Distributed Movie Ticket Booking System Using Java RMI

Written by:
Student Name: Aryan Saxena
Student ID: 40233170

Table of Contents

I. INTRODUCTION

A. Problem Statement

To Design a Distributed Movie Ticket Booking System (DMTBS) which would be a comprehensive, easy, and the most efficient solution to access, book and managing the chains of theatre servers and to manage movie tickets for each server. The system is based to address the challenges that might be faced in a traditional movie ticketing booking system with the help of distributed system design to design an effective architecture which will help customers to book tickets across multiple servers and for the administrator(s) to manage those servers easily. The DMTBS or Distributed Movie Ticketing System is accessible for both Customers and Administrators. It is designed in a way so that, a customer is able to book a movie ticket from any of the servers until the tickets are available and can view all the shows for a movie from all the servers. The is user friendly design that is easy to understand and use. It also incorporates Administrator functions, so that the administrator can add movie slots with the name of a movie, ID for the movie and a value for how many slots are to be added for it. Additionally, administrator can also easily get the list of all the shows for a particular movie (using movie name) from across all the servers. This was implemented using UDP/IP protocol for server-to-server communication. The DMTBS is based on HashMap data structure that manages the movie details along with the customer details. It stores the details so that it is easier for the administrator to access, manage and alter information as required. It is also easy for the customer to fetch the details that might be required in an efficient manner. Additionally, the system also creates and maintains logs for every event that occurred, on server side as well as on the client side that will make it easier to log and backtrack any unexpected events that might occur, also providing detailed and comprehensive history of the system's activities.

This design documentation will provide a detailed overview of all the functionalities, architecture, and components of the system. The DMTS is an effective solution providing a cutting-edge solution for movie ticketing and its management and is secure, efficient, and provides a user-friendly experience for customers as well as administrators.

B. Objective

The object project is to develop a Distributed Movie Ticket Booking System to develop a secure, efficient, and user-friendly solution for movie ticket booking along with management for the theatres or servers. The system provides an efficient and centralised approach to the platform for the theater managers to manage the information of the movies, cancellations, and the shows across the server. The system is implemented on the basis of unique IDs of administrators and customers to identify the server/area that they belong. The customer can perform bookings, cancel bookings, and list all the bookings across all the servers effectively. Additionally, the system also consists of a logger function that will maintain logs for all the functions that are being performed which will make it easier for the developer to backtrack problems, identify anomaly and also catch any unexpected situations that might arise. The final goal of the project is to develop an effective solution that will be effective, efficient, and easy-to-use for the customer and also for the user.

II. TECHNINCAL APPROACH

A. Architecture Design

The system incorporates three different servers: Atwater (ATW), Verdun (VER), and Outremont (OUT). For every server, there are two data structures, first one contains the information of the movies, and the other contains the Customer Information. All the HashMap data structures are secured and cannot be access directly outside the class. This is done to securely have the data stored in the servers for all the clients. The RMIs interface is created which will help us invoke the methods remotely.

The implementation is based on client-server architecture. The client includes customers and administrators, that can access the servers with the help of Java remote method invocation, also known as Java RMI. We inculcate the methods of the server in order to perform all the necessary tasks that are required from each client. The administrator can create movie slots, delete movie slots, list a movie across all the servers, Create new administrators. The customer can book movies, cancel movie tickets, display the shows from across different servers.

The server-to-server connection was done using UDP (Unique Datagram Protocol) in order to effectively communicate the information across from one server to another. As the system consists of multiple servers and multiple administrators across different servers, it was important to inculcate UDP/IP protocol for data transfer across servers. Both the clients, i.e Customers and the administrators will be interactive to the servers through a client. The client will incorporate a user interface (UI), that will just take the necessary inputs from the client and will process to do the necessary actions and validations, then takes the request and sends to the target server. The UI component will display the response from the server on the client side itself, with the help of the console.

The entire system works on the cornerstone of the prefix of the ID's (adminID and the customerID) in order to determine which server does the user belong to. For Example: If the prefix of the userID starts with "VERA", then this implies that the user belongs to the server Verdun("VER") and is the administrator of the server ("A"). So, the UI will process accordingly.

