Movie Ticketing System

Version 1.0

Software Architecture Document

June/2024

GROUP 11

Revision History

NOTE: *The revision history cycle begins once changes or enhancements are requested after the initial version of the Software Architecture Document has been completed.*

Date	Version	Description	Author
06/17/2024	1.0	<details>	Ben Coon
			LUCINE BABIKIAN
			RAUL MONTES

Table of Contents

Software Architecture Document

1. Introduction

1.1. Scope

This document provides the Software Architecture Specification for the Movie Ticketing System, which supports functionalities such as browsing movies, booking tickets, and managing bookings.

1.2. Definitions, Acronyms, and Abbreviations

- SDS: Software Design Specification
- SQL: Structured Query Language
- NoSQL: Not Only SQL
- API: Application Programming Interface
- RBAC: Role-Based Access Control

1.3. References

- "Software Architecture: The '4+1' View Model" by Philippe Kruchten
- IBM Rational Unified Process
- "Entity Relationship Diagrams" whitepaper: link here

1.4. Overview

This document describes the architecture of the Movie Ticketing System, including its goals, use-case view, logical view, process view, deployment view, implementation view, and data view.

2. Architectural Representation

The software architecture for the Movie Ticketing System is illustrated using the "4 + 1 architecture view model" proposed by Philippe Kruchten. This model provides five distinct perspectives on the system's architecture: Use-Case, Logical, Process, Deployment, and Implementation views. Each viewpoint describes various aspects of the system's models, stakeholders, and their concerns. Additionally, a Data View is included to represent the persistent data objects within the system. The following sections provide detailed descriptions of each view.

Use-Case View

The Use-Case View captures the functional and non-functional significant architectural requirements (SARs). Use cases serve as the glue that integrates all other views. This view includes:

- Actors: Entities interacting with the system like customers and administrators.
- Use Cases: Specific interactions or functions performed by the system like booking tickets and canceling bookings.
- Classes: Core classes that participate in the use cases.
- Collaborations: Interactions between classes and actors to fulfill the use cases.

Logical View

The Logical View focuses on key design mechanisms and architecturally significant elements, organizing them into subsystems and layers. This view includes:

- Objects: Instances of classes that are used to represent data.
- Classes: Architecturally significant classes, including their responsibilities, operations, and attributes.
- Collaborations: Interactions between objects and classes.
- Interactions: Communication between objects and subsystems.

- Packages: Groupings of related classes.
- Subsystems: Higher-level components composed of multiple packages or classes.

Implementation View

The Implementation View describes key implementation elements such as code artifacts, executables, and modules. This view includes:

- Modules: Self-contained units of software with specific functionalities.
- Subsystems: Groupings of modules that provide broader functionalities.
- Interfaces: Defined ways for modules and subsystems to interact with each other.

Process View

The Process View outlines the system's decomposition into lightweight and heavyweight processes, focusing on their communication and interactions. This view includes:

- Tasks, Threads, Processes: Units of execution within the system.
- Interactions: Communication methods between processes like message passing and interrupts.

Deployment View

The Deployment View describes the physical network configurations on which the software runs, including the mapping of processes to physical nodes. This view includes:

