Low Level Design – Movie Ticket Booking

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE & ENGINEERING
(ARTIFICIAL INTELLIGENCE & DATA SCIENCE)

Submitted by

K. Bhargava Sai (22PA1A4551)
M. Abhiram (22PA1A4571)
V. Kranthi Kumar(23PA5A4510)

DEPARTMENT OF CSE (ARTIFICIAL INTELLIGENCE AND DATA SCIENCE)

VISHNU INSTITUTE OF TECHNOLOGY
(Approved by AICTE, Affiliated to JNTU Kakinada)
BHIMAVARAM 534 202
2024 2025

Table of Contents

1. Project Overview…….
   - Description of the Movie Ticket Booking System
   - Key features and functionality

2. System Components…….
   - Overview of main system components and their relationships

3. Classes Design……..
   - Show
   - Theatre
   - User
   - Movie
   - Screen
   - Seat
   - SeatLock
   - SeatType
   - BookingStatus

4. Service Layer……
   - Interaction between services and models
   - Key service methods explained

5. Methods Documentation……..

6. Usage Flow…….....
   - Walkthrough of system interactions
   - User flows: Booking process, cancellations, and seat locking mechanisms

7. Code Organization……
   - Package structure
   - Integration between components

8. Conclusion…….

Project Overview

Description of the Movie Ticket Booking System

The Movie Ticket Booking System is a dynamic and scalable platform designed to streamline the entire movie ticketing process for both users and theatre administrators. By leveraging modern software design principles, the system integrates multiple entities like users, movies, theatres, and seats into a unified framework, ensuring a seamless and efficient booking experience.

This system supports real time operations, such as seat selection and locking, to minimize conflicts and maximize user satisfaction. Designed to cater to the needs of both casual users and regular moviegoers, it includes features for exploring available shows, reserving seats, and customizing preferences for an enhanced viewing experience. Administrators can manage showtimes, seating arrangements, and bookings through intuitive interfaces, making the system robust and operationally efficient.

The architecture emphasizes modularity and scalability, allowing for the addition of advanced features such as loyalty rewards, multi theatre management, and payment integrations in the future. With user experience at its core, the system combines speed, reliability, and ease of use to redefine the movie ticketing process.

Key Features and Functionality

1. **User Management**: Enables users to create and manage profiles, distinguishing between customers and administrators.
2. **Movie and Theatre Integration**: Provides detailed listings of movies, theatres, and showtimes for user convenience.
3. **Seat Selection and Locking:** Offers real time seat availability with locking mechanisms to prevent booking conflicts.
4. **Booking Management:** Supports end to end booking workflows, including payments and cancellations.
5. **Customizable Seat Types:** Includes support for various seat categories (e.g., regular, premium) with dynamic pricing.
6. **Scalability:** Designed to handle high volumes of users and transactions efficiently.
7. **Future Enhancements:** Open for additional features such as loyalty programs and integration with payment gateways.

System Components

Overview of Main Components and Their Relationships:

The Movie Ticket Booking System is built around several well-defined components, each designed to encapsulate a specific part of the system's functionality. Below is an explanation of the main system components, their relationships, and code examples to illustrate their interactions.

Key Components:

User:

- Represents the customer using the system.
- Stores user specific information like name, email, and booking history.
- Responsible for initiating bookings and managing their history.

Theatre:

- Represents a physical location containing multiple screens.
- Manages screens and the shows playing on those screens.

Screen:

- Represents a screen in a theatre where movies are shown.
- Contains seats and manages their availability for a given show.

Seat:

- Represents an individual seat in a screen.
- Tracks status (available, booked, locked) and type (VIP, regular, etc.).

Movie:

- Represents a movie, storing metadata like title, duration, and genre.

Show:

- Represents a specific screening of a movie in a particular theatre at a set time.
- Tracks available seats and manages bookings for that screening.

