Distributed System Design
COMP 6231

Assignment 2
Distributed Movie Ticket Booking System using web services

Submitted by - Apoorva Sharma
40256283

Introduction: Distributed Movie Ticket Booking System using web services

The distributed movie ticketing system is a software application that allows customers to purchase movie tickets online from a platform and allows administrators to manage movie theater operations efficiently and effectively.

- Customers
    - Customers can browse available movies and select the desired movie and theater
    - Customers can book the tickets, view their booking history, cancel booked movies.
- Admin
    - Admins manage the movie shows.
    - Admins can add different movie slots, remove movie slots and can also view the available movie shows

Overall, this application provides a convenient experience for customers and a comprehensive management system for administrators.

Technologies used in the project

WEB SERVICES

Web services are a type of software system designed to support interoperable machine-to-machine interaction over a network. They are often utilized in distributed systems where different software applications need to communicate with each other.

SOAP is a protocol used to exchange structured information between applications. It is a messaging protocol that uses XML for message exchange, and it is often used in enterprise applications. In order to implement web services using SOAP, several components are required. These components include a service provider, a service requester, a message, and a message exchange pattern.

The service provider is responsible for publishing the web service and making it available for use. The service requester is the client that consumes the web service. The message is the data that is passed between the service provider and the service requester. The message exchange pattern describes the sequence of message exchanges between the service provider and the service requester.

When implementing web services using SOAP, the service provider first creates a WSDL (Web Services Description Language) file that describes the service. The WSDL file specifies the operations supported by the service, the format of the messages exchanged, and the location of the service.

The service requester then uses the WSDL file to generate client code that can be used to consume the web service. The client code includes methods that correspond to the operations defined in the WSDL file.

When the service requester sends a request to the service provider, it includes a SOAP message that conforms to the format specified in the WSDL file. The service provider processes the request and sends a response back to the service requester.

In conclusion, SOAP-based web services use XML-based messages to exchange data between applications. Implementing web services using SOAP requires several components, including a service provider, a service requester, a message, and a message exchange pattern.

Users of the Application

The application's users are separated into two categories, Admins and Customers, with each group having distinct functionalities based on the tasks they are able to perform within the application.

Admin

The administrator has the capability to control the movie show timings for their specific servers. The administrator can utilize the following functions to manage bookings within their city's theater.

- **Add movie slots (*addMovieSlots*(*movieID*, *movieName*, *bookingCapacity*)):** Admins can add date, time and location – wise movie slots that customers can reserve by providing the movie-id and movie-name for which they want to add slots, and define their booking capacity.

- **Remove Movie Slots (removeMovieSlots(movieID, movieName)):** Admins can remove the existing movie slots by providing movie-id and movie-name that they want to remove the slots for.

- **Checking Movie Slot Availability(listMovieShowsAvailability(movieName)):** Admins can check movie show availability across all the theater servers by entering the movie-name that they want to check availability for.

Customer

Customers use the application to book and manage their movie ticket reservations, with each customer having the ability to handle their own bookings through provided functions.