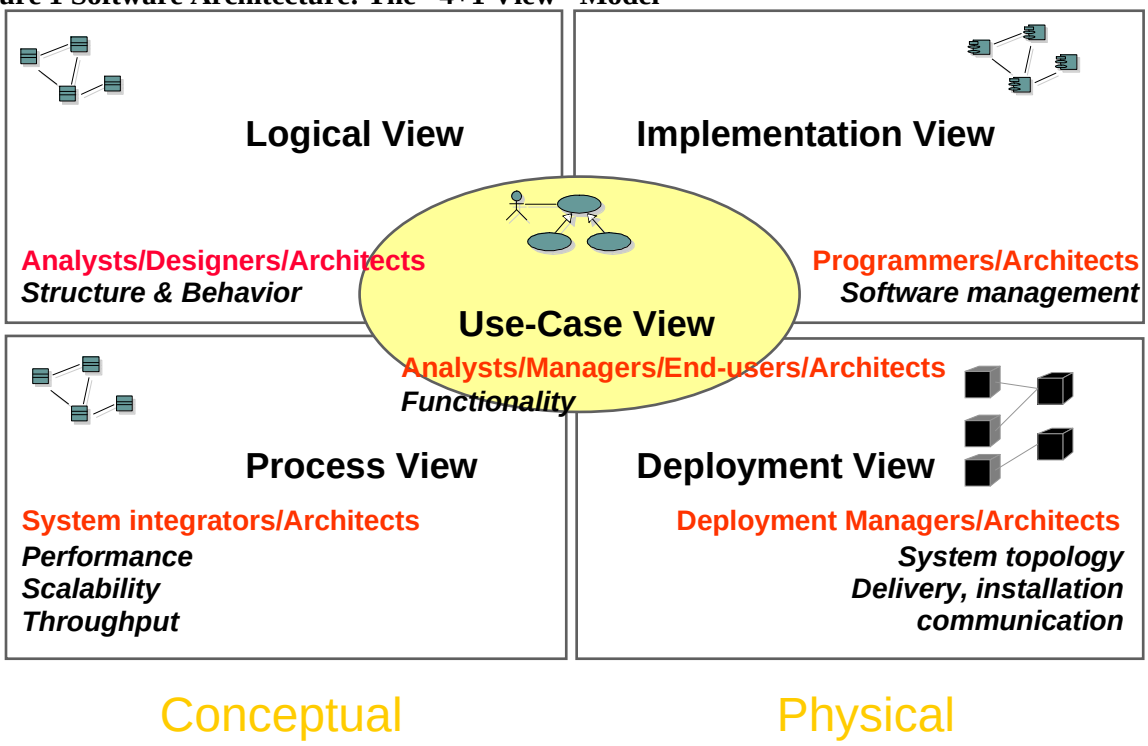
- Nodes: Physical or virtual machines executing the software.
- Modules: Software components deployed on the nodes.

Data View

The Data View describes the logical data model of the system and includes an Entity Relationship Diagram (ERD). This view focuses on:

- Logical Data Model: Representation of data structures.
- Entity Relationship Diagram (ERD): Diagram depicting relationships between data entities.

Figure 1 Software Architecture: The "4+1 View" Model[1]



3. Architectural Goals and Constraints

The Movie Ticketing System's architecture is influenced by several key software requirements and objectives:

- Scalability: The system must handle high volumes of concurrent users, especially during peak times like new movie releases.
- Security and Privacy: User data, including payment information and personal details, must be protected through robust security measures.
- Reliability: The system must ensure high availability and minimal downtime to provide consistent access for users.
- Portability: The system should be deployable on various platforms and devices, including web, mobile, and kiosk interfaces.
- Maintainability and Reusability: The design should allow for easy updates and modifications, with reusable components to reduce development time for new features.
- Integration with Third-Party Services: The architecture must support integration with external payment gateways, movie databases, and notification services.
- Compliance: Adherence to industry standards and regulations, such as PCI-DSS for payment processing.

4. Use-Case View

The Use-Case View of the Movie Ticketing System outlines the primary functional and non-functional requirements from an architectural standpoint. This view serves as a pivotal component in defining the system's behaviors and interactions with its actors.

Actors

The Movie Ticketing System involves two primary actors:

- Customer: The end-user who interacts with the system to browse movies, book tickets, and manage bookings.
- Administrator: Responsible for managing movie listings, theater details, and user accounts within the system.

Use Cases

Several core use cases define the functionality of the Movie Ticketing System:

- Browse Movies: This use case enables customers to browse available movies, sorted by genre, release date, and other filters. Customers must be logged in to access detailed movie information.
- Book Tickets: Customers can select a movie, theater, showtime, and seats to book tickets. This use case ensures that tickets are reserved and customers receive confirmation.
- Manage Bookings: Customers have the ability to view and manage their booked tickets, including options to cancel or modify bookings. This functionality requires customers to be logged in and have active bookings.
- Manage Movies: Administrators can add, update, or remove movies from the system. This use case ensures that the movie database is kept current with new entries or changes.
- Manage Theaters: Administrators can manage theater details such as location, seating arrangement, and amenities. This use case enables administrators to update theater information or add new theaters to the system.
- Manage Users: Administrators can manage user accounts, including registration, authentication, and permissions. This use case provides administrators with tools to update user information or add new users to the system.