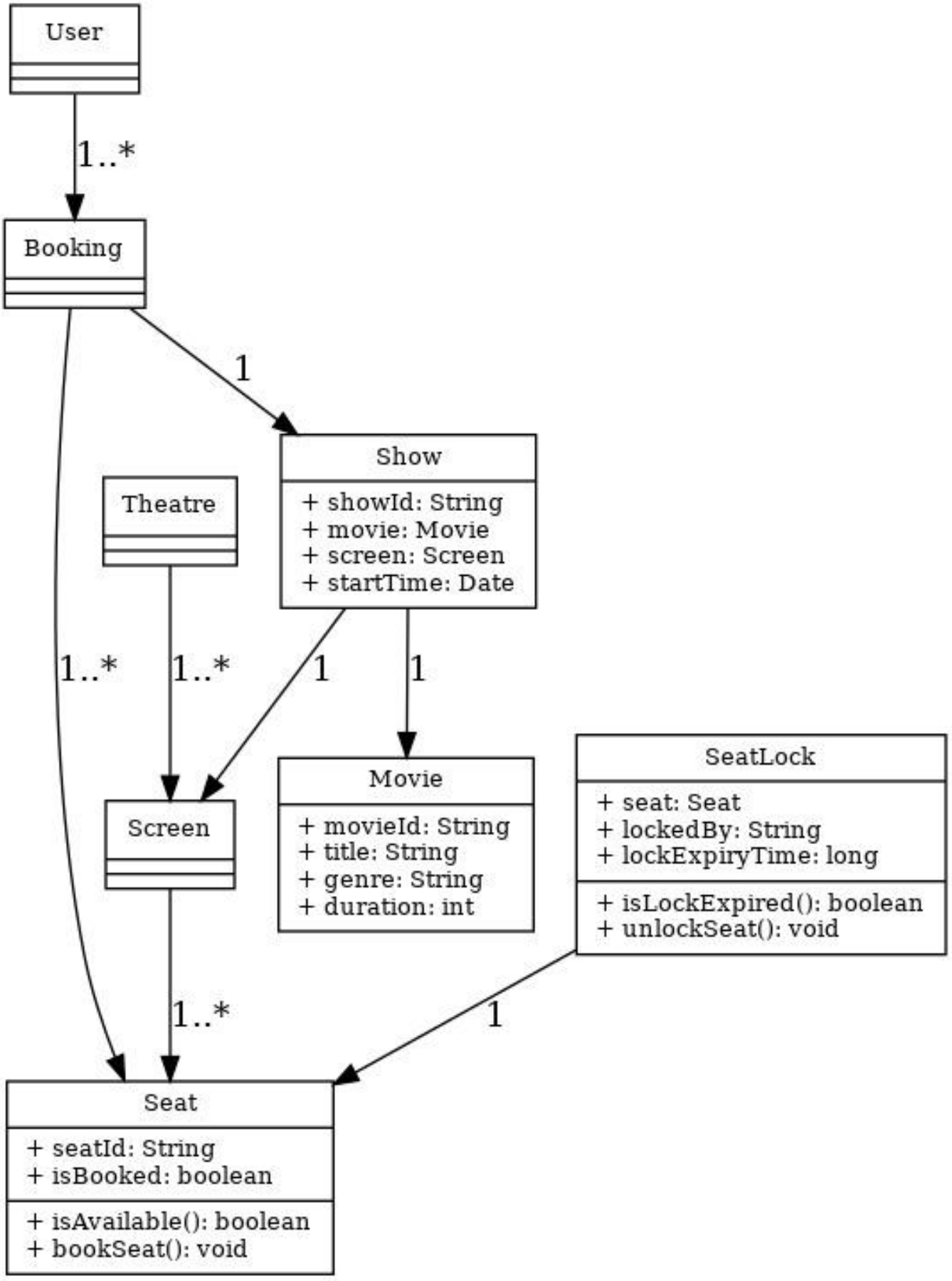
Booking:

- Represents a user’s reservation for a show.
- Stores booking details like user, show, and the list of seats booked.

SeatLock:

- Handles temporary locks on seats to avoid double booking during the checkout process.

UML Diagram representing the relationships among the components



Class Design

1 . Show:

Overview:

- The Show class represents a specific movie screening at a given time in a particular theatre.