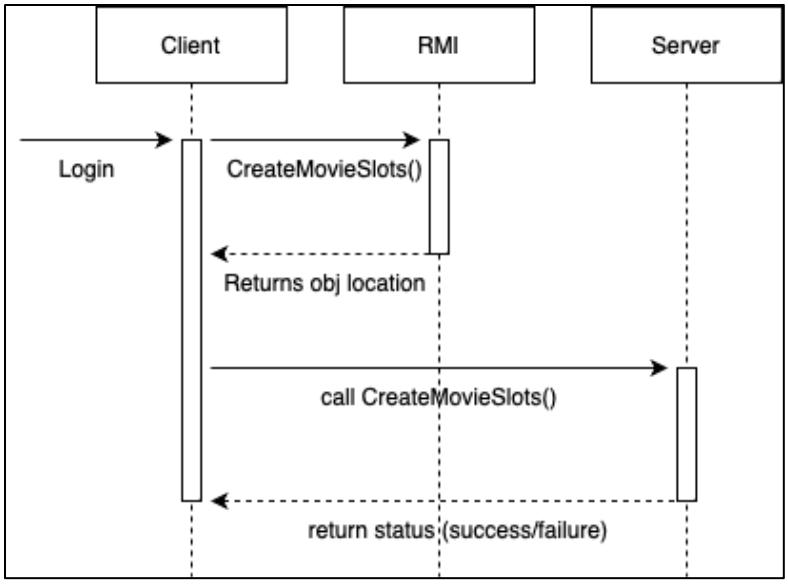

Diagram: UML Sequence Diagram

B. Data Structure

1. HashMap

HashMaps are used to store the information across all the servers. The information is incoming from two clients: Customer and Administrators and is securely stored on the servers. All the information is stored in mainly two hashmaps i.e movie hashmap and customer hashmap. Movie hashmap contains the name of the movie as the key ("Avatar","Avengers", and "Titanic") and has a sub-hashmap for value, the sub-hashmap will have the movieID as the key and the amount of seats available as the value. The other hashmap, customer hashmap will use the CustomerID as the key, and has a sub-hashmap for the value. Inside the sub-hashmap there is a combination of moviename:movieId as the key and number of tickets booked as the value. The diagram below can help visualise the movies hashmap. The customer HashMap has been implemented in a similar fashion.

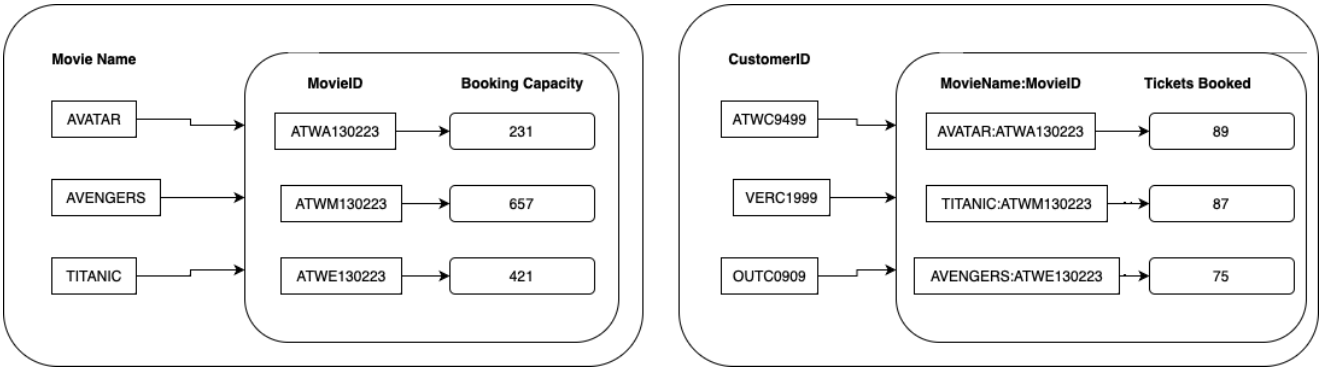

Diagram: Movies and Customer HashMap Implementation

2. ArrayList

The administrators are fixed for the servers. The server's start with a single admistrators. However, that admin has been granted the ability to create more administrators. All the information of the administrators is stored on an ArrayList<String>. Whenever an Admin ID hits the server, it gets verified whether it is valid or not. The validity is checked using the '. contains' functionality of the ArrayList. So, if the administrators are present in the ArrayList then the server allows the admin to login and proceed. Otherwise, the process is repeated, and the user is asked for a UserID again. When the admin is authenticated, it has add another administrator using the '.add' feature of the ArrayList. The dynamic size of the ArrayList<String> helps us manage the administrators with ease.

3. LogFile

The logs are generated on the Server side as well as on the Client side to effectively get a comprehensive action that are being performed. The logs will help in case of an anomaly or some unexpected event that might occur. The logs on the Server side are created with the help of Serverlogwriter function, that takes in the event occurred, key values, and status as the parameters and creates the logs for the event on a file for the server.
In a similar fashion, logs are created with clientlogwriter function on the client side, takes in the same parameters as mentioned above and creates a log for the client based on the current date.

4. Date and Time

Java's inbuilt function of 'Date and Time' are used in order to get systems current date and perform different operations in the date format as requested by the customer or the administrators.

III. DETAIL DESIGN

Let us break-down the detail design between different headings in order to understand it in a better way.