Classes

Key classes participating in the use cases include:

- Customer: Represents the end-user interacting with the system.
- Administrator: Manages movie listings, theater details, and user accounts.
- Movie: Represents the details and attributes of movies available for booking.
- Theater: Represents the details and attributes of theaters where movies are screened.
- Booking: Represents the details and attributes of customer bookings.
- User: Represents registered users interacting with the system.

Collaborations

The interactions between actors and classes are crucial for fulfilling the use cases:

- Customers interact with the Movie class to browse available movies and with the Booking class to book tickets.
- Administrators interact with the Movie class to manage movie listings and with the Theater class to manage theater details.

5.Logical View

This section presents the architecturally significant parts of the design model of the Movie Ticketing System, including its decomposition into subsystems and packages. Each significant package is detailed, along with its contained classes, their responsibilities, relationships, operations, and attributes.

5.1. Overview

The design model of the Movie Ticketing System is organized into several architecturally significant packages and layers, reflecting the system's functionality and structural hierarchy. These packages are designed to encapsulate related functionalities and components, promoting modularity and ease of maintenance.

5.2. Architecturally Significant Design Packages

5.2.1. User Management Package

This package manages user-related functionalities, including registration, authentication, and authorization within the system.

UserManager:

- Description: Manages user accounts and their permissions.
- Responsibilities: Registration, authentication, and authorization of users.
- Operations: registerUser(), authenticateUser(), authorizeUser()
- Attributes: userId, username, password

User:

- Description: Represents a registered user of the system.
- Responsibilities: Stores user profile information.
- Attributes: userId, username, email, role

5.2.2. Movie Management Package

This package handles movie-related functionalities, including listing movies, managing movie details, and categorizing movies by genre.

MovieManager:

- Description: Manages the listing and categorization of movies.
- Responsibilities: Add, update, delete movies; retrieve movie details.
- Operations: addMovie(), updateMovie(), deleteMovie(), getMovieById()
- Attributes: movieId, title, genre, duration

Movie:

- Description: Represents a movie available for booking.
- Attributes: movieId, title, genre, duration

5.2.3. Booking Management Package

This package handles the booking and management of movie tickets within the system.

BookingManager:

- Description: Manages the booking of tickets for movies.
- Responsibilities: Reserve seats, confirm bookings, and manage booking details.
- Operations: reserveSeats(), confirmBooking(), cancelBooking()
- Attributes: bookingId, customerId, movieId, showtime, seats

Booking:

- Description: Represents a booking made by a customer.
- Attributes: bookingId, customerId, movieId, showtime, seats

6. Process View

The Process View of the Movie Ticketing System describes its decomposition into lightweight processes (single threads of control) and heavyweight processes (groupings of lightweight processes). It organizes the section by groups of processes that communicate or interact, detailing the main modes of communication between processes.

6.1. Lightweight Processes

The Movie Ticketing System employs lightweight processes to handle concurrent operations within the system. Each lightweight process represents a single thread of control, focusing on specific tasks and interactions.

6.1.1. User Interaction Process

Handles user interactions, such as browsing movies, selecting showtimes, and booking tickets.

Mode of Communication:

- Message Passing: Between user interface components and backend services.
- Synchronous and Asynchronous Operations: For responsive user experience.

6.1.2. Booking Management Process

Manages the booking and reservation of movie tickets.

Mode of Communication:

- Message Passing: Between booking components, user interface, and database services.
- Synchronous Operations: For confirming and updating booking details.

6.2. Heavyweight Processes

Heavyweight processes in the Movie Ticketing System are groupings of lightweight processes, focusing on broader functionalities and complex interactions.