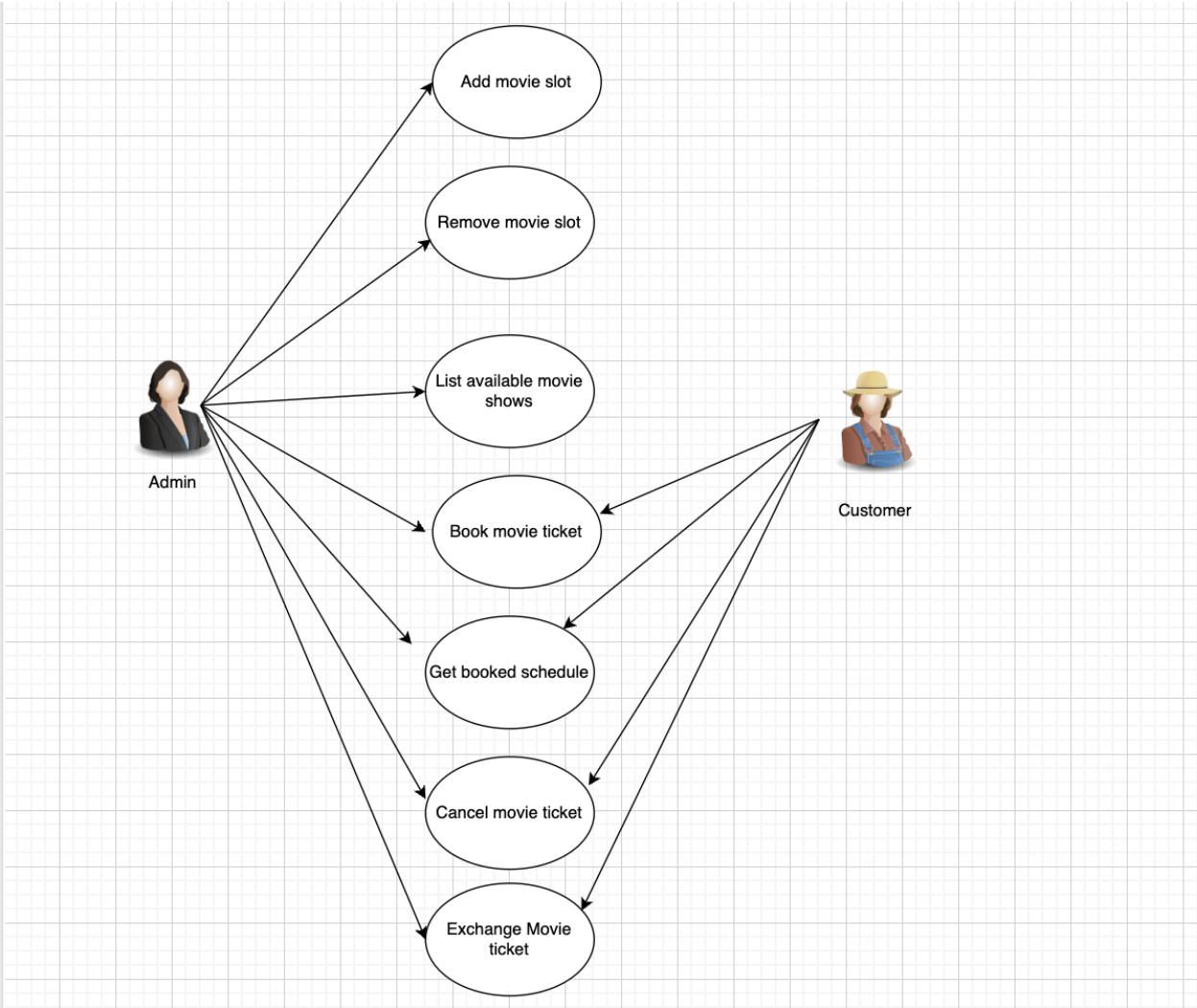
- **Book Movie Tickets (bookMovieTickets(customerID, movieID, movieName, numberOfTickets))**: Customers can book movie tickets by entering their customer-id, movie-id, movie-name and number of tickets that they want to book.

- **Get Booking Schedule (getBookingSchedule(customerID))**: The customers can check their movie reservations by entering their customer-id.

- **Cancel Movie Ticket (cancelMovieTickets(customerID, movieID, movieName, numberOfTickets))**: Customers can cancel their booked movie tickets by entering their customer-id, movie-id and movie-name that they want to cancel tickets for, and the number of tickets that they want to cancel.

- **Exchange movie tickets (exchangeTickets (customerID, movieID, new_movieID, new_movieName, numberOfTickets)):** A customer can exchange the number of tickets of already booked tickets with movieID for another movie id and movie name. If the new movie is available and has required capacity then the previous bookings of the user will be canceled and the new will be booked.