Attributes:

- **showId:** Unique identifier for the show.
- **movie**: Reference to the Movie being shown.
- **theatre:** Reference to the Theatre where the show is being hosted.
- **screen:** The Screen assigned for this show.
- **timing**: Date and time for the show.
- **availableSeats:** List of seats available for booking.

Methods:

- **getAvailableSeats ()**: Returns a list of seats that can be booked.
- **reserveSeat(seat)**: Reserves a specific seat for the show.

Usage:

- The Show class is used to manage and retrieve details of specific screenings and interact with available seats for that show.
- Helps tie together movies, screens, and theatres into cohesive units.

2. Theatre:

Overview:

- The Theatre class represents a physical movie theatre.

Attributes:

- **theatreId**: Unique identifier for the theatre.
- **Name**: Name of the theatre.
- **screens**: List of screens in the theatre.

- **Location**: Physical location of the theatre.

Methods:

- **getAvailableScreens()**: Retrieves screens not currently in use.
- **addScreen(screen):** Adds a new screen to the theatre.

Usage:

- This class serves as the parent container for screens and helps in managing the physical space of the system.

3. User

Overview:

- The User class handles user related functionalities and user roles.

Attributes:

- **userId:** Unique identifier for the user.
- **name:** User's name.
- **email:** User's contact email.
- **role:** Role of the user (e.g., Customer, Admin).

Methods:

- **getUserDetails()**: Returns the user's profile information.
- **updateUserDetails(updatedInfo)**: Updates the user's information.

Usage:

- This class centralizes user management, ensuring that roles and user specific data are handled efficiently.

4. Movie

Overview:

- The Movie class represents a film that is available for booking.

Attributes:

- **movieId:** Unique identifier for the movie.

- **title:** Name of the movie.
- **Genre:** Genre of the movie.
- **Duration:** Duration in minutes.
- **Language:** Language of the movie.

Methods:

- getDetails() : Returns details about the movie.

Usage:

- Acts as a data entity to retrieve and store movie specific details in the system.

5. Screen

Overview:

- The Screen class represents individual screens inside a theatre.

Attributes:

- screenId: Unique identifier for the screen.
- Name: Screen name or number.
- Capacity: Total number of seats in the screen.
- Seats: List of seats available in the screen.

Methods:

- **getAvailableSeats():** Fetches unoccupied seats in the screen.
- **assignShow(show):** Assigns a specific show to this screen.

Usage:

- Links theatres and shows while managing seating and availability.

6. Seat

Overview:

- The Seat class represents individual seats in a screen.

Attributes:

- **seatId:** Unique identifier for the seat.
- **row:** Row number of the seat.

- **Number:** Seat number.
- **seatType:** Type of the seat (e.g., Regular, Premium).

Methods:

- **isAvailable():** Checks whether the seat is available.

Usage :

- Forms the building block for seat reservations and ticket bookings.

7. SeatLock

Overview :

- The SeatLock class implements mechanisms to lock seats during booking to avoid conflicts.

Attributes :

- seat The Seat being locked.
- show The Show for which the seat is locked.
- lockedBy The user who locked the seat.
- lockTimeout Time duration for which the seat remains locked.

Methods :

- isLockExpired() Checks whether the lock has expired.
- unlock() Unlocks the seat after expiration or cancellation.

Usage :

- Prevents double bookings by ensuring seat availability during concurrent access.

8. SeatType

Overview :

- The SeatType class categorizes seats into types.

Attributes :

- typeId Unique identifier for the seat type.
- typeName Name of the seat type (e.g., Regular, Premium).
- priceMultiplier Multiplier for the base price.

Methods :

- getPrice(basePrice) Calculates seat price based on its type.

Usage :

- Used for pricing differentiation and customer experience customization.

9. Booking

Overview :

- The Booking class manages the process of booking tickets.

Attributes :

- bookingId Unique identifier for the booking.
- user User making the booking.
- show Show being booked.
- seats List of seats being reserved.
- status Current status of the booking.