6.2.1. Payment Processing Service

Handles payment transactions for confirmed bookings.
Sub-Processes:

- Payment Verification: Verifies payment details and processes transactions.
- Transaction Logging: Records payment transactions for auditing purposes.

Mode of Communication:

- Message Passing: Between payment gateway services, booking management, and user notification components.
- Asynchronous Operations: For handling payment responses and transaction logging.

6.2.2. Notification Service

Sends notifications to users regarding booking confirmations and updates.
Sub-Processes:

- Notification Composition: Generates notification messages based on booking events.
- Delivery Service: Sends notifications via email or SMS.

Mode of Communication:

- Message Passing: Between booking management, user interface, and external notification APIs.
- Asynchronous Operations: Ensures timely delivery and updates to users.

6.3. Interaction Modes

The Movie Ticketing System uses several modes of interaction between processes:

- Message Passing: Used extensively for communication between components, ensuring modularity and loose coupling.
- Synchronous Operations: Ensures immediate responses for critical user interactions, such as booking confirmations.
- Asynchronous Operations: Handles background tasks, such as notification delivery and transaction logging, without blocking user interactions.

7. Deployment View

The Deployment View describes the physical network configurations where the Movie Ticketing System runs. Configuration A includes a Web Application Server Cluster with Node 1 hosting the web and application servers and Node 2 hosting the database server, interconnected via LAN and Point-to-Point connections. Configuration B consists of Node 3 for the Payment Gateway and Node 4 for the Notification Service, interconnected via Point-to-Point connections. Processes such as User Interaction and Booking Management are mapped to these configurations.

8. Implementation View

This section describes the overall structure of the implementation model, the decomposition of the software into layers and subsystems, and any architecturally significant components.

8.1. Overview

The Movie Ticketing System is structured into several layers, each serving a specific purpose and governed by clear inclusion rules and boundaries to ensure separation of concerns and maintainability. The layers include:

- Presentation Layer: This layer handles user interaction and includes components for the user interface and application presentation logic.
- Application Layer: The core of the system, this layer manages business logic and workflows, including use case implementations and service orchestration.
- Domain Layer: Represents the business objects and entities, encapsulating the business rules and logic. It includes entities, value objects, and domain services.
- Infrastructure Layer: Provides services such as data access, external integrations, and communication with external systems. This layer includes repositories, external APIs, and other infrastructure services.

8.2. Layers

Presentation Layer

The Presentation Layer is responsible for managing user interactions and includes the following subsystems:

- User Interface: Provides interfaces for user interaction, including web and mobile interfaces.
- Controllers: Orchestrates user input and invokes appropriate actions.

Application Layer

The Application Layer encapsulates the business logic and includes the following subsystems:

- Use Case Implementations: Implements system use cases and workflows.
- Service Layer: Provides a set of services that application clients can invoke.

Domain Layer

The Domain Layer defines the business objects and entities and includes the following subsystems:

- Entities: Represents business objects and domain entities.
- Value Objects: Immutable objects that are distinguished by their state rather than their identity.
- Domain Services: Implements the business logic that does not naturally fit within the entities or value objects.

Infrastructure Layer

The Infrastructure Layer provides services to the other layers and includes the following subsystems:

- Data Access: Provides access to persistent storage.
- External Services: Integrates with external systems and services.
- Utilities: Provides utility functions used across the application.

9. Data View

This section provides a description of the Movie Ticketing System's persistent data storage perspective. The system's data architecture includes both a logical and physical data model. The logical data model incorporates design patterns to manage entities such as movies, users, bookings, and payments, ensuring efficient data organization and retrieval. The physical data model adopts a normalized design approach to maintain data integrity and reduce redundancy. Database views are used to simplify complex queries and provide tailored data perspectives for different system components. The system employs a transaction strategy that ensures atomicity, consistency, isolation, and durability (ACID properties) for critical operations. Data distribution across multiple nodes supports scalability and fault tolerance. Concurrency control mechanisms manage simultaneous access to data to prevent inconsistencies. The system ensures data quality through validation rules and constraints, controls redundancy by using unique keys and indexes, and integrates authoritative sources for accurate data. Mechanisms for data access by external systems include RESTful APIs and secure database connections, adhering to industry standards for data exchange and security.