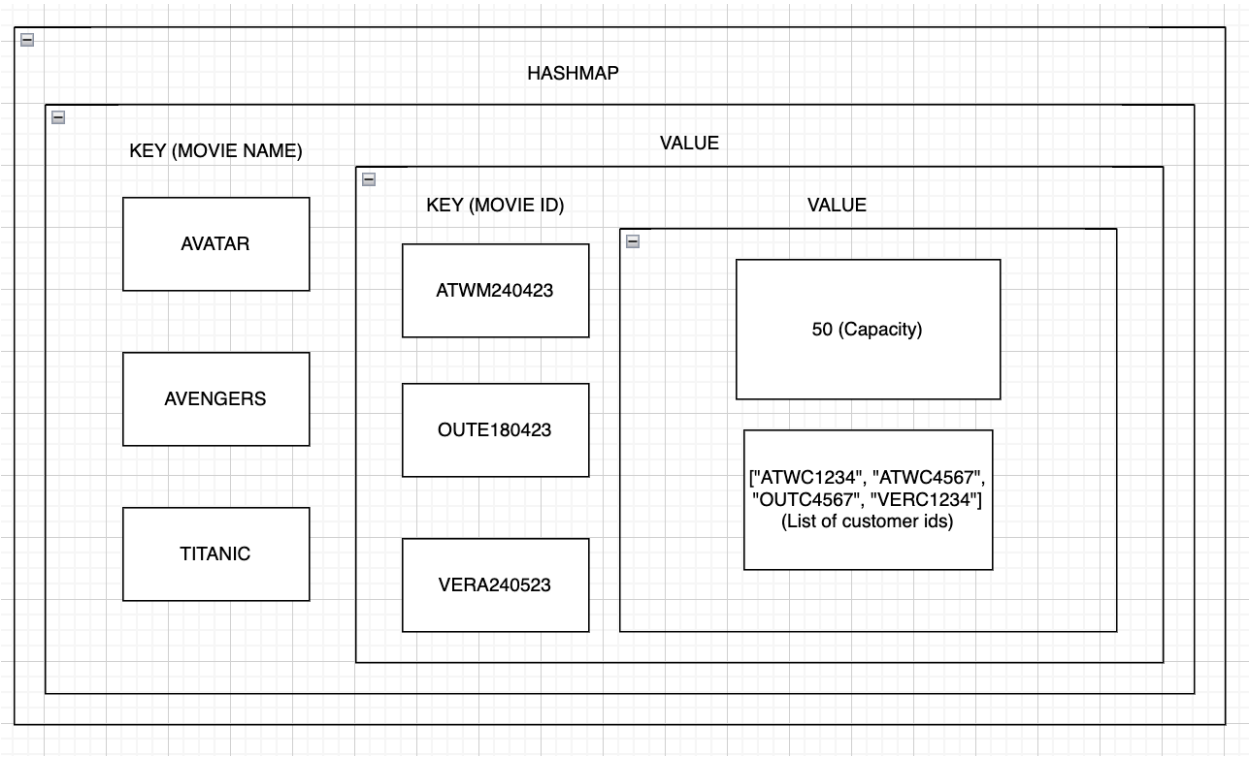
NOTE: Admin can also book movie tickets, cancel movie tickets, get booking schedule and exchange movie tickets using above functions.

