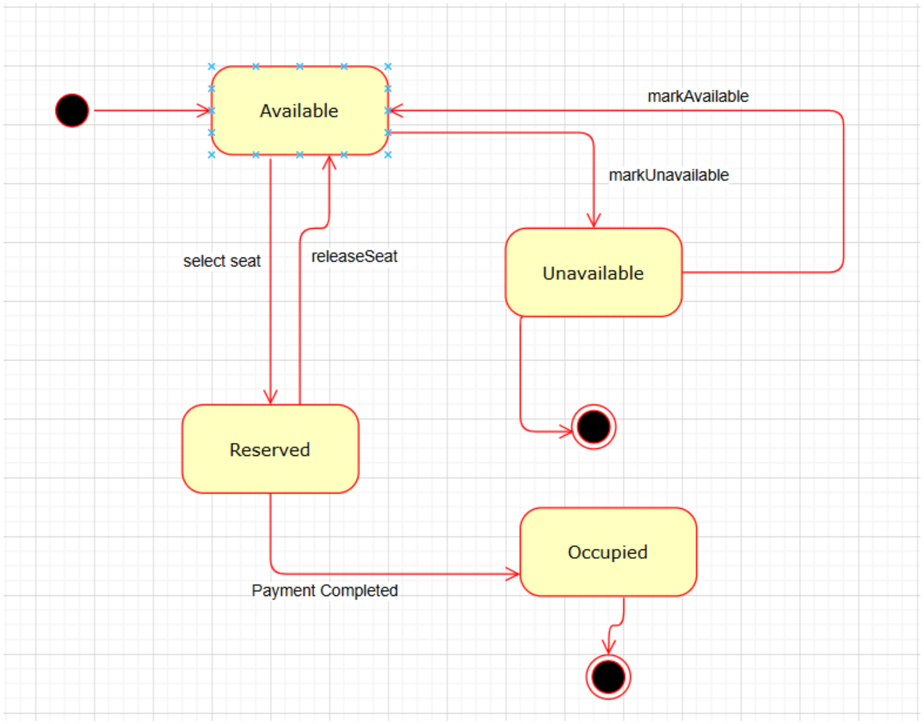
- Description:
    1. This diagram shows the process of completing a payment after selecting seats and tickets.
- Steps:
    1. The user provides payment details.
    2. The system validates the payment information.
    3. The system sends a request to the payment gateway to process the payment.
    4. The payment gateway responds with the status of the transaction:
        - If the payment is successful, the system generates the ticket.
        - If the payment fails, the user is prompted to retry.
- **Visual Reference**: Activity Diagram 3.



7. State Diagrams

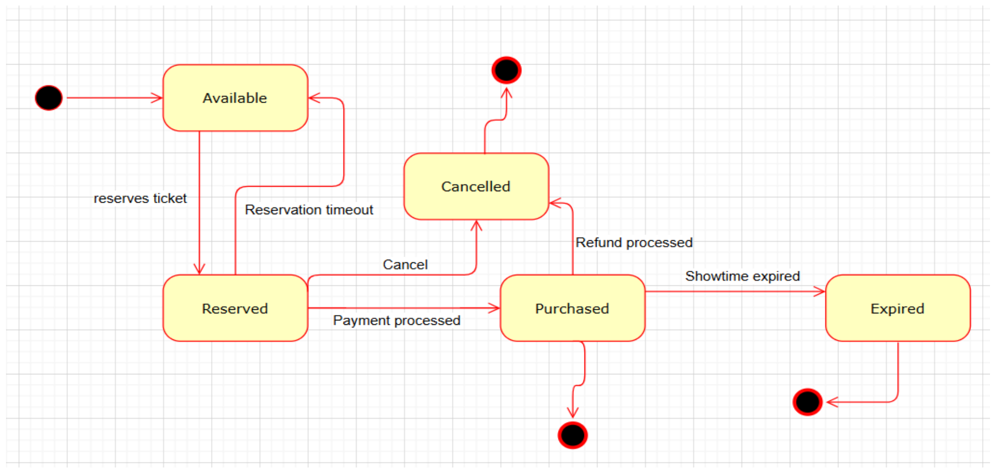
7.1 Seat State Diagram

- States:
    - Available: The seat is open for selection.
    - Reserved: The seat is temporarily held for a user.
    - Unavailable: The seat is blocked or not available for selection.
    - Occupied: The seat is confirmed and marked as taken for the showtime.
- **Visual Reference**: Seat State Diagram. Done by Frank.