10. Size and Performance

This section describes the major size and performance characteristics of the Movie Ticketing System that impact its architecture. The system is designed to handle a large volume of concurrent users during peak times, such as new movie releases, ensuring scalability. Performance constraints include response times for critical operations like ticket booking and payment processing, aiming for sub-second response times to enhance user experience. The system is optimized for efficient memory and CPU usage to support its deployment across various platforms, including web, mobile, and kiosk interfaces. Performance testing ensures that the system meets these constraints under load, maintaining high availability and reliability.

11. Quality

The software architecture of the Movie Ticketing System significantly contributes to several capabilities beyond its core functionality, including extensibility, reliability, and portability. Special emphasis is placed on ensuring safety, security, and privacy implications.

11.1. Applicability to Technical Reference Model/Standards Profile (TRM/SP)

The Movie Ticketing System adheres to the VA TRM/SP, ensuring that new system development and selection conform to approved standards and rules. This adherence enables the system to promote interoperability, portability, and adaptability within systems, thereby enhancing quality assurance and enabling the VA to leverage current technology effectively.

11.2. Applicability to Common Services Architecture

The Movie Ticketing System aligns with the defined Common Services Architecture specified by the VA EA and the HealtheVet Logical Model. It employs an n-tier architecture and a common, shared services approach, maximizing standardization and reuse while minimizing maintenance impacts, technology dependencies, and database dependence. The design ensures a clear separation between application logic, user interface, authorization support, and data storage. The system is divided into areas of responsibility implemented as service-based components, and it delegates processing to external common services where applicable.

11.3. Applicability to HSD&D Core Specifications for Rehosting Initiatives

The Movie Ticketing System adheres to the HSD&D Core Specifications for Rehosting Initiatives, ensuring compliance with HSD&D-approved architectural and integration standards and policies. These standards, strategies, and guidelines provide a fundamental framework of expectations for all systems enabled by HSD&D, promoting established VA standards through preferred implementation specifics. This approach allows HSD&D to offer a common solution to comply with all architectural goals, ensuring consistency and efficiency across the VA.

12. Architectural Mechanism

The Movie Ticketing System's design supports anticipated system load, growth, availability, concurrency, and distribution requirements, including server, workstation, storage, bandwidth, and middleware considerations.

12.1. Analysis Mechanisms

- Persistency: The system ensures persistency using object-oriented database management systems (OODMS) for new data and relational database management systems (RDBMS) for data from legacy systems.

- Distribution: Distribution across system nodes is achieved using Remote Method Invocation (RMI) in Java.

- Security: Access control mechanisms ensure security, including components like Secure.java and UserContextRemoteObject.

- Legacy Interface: A legacy interface allows access to existing systems with their current interfaces.

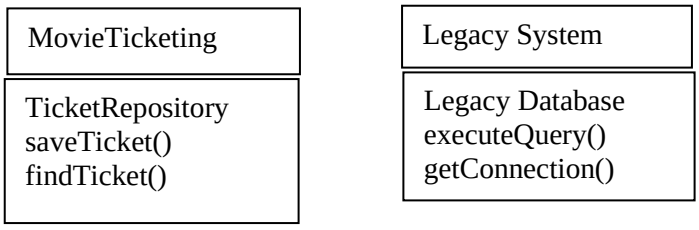
12.2. Analysis-to-Design-to-Implementation Mechanisms Map

Analysis Mechanisms	Design Mechanisms	Implementation Mechanisms
Persistency	OODMS (new data)	ObjectStore
Persistency	RDBMS (data from legacy database)	JDBC to Ingres
Distribution	Remote Method Invocation (RMI)	Java 1.1 from Sun
Security	Access Control	Reverse Engineered Secure.java and UserContextRemoteObject components
Legacy Interface	Interface Proxy	Existing Interface Wrapper