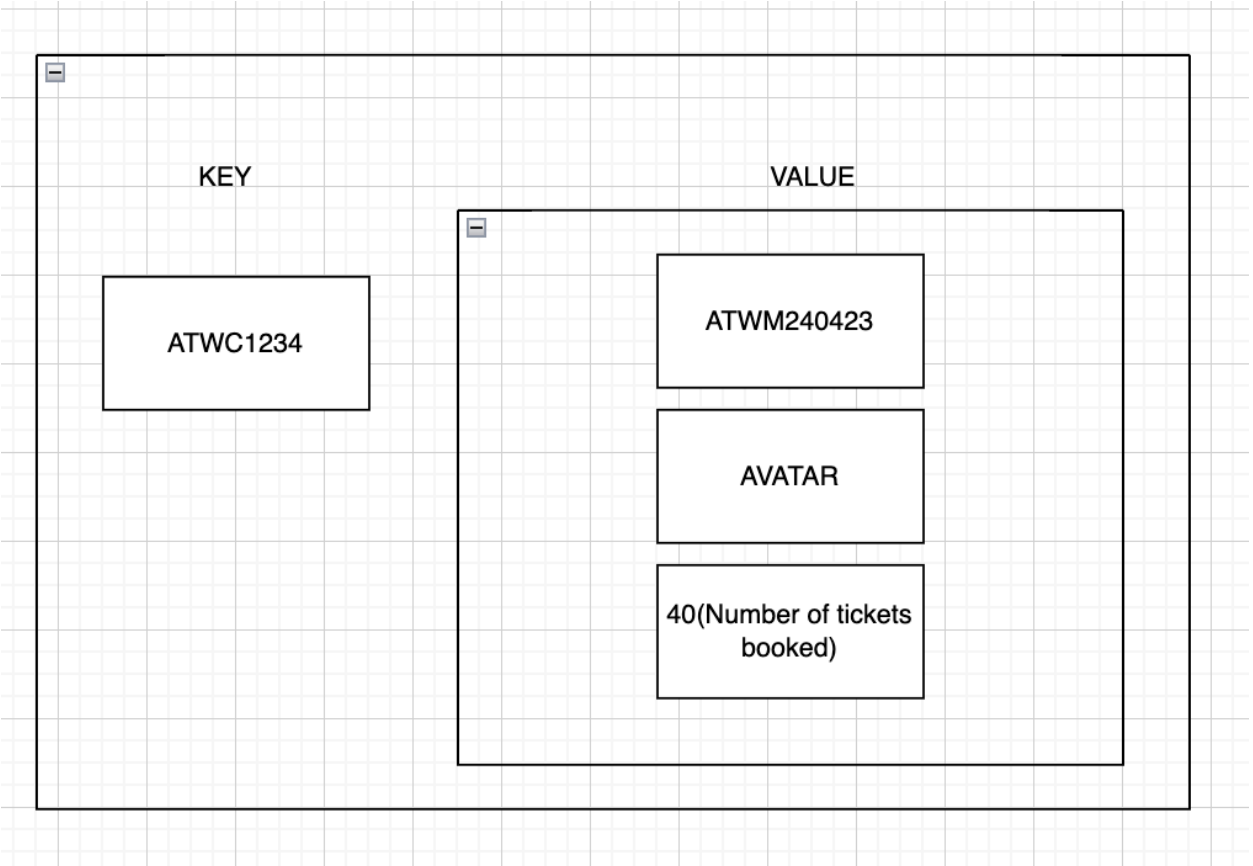


The above image shows the users and the respective actions they can perform.

Data structure

Two hashmaps with key value pairs are the type of data structure used in the application. Using such structure helped in the quick access and storage of information. The structure in detail is shown below.


HASHMAP
KEY (MOVIE NAME)
VALUE
AVATAR
AVENGERS
TITANIC
KEY (MOVIE ID)
VALUE
ATWM240423
OUTE180423
VERA240523
50 (Capacity)
["ATWC1234", "ATWC4567", "OUTC4567", "VERC1234"] (List of customer ids)


KEY
VALUE
ATWC1234
ATWM240423
AVATAR
40(Number of tickets booked)