Methods :

- confirmBooking() Confirms the booking after successful payment.
- cancelBooking() Cancels the booking and releases locked seats.

Usage :

- Central to the user journey, managing the end to end ticket reservation process.

10. BookingStatus

Overview :

- The BookingStatus class defines the states of a booking.

Attributes :

- statusId Unique identifier for the status.
- statusName Name of the status (e.g., Confirmed, Cancelled).

Methods :

- isFinalStatus() Checks if the booking has reached a terminal state.

Usage :

- Provides clear state management for bookings, enabling efficient processing.

Service Layer

The Service Layer acts as the glue between the business logic (models) and external components like controllers or APIs. It orchestrates operations, handles exceptions, and ensures consistent interactions between different modules while encapsulating complex business logic. Here's a detailed overview of the service components, their responsibilities, and design considerations.

Key Components in the Service Layer

1. ShowService

Overview:

- The ShowService is responsible for managing the lifecycle of a show, from creation to retrieving seat availability.

Key Functionalities:

- Creation of shows, linking movies, screens, and theatres.
- Providing the list of available seats for a particular show.

Design Choices:

- A dictionary is used to map showId to Show objects, enabling fast retrieval and updates.
- Encapsulation of business rules such as preventing duplicate seat bookings ensures the logic is centralized and maintainable.

Tradeoffs:

- Using a dictionary ensures efficiency but requires careful handling of thread safety if the system supports concurrent access.

2. BookingService

Overview:

- This service manages booking operations, seat locks, and transitions between booking statuses.

Key Functionalities:

- Initiating and confirming bookings.
- Handling booking cancellations and releasing locked seats.

Design Choices:

- Bookings are tracked in a dictionary for quick access, while seat reservations use a thread-safe structure like a ConcurrentHashMap to handle multi-user scenarios.
- The service enforces atomic operations to ensure data consistency during seat reservation and booking confirmation.

Tradeoffs:

- Using a thread-safe data structure ensures safety in a concurrent environment but adds slight overhead in operations.

3. SeatLockService

Overview:

- The SeatLockService manages the temporary locking of seats to ensure they are not booked by multiple users simultaneously.

Key Functionalities:

- Locking seats during booking.
- Releasing locks upon booking confirmation or expiration.

Design Choices:

- Locks are stored using a hash map for efficient seat lookup, with expiration tracking implemented using timestamps.
- Seat locks are refreshed periodically to prevent unintended expiration, ensuring a smooth booking experience for users.

Tradeoffs:

- Managing expiration times increases system complexity, but it prevents stale locks and optimizes seat availability.

4. UserService

Overview:

The UserService manages user-related operations, such as registration, authentication, and role management.

Key Functionalities:

- Storing and retrieving user information.
- Differentiating users based on roles (e.g., Admin, Customer).

Design Choices:

- A single service for user management ensures a unified interface for all user-related operations.
- The user role is stored as an attribute within the User model, enabling role-based access without the need for additional structures.

Tradeoffs:

- Centralizing user management makes the system easier to maintain, but it may require optimization when scaling to large user bases

Methods Documentation

This section provides a detailed overview of the methods for each class. It explains their purpose, parameters, return values, and how they interact with other components in the system.

1. Show Methods

Overview:

- The Show class handles the lifecycle of a movie show and its associated attributes like movie, theatre, screen, and time.

- getAvailableSeats()

- Purpose: Returns a list of all available seats for a particular show.
- Parameters: None.
- Returns: A list of available Seat objects.
- Usage: Interacts with Seat and SeatLock classes to determine locked and booked seats.

- addShow()

- Purpose: Adds a new show by linking a movie, screen, and timing.
- Parameters: movieId, screenId, startTime, endTime.
- Returns: Confirmation of show creation.

- getShowDetails()

- Purpose: Retrieves details of a specific show, including movie and theatre information.
- Parameters: showId.
- Returns: Show details as a dictionary.

2. Theatre Methods

Overview:

- The Theatre class manages theatre details and associated screens.

- addScreen()

- Purpose: Adds a new screen to the theatre.
- Parameters: screenId, capacity.
- Returns: Confirmation of screen addition.