12.3. Implementation Mechanisms

12.3.1. Persistency

12.3.1.1. Static View



Class Descriptions

MovieTicketing

TicketRepository

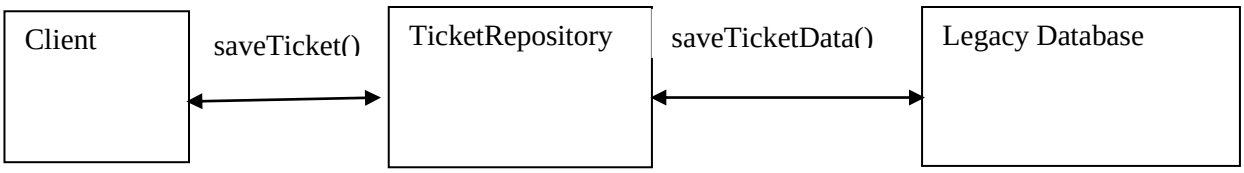
- Description: Manages persistence for movie tickets.
- Responsibilities: Save and retrieve tickets.

Legacy System

Legacy Database

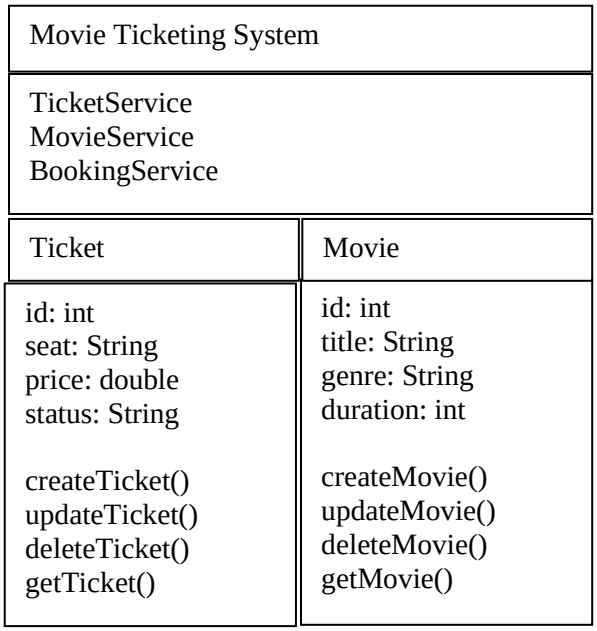
- Description: Legacy database system storing historical ticket data.
- Responsibilities: Execute SQL queries, manage database connections.

12.3.1.2. Dynamic View



12.3.2. Distribution

12.3.2.1. Static View

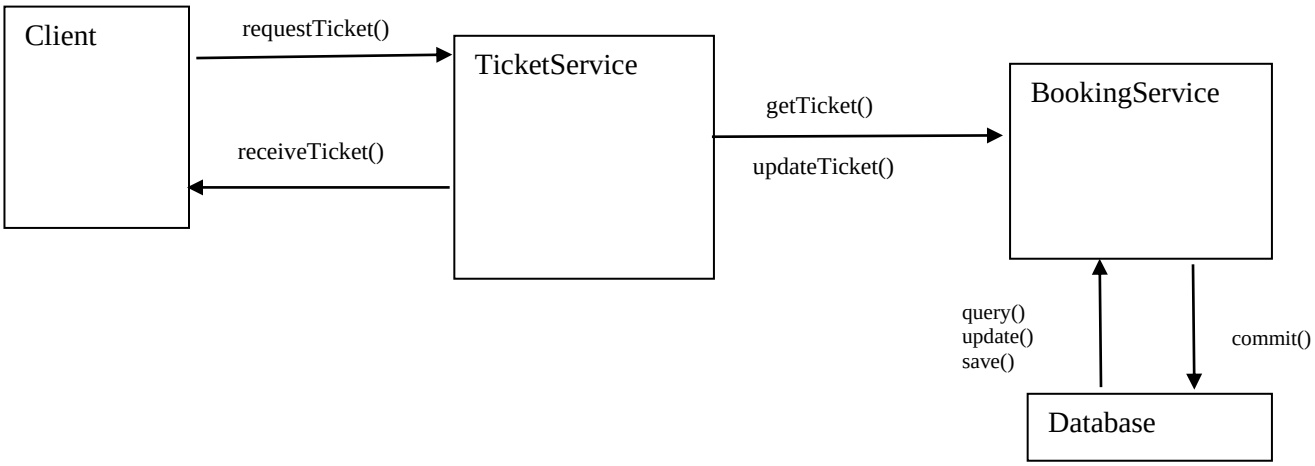


Class Descriptions

- TicketService, MovieService, and BookingService: These are components responsible for managing tickets, movies, and bookings respectively.
- Database: Represents the database where ticket, movie, and booking information is persisted.