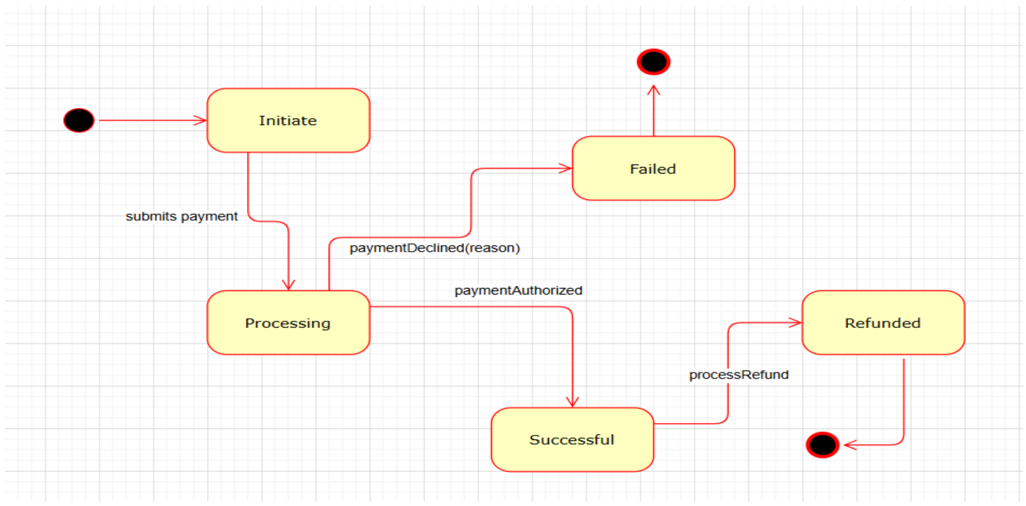
7.2 Ticket State Diagram

- States:
  - Available: The ticket is open for reservation.
  - Reserved: The ticket is temporarily held for a user.
  - Purchased: The ticket is booked and paid for.
  - Cancelled: The ticket reservation is canceled.
  - Expired: The ticket is invalidated after the showtime ends.
- **Visual Reference**: Ticket State Diagram. Done by Jaskirat.



7.3 Payment State Diagram

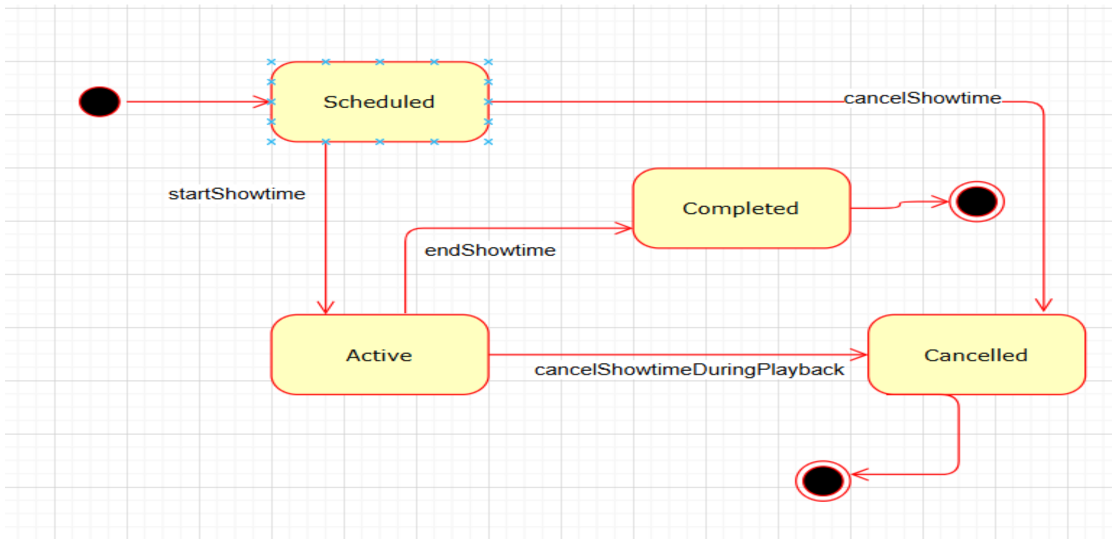
- States:
  - Initiate: The payment process starts.
  - Processing: The system validates the payment details.
  - Successful: The payment is approved and completed.
  - Failed: The payment is declined or encounters an error.
  - Refunded: The payment is reversed.
- **Visual Reference:** Payment State Diagram. Done by Samin.