- getScreens()

- Purpose: Retrieves all screens associated with the theatre.
- Parameters: None.
- Returns: A list of Screen objects.

- getTheatreDetails()

- Purpose: Provides details about the theatre, including location and available facilities.
- Parameters: theatreId.
- Returns: Theatre information

3. User Methods

Overview:

- The User class handles user attributes and role-based actions.

- createUser()

- Purpose: Registers a new user.
- Parameters: name, email, password, role.
- Returns: User ID of the newly created user.

- getUserDetails()

- Purpose: Retrieves user information.

- Parameters: userId.
- Returns: User details as a dictionary.

- isAdmin()

- Purpose: Checks if the user has an admin role.
- Parameters: None.
- Returns: Boolean indicating admin status

4. Movie Methods

Overview:

- The Movie class contains details about movies being played.

- addMovie()

- Purpose: Adds a new movie to the database.
- Parameters: title, duration, genre, language.
- Returns: Confirmation of movie addition.

- getMovieDetails()

- Purpose: Fetches details of a specific movie.
- Parameters: movieId.
- Returns: Movie attributes such as title, duration, genre.

- listMovies()

- Purpose: Retrieves a list of all movies available.
- Parameters: None.
- Returns: A list of Movie objects.

5. Screen Methods

Overview:

- The Screen class manages screen-specific attributes and operations.

- allocateSeats()

- Purpose: Allocates seats to a show.
- Parameters: seatCount.
- Returns: Confirmation of seat allocation.

- getScreenCapacity()

- Purpose: Retrieves the total capacity of the screen.
- Parameters: screenId.
- Returns: Number of seats available.

6. Seat Methods

Overview:

- The Seat class represents individual seats and their attributes.

- getSeatType()

- Purpose: Returns the type of seat (e.g., VIP, Regular).
- Parameters: None.
- Returns: A SeatType object.

- isAvailable()

- Purpose: Checks if the seat is available for booking.
- Parameters: None.
- Returns: Boolean indicating seat availability.

7. SeatLock Methods

Overview:

- The SeatLock class handles temporary reservations of seats.

- lockSeat()

- Purpose: Locks a seat for a specified duration.

- Parameters: seatId, showId, userId, lockDuration.
- Returns: Confirmation of seat lock.

- isSeatLocked()

- Purpose: Checks if a seat is currently locked.
- Parameters: seatId, showId.
- Returns: Boolean indicating lock status.

- releaseLock()

- Purpose: Releases the lock on a seat.
- Parameters: seatId, showId.
- Returns: Confirmation of lock release.

8. SeatType Methods

Overview:

- The SeatType class categorizes seats based on pricing and features.

- getPrice()

- Purpose: Retrieves the price associated with the seat type.
- Parameters: None.
- Returns: Price as a float.

- getDescription()

- Purpose: Provides a description of the seat type.
- Parameters: None.
- Returns: Description as a string.

9. Booking Methods

Overview:

- The Booking class handles all aspects of seat booking.

- createBooking()

- Purpose: Creates a new booking for a user.
- Parameters: userId, showId, seatIds.
- Returns: Booking confirmation.

- getBookingDetails()

- Purpose: Retrieves details of a specific booking.
- Parameters: bookingId.
- Returns: Booking attributes.

- cancelBooking()

- Purpose: Cancels an existing booking.
- Parameters: bookingId.
- Returns: Confirmation of cancellation.

10. BookingStatus Methods

Overview:

- The BookingStatus class manages different statuses of bookings.

- setStatus()

- Purpose: Updates the status of a booking.
- Parameters: bookingId, status.
- Returns: Confirmation of status update.

- getStatus()

- Purpose: Retrieves the current status of a booking.
- Parameters: bookingId.
- Returns: Current status as a string.

Usage Flow

Walkthrough of System Interactions

The Movie Ticket Booking System facilitates seamless interactions between various components to enable users to browse movies, book tickets, and manage bookings. The flow ensures efficient data handling and quick response times by leveraging well-structured data models and services.