Test cases

S. NO.	Test case	Function calls	Result	
1	Add Movie Slots	addMovieSlots("ATWE140323", "Avengers", 1)	Pass	
		addMovieSlots("ATWM150323", "Avatar", 54)	Pass	
		addMovieSlots("ATWM160323", "Avatar", 54)	Pass	
		addMovieSlots("ATWA160323", "Avengers", 1)	Pass	
		addMovieSlots("ATWM140323", "Avatar", 7)	Pass	
		addMovieSlots("VERM140323", "Avatar", 7)	Pass	
		addMovieSlots("ATWM170323", "Avatar", 7)	Pass	
		addMovieSlots("ATWE180323", "Avengers", 20)	Pass	
2	Remove upcoming slot	removeMovieSlots("ATWM160323", "Avatar")	Pass	
3	Remove past slot	removeMovieSlots("ATWE230123", "Avengers")	Fail (Cannot remove a slot that occured before current date)	
4	Listing availability	listMovieShowsAvailability("Avengers")	Pass (Lists the available slots for the given movie)	
5	Listing availability - Invalid Movie Name	listMovieShowsAvailability("Inception")	Fail (No movie exists by the given name)	
6	Booking - Ideal Case	bookMovieTickets("ATWC1825", "ATWA160323", "Avengers", 1) bookMovieTickets("ATWC1825", "ATWM140323", "Avatar", 3)	Pass	
7	Booking invalid movie slot	bookMovieTickets("ATWC1825", “ATWA160324", "Avengers", 1)	Fail (Movie slot with given id does not exist)	
8	Booking Housefull	bookMovieTickets("ATWC1825", "ATWA160323", "Avengers", 2)	Fail (The capacity of this theater set to only 1, and user already booked 1 ticket in this)	
9	Booking 3 tickets in other area	bookMovieTickets("ATWC1825", "VERM140323", "Avatar", 3)	Pass (Users are allowed to book upto 3 tickets outside their own area	

10	Booking 4th ticket in other area	bookMovieTickets("ATWC1825", "VERM140323", "Avatar", 1)	Fail (Users are allowed to book upto only 3 tickets outside their own area)	
11	Get bookings schedule	getBookingSchedule("ATWC1825")	Pass (Lists all the bookings by the customer)	
12	Get bookings schedule - Invalid customer id	getBookingSchedule("ATWC1826")	Fail (No bookings by the given customer id)	
13	Cancel Booking - Ideal Case	cancelMovieTickets("ATWC1825", "ATWM140323", "Avatar", 1)	Pass	
14	Cancel Booking - Invalid movie id	cancelMovieTickets("ATWC1825", "ATWM140324", "Avatar", 3)	Fail	
15	Cancel Booking - Movie not booked	cancelMovieTickets("ATWC1825", “ATWE180323", "Avengers", 1)	Fail	
16	Removal of slot by admin that has already been booked should book the next available slot for the customer	bookMovieTickets("ATWC1825", "ATWM140323", "Avatar", 7) removeMovieSlots("ATWM140323", "Avatar") getBookingSchedule("ATWC1825")	Pass (Should show slot "ATWM300323" booked for the user)	
17	Exchange not available if the customer has not booked the tickets they are trying to exchange. (Before this test case, add two slots by the ids "ATWM140323" and "ATWM170323" for the movie Avatar with capacities 8 and 5 respectively)	exchangeTickets("ATWC1825", "ATWM140323", "ATWM290223", "Avatar", 7)	Fail (As the customer has not booked tickets for the slot "ATWM140323", there is noting to exchange)	
18	Cannot exchange if the new slot does not have enough capacity as the number of tickets that the user wants to exchange. (Slots with ids "ATWM140323" and "ATWM160323" created for the movie Avatar in the test case 17)	bookMovieTickets("ATWC1825", "ATWM140323", "Avatar", 7) exchangeTickets("ATWC1825", "ATWM140323", "ATWM160323", "Avatar", 7)	Fail (As the capacity of the slot that the user wants to book for is 5, and the user needs 7 tickets)	
19	Invalid new slot id	exchangeTickets("ATWC1825", "ATWM140323", "ATWM280323", "Avatar", 4)	Fail (No movie slot by the given new slot id exists for the movie Avatar)	
20	Invalid current slot id	exchangeTickets("ATWC1825", "ATWM280223", "ATWM160323", "Avatar", 4)	Fail (No movie slot by the given current slot id exists for the movie Avatar)	
21	Ideal Case - Cannot exchange if the new slot does not have enough capacity as the number of tickets that the user wants to enchange. (Slots with ids "ATWM140323" and “ATWM160323" created for the movie Avatar in the test case 17)	exchangeTickets("ATWC1825", "ATWM140323", "ATWM160323", "Avatar", 4)	Pass	

Most difficult part of the assignment

1. Designing the project architecture as well as the data structures was the most difficult part as it involved nested hashmaps and linking.
2. “removeMovieSlots” function was difficult to implement as there were a lot of scenarios and conditions to keep in mind.