7.4 Showtime State Diagram

- States:
  - Scheduled: The showtime is created and open for bookings.
  - Active: The showtime is in progress.

- Completed: The showtime has ended.
- Cancelled: The showtime is canceled.
- **Visual Reference:** Showtime State Diagram (attached). Done by Saba.



---

8. Additional Features

8.1 Notification System

The NotificationController allows registered users to receive personalized notifications, such as announcements or reminders. The system supports enabling or disabling notifications.

8.2 Admin Features

The HomeController provides admin functionalities, including system health checks and dashboard access.

8.3 Pre-Booking Limits

The system enforces a limit on pre-booking for registered users, ensuring no more than 10% of total seats are pre-booked per showtime.

---

9. API Endpoints

9.1 User Management

- **Register User:** Allows new users to create an account by providing basic information like name, email, password, and address.
    - http://localhost:8080/users/register
- **Login User:** Authenticates users with their email and password, returning a user ID upon success.
    - http://localhost:8080/users/login
- **View Profile:** Retrieves user profile details using their user ID.
    - http://localhost:8080/users/%7BuserID%7D
- **Update Profile:** Updates user information (currently not working).
    - http://localhost:8080/users/%7BuserID%7D

9.2 Movie Management

- **List Movies:** Retrieves all available movies with basic details like title, genre, and rating.
    - http://localhost:8080/movies/
- **View Movie Details:** Fetches detailed information about a specific movie, including showtimes.
    - http://localhost:8080/movies/%7BmovieID%7D

9.3 Theatre and Showtime Management

- **List Theatres:** Returns a list of theatres with their locations and capacities.
    - http://localhost:8080/theatres/
- **View Showtimes:** Retrieves showtimes for a specific theatre.
    - http://localhost:8080/theatres/%7BtheatreID%7D/showtimes

9.4 Seat Management

- **Get Seats:** Displays available and reserved seats for a specific showtime.
    - http://localhost:8080/showtimes/seats/%7BshowtimeID%7D

9.5 Ticket Management

- **Create Ticket:** Generates a ticket after reservation and payment.
    - http://localhost:8080/tickets/
- **View Ticket:** Retrieves all details of a specific ticket.

        - http://localhost:8080/tickets/%7BticketID%7D
- **Cancel Ticket:** Cancels a booked ticket and processes any applicable refunds.
    - http://localhost:8080/tickets/cancel/%7BticketID%7D

9.6 Payment Management

- **Make Payment:** Processes payments using user-provided card details.
    - http://localhost:8080/payments/

9.7 Notifications

- **Send Notification:** Sends custom notifications to users.
    - http://localhost:8080/notifications/

---

10. Summary

This project demonstrates a robust, scalable design for a movie booking system. The comprehensive use of UML diagrams, including class, sequence, activity, and state diagrams, ensures clear documentation and alignment with project goals.

---

10. References

- Spring Boot Documentation.
- MySQL Database Design Guides.
- Draw.io for UML Diagramming.