User Flows

1. Booking Process

- **Step 1:** A user selects a movie and its showtime.

  Data Structures Used: HashMap (to store and retrieve movie and show details efficiently).

- **Step 2:** The system fetches the available seats for the selected show.

  Data Structures Used: List (to maintain the seats associated with a screen).

- **Step 3:** The user selects a seat, and the system temporarily locks it.

  Data Structures Used: ConcurrentHashMap (for managing seat locks with thread-safe operations).

- **Step 4:** The booking is confirmed, updating the seat status and generating a booking record.

  Data Structures Used: HashMap (for bookings), Enum (to track BookingStatus).

2. Cancellations

- **Step 1:** A user requests to cancel a booking.

  Data Structures Used: HashMap (to fetch the booking details).

- **Step 2:** The system updates the seat status to make it available for other users.

  Data Structures Used: Enum (to revert the seat's SeatStatus).

3. Seat Locking Mechanisms

Purpose: Prevents race conditions during simultaneous booking attempts.

Process:

- When a user selects a seat, it is locked for a defined duration.
- If the booking is not completed within the lock period, the lock expires, and the seat becomes available.

Data Structures Used:

- ConcurrentHashMap (for seat lock management).

- Timer/Queue (to manage lock expiry).

Emphasis on Data Structures

HashMap/ConcurrentHashMap: Core for managing entities like movies, shows, seats, and bookings. They provide O(1) time complexity for CRUD operations.

List: Used to maintain ordered collections like seats in a screen or shows in a theatre.

Enum: Used for defining constants such as BookingStatus and SeatType, ensuring clear and manageable state transitions.

Queue/Timer: Manages timed operations like seat lock expirations.

By combining these data structures, the system achieves high performance, scalability, and reliability in handling complex user flows.

Code Organization

Code Organization of the Movie Ticket Booking System

The code organization is a crucial aspect of the design, ensuring maintainability, scalability, and readability of the system. The package structure divides the code into logical units based on functionality and domain, with clear responsibilities for each layer. Here's an elaboration of the package structure, why the structure was chosen, and how integration between components happens.

Package Structure

Models

Purpose: Contains the core entities of the system, which represent the real-world objects and their relationships.

Classes Included:

- Show
- Theatre
- User
- Movie
- Screen
- Seat
- SeatLock
- SeatType
- Booking
- BookingStatus

Why Models?:

Models encapsulate the attributes and basic behaviors of the system's core entities. They act as the building blocks of the application. For instance, the Show model defines the properties of a movie show, while the Booking model manages the booking details.

Integration:

Models interact with each other through references or foreign key-like relationships. For example:

Show references a Movie, Theatre, and Screen.

Booking references a User, Show, and Seats.

Services

Purpose: Contains the business logic and orchestrates operations across multiple models.

Classes Included:

- ShowService
- TheatreService
- UserService
- MovieService
- BookingService
- SeatLockService

Why Services?

Services separate the business logic from the models, making the code modular and easier to maintain. For instance, the BookingService handles the booking workflow, ensuring seat availability, locking mechanisms, and payment validation.

Integration:

Services depend on models to retrieve, update, or manipulate data.

For example, the BookingService calls SeatLockService to lock seats and interacts with the Booking model to create a booking entry.

APIs

Purpose: Exposes endpoints to interact with the system, acting as the entry point for external requests.

Classes/Controllers Included:

- ShowController
- TheatreController
- UserController
- MovieController
- BookingController

Why APIs?

APIs abstract the underlying implementation and provide a simple interface for clients (web or mobile). They ensure that only valid operations can be performed by external users.

Integration:

APIs call services to execute the business logic.

Example: The BookingController calls BookingService.createBooking() when a user submits a

booking request.

Providers

Purpose: Manages the external dependencies and utilities like data persistence, caching, or payment processing.

Components Included:

- PaymentProvider
- CacheProvider
- **DatabaseProvider**

Why Providers?

Providers abstract the external dependencies, making it easier to swap implementations (e.g., changing from one database to another). This abstraction enhances flexibility and modularity.

Integration:

Providers are used by services to perform operations.