- Client: Initiates requests for ticket booking.
- Client -> TicketService -> MovieService -> BookingService -> Database: Sequence of interactions for booking a ticket.

12.3.2.2. Dynamic View



12.3.3. Security

12.3.3.1. Static View

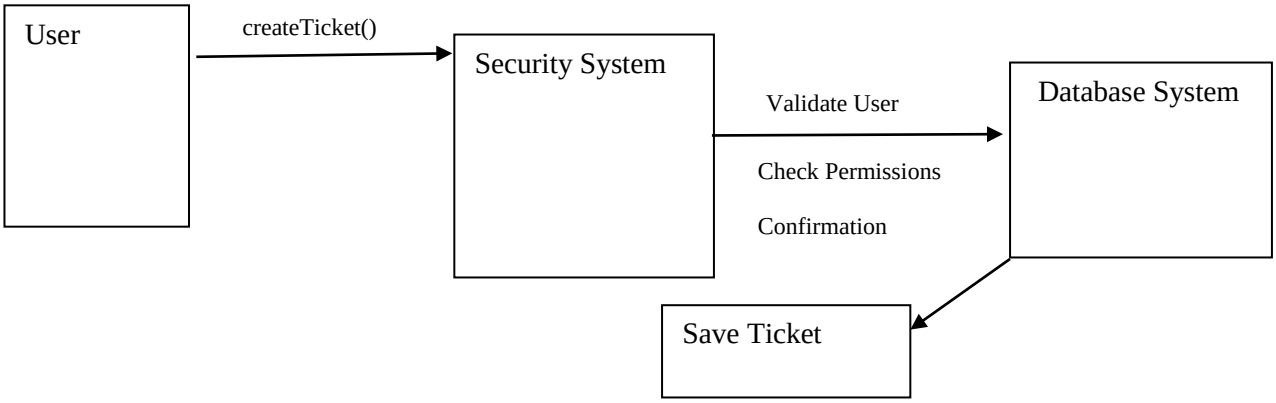
Ticket	Movie
id: int seat: String price: double status: String createTicket() updateTicket() deleteTicket() getTicket()	id: int title: String genre: String duration: int createMovie() updateMovie() deleteMovie() getMovie()

Class Descriptions

The Ticket class represents a ticket in the Movie Ticketing System. It has attributes including an integer ID, a seat designation stored as a string, a price stored as a double, and a status stored as a string. This class provides methods to create, update, delete, and retrieve ticket information.

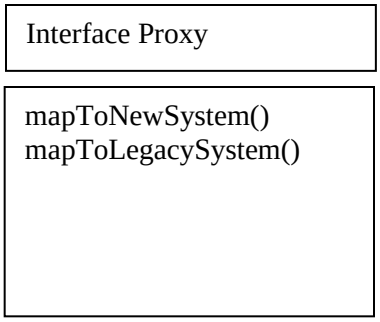
The Movie class represents a movie in the Movie Ticketing System. It has attributes including an integer ID, a title stored as a string, a genre stored as a string, and a duration stored as an integer. This class provides methods to create, update, delete, and retrieve movie information.

12.3.3.2. Dynamic View



12.3.4. Legacy Interface

12.3.4.1. Static View



Class Descriptions

Interface Proxy:

Acts as an intermediary between the Movie Ticketing System and existing legacy systems.

Methods:

- mapToNewSystem(): Maps data from the legacy system to the new system format.
- mapToLegacySystem(): Maps data from the new system to the legacy system format.

12.3.4.2. Dynamic View