Example: BookingService uses PaymentProvider to validate payments and DatabaseProvider to save booking details.

Integration Between Components

The integration is achieved through a layered approach, ensuring each layer focuses on its specific responsibility while interacting seamlessly with other layers.

Step-by-Step Workflow Integration Example (Booking Workflow)

API Layer (Controller)

Receives a booking request from the user.

Example: BookingController accepts parameters like showId, seatIds, and userId.

Service Layer

BookingController calls the BookingService to handle the booking process.

BookingService ensures seat availability using ShowService and locks the selected seats using SeatLockService.

Model Layer

BookingService creates a Booking model object to store the booking details.

The SeatLock model is used to temporarily lock the seats for the user.

Provider Layer

BookingService uses PaymentProvider to validate and process payments.
The DatabaseProvider persists the booking details and updates the seat availability.
Data Flow

The booking confirmation is returned to the BookingController, which sends a response to the client.

Design Choices

Separation of Concerns:

Models handle data representation, services manage business logic, and APIs expose system functionality. This separation makes the code modular and easy to debug.

Use of Classes:

Classes like Show, Seat, and Booking encapsulate both attributes and behavior, making it easier to manage real-world entities.

Example: The Show class encapsulates properties like **startTime, endTime**, and methods like **getAvailableSeats().**

Inheritance:

In this LLD, inheritance may be used for extending functionality. For instance, if different User roles **(Admin, Customer)** require different behaviors, they can inherit from a base User class.

Use of Data Structures:

HashMap: Used for fast retrieval of seat locks or show details.

Lists: Used for managing collections like available seats or shows in a theatre.

Queues (Optional): For locking mechanisms where multiple users attempt to book the same seat.

Decoupling Through Providers:

Providers abstract away external dependencies, making the system flexible and easier to test.

Why This Structure?

Maintainability: Each layer has a specific responsibility, reducing interdependencies and making the system easier to maintain and update.

Scalability: The service and provider layers are designed to handle scaling requirements, such as adding new features or integrating with new third-party tools.

Readability: A clear package structure helps developers understand the flow and relationships between components.

This layered design, with clearly defined responsibilities and integration points, ensures the system is robust, scalable, and easy to understand, meeting the requirements of a dynamic movie ticket booking platform.

Conclusion

Conclusion of the Movie Ticket Booking System LLD

The Movie Ticket Booking System LLD is designed to be modular, scalable, and maintainable by leveraging a clear separation of concerns and a well-structured layered architecture. The system comprises:

Core Models:

Encapsulating real-world entities like Show, Theatre, Seat, Booking, and their relationships ensures clarity and simplicity in data representation.

Service Layer:

Business logic is centralized in the service layer, making operations like seat locking, booking creation, and user management efficient and easy to maintain.

API Layer:

Exposing the system functionality through clean, ensures seamless interaction with clients while abstracting implementation complexities.

Provider Layer:

Abstraction of external dependencies (e.g., payment processing, caching, or database operations) ensures flexibility and adaptability to changes.

Design Choices:

The use of classes ensures encapsulation and better management of entities.

Logical grouping into packages like models, services, **apis**, and providers enhances modularity and scalability.

The use of data structures like **HashMaps** for quick lookups and Lists for sequential operations optimizes performance.

Separation of concerns facilitates maintainability and debugging.

Integration:

The system components are tightly integrated while maintaining independence, ensuring smooth workflows like booking a movie ticket, handling seat locks, or managing user accounts.

Key Highlights

Scalability: The layered architecture supports scaling in terms of both users and features, making it adaptable for larger systems.

Maintainability: The decoupled design ensures easy updates and debugging. Each component can be independently developed, tested, and replaced.

Flexibility: Abstraction in providers allows the system to integrate with different external tools and services without impacting the core logic.

In summary, this LLD emphasizes modularity, flexibility, and scalability, ensuring a robust and user-friendly movie ticket booking experience. This design not only meets current requirements but also leaves room for future enhancements, such as adding dynamic pricing, loyalty programs, or integration with external ticket aggregators.