A. Server Side

1. Verifications

There are two verifications that are performed to the input, which are imperative for the functioning of the system and work properly. The prefix of any 'ID' determines which server will be hit for the verification. There only three servers are 'ATW', 'VER' and 'OUT'. Any prefix other than these will be considered invalid, and the user will be prompted to re-enter the ID.

1.1. ID Verification

The String is first sent to the server, post which the function *verifyID(String ID)* is called. Post which, the ID is sent to the server for verification. First it checks whether the ID is 8 characters or not.

The first three are prefix, fourth character determines admin or customer, and the rest is a unique ID for the admin or the customer. If the fourth character of the ID is 'A' then it is determined as an Admin. The it is checked if the admin exists in the admin ArrayList or not, using the '. contains' function. Only when it is verified can the admin access the server otherwise, it will return the invalid message. On the other hand, if the fourth character is 'C' then the CustomerMenu is displayed. However, the CustomerID will only be added to the database post the booking, otherwise it is checked whether the ID is correct or not. In any other case, an invalid message is returned to the client.

1.2. MovieID Verification

For movie ID verification, the verifyMovieID(String movieID) is called with which the movie ID is verified. First it is checked whether the movieID is of 10 characters or not. If not, appropriate message is returned. After this, the function reads the fourth character. The fourth character has to be 'M','E' or 'A' signifying Morning, Evening or Afternoon respectively. If not, appropriate message is returned. Afterwards, the last 6 are checked. Out of the 6 characters, it is check whether the date is correct or not. It checks the date in ddMMyy format and checks whether there is any anomaly. Then it checks the systems date, and checks whether the date within today and one week from now, if not then appropriate message is shown to the user.

After which the movieID is validated. And 'valid' keyword is returned to the client.

2. Admin Operations

There are 4 unique operations that can be performed by an administrator.

2.1. Add Movie Slots

This function adds the movie slots to the server depending on the prefix of the AdminID, which will determine which server will be accessed by the client. Post accessing, the function *addMovieSlots(String movieID, String movieName, int bookingcapacity)* is called. In the movieID a unique movie ID is input from the admin for which he/she wishes to input data in the hashmap for a movieName. Then the admin is prompted to add the booking capacity. In this function a subhashmap is created with the booking capacity and the movieID, which is then put in the main hashmap with movieName. So the data is added to the 'movies' Hashmap. The subhashmap is created everytime this function is called and then it is added/appended to the main hashmap depending on whether the movie ID exists or not.

2.2. Remove Movie Slots

The function removeMovieSlots(String movieID, String movieName) is called when the request hits the server. Post the function call, the function checks the movie Hashmap. If the movie does not exists, then throws the message displaying 'the movie does not exist', otherwise it proceeds, after that it checks whether the sub-hashmap contains the movieID asked to delete. If it does not exists, the displays the appropritate message to the client, otherwise proceeds. After it finds the movieID for the movie that needs to be deleted. It performs the delete operation on the movieID, and remove the subhashmap of the requested slot, and sends the confirmation message to the client.

2.3. List Movie Shows Availability

This is an important function for an administrator. In order for the admin to view the list of shows from all the different servers; **UDP/IP protocols** were utilized and implemented for effective server-to-server information exchange.

Let us understand the working of this function. Initially the server checks its own hashmap for movies if in case a movie exists in the server or not, then the result is appended to an output string. Then a datagram socket is created that will send the data to the other two servers on their always open port in order for them to get, process and resend the request to the base server. We will understand this working later in this document in the UDP/IP working below. It keeps the port open in order to receive the data from the servers using a while loop. When the data is received from the server. It is then appended to the output string. The same process repeats for the other server. After the output string contains the data from the three servers (own (accessed locally)+other two(received using UDP/IP)) then the logs are generated for successful completion, and the output string is returned to the administrator successfully.

UDP/IP Working: All three servers incorporate an always on Datagram port that will always be open in order to get request in case any other server request for data from them. When the data hits the datagram socket, it takes the data from it, converts it to a string. The function *listMovieServertoServer (String movieName, String serverrequest)* is called, and then proceeds to decipher the source from which the request was received from. Once that is confirmed. It calls the data from the hashmap stored inside it locally, converts it to string and and the converts it into a byte[1024]. After this, it returns it back to the data ports (the one from which the data was received) of the other server and sends backs its data. The same step is repeated in the other server. Resulting in two strings being returned, which are then processed in the base server back into a string and is added to the output string.

2.4. Add Another Admin

There are one base admin in each of the servers. Only the administrator has the right to create more admins. This can be performed with the help of *addadmin(String adminID)* function. When this function is called, it takes in a string from the user that will contain the details of the adminID that is to be created.

The adminID that is to be created is first validated, using the constraints as explained above. And after the validation is done, the adminID is added to the ArrayList of the database. Thus, creating a new admin of the server.

3. Customer Operations

There are 3 unique operations that can be performed by a customer.

3.1. Book Movie Ticket

When a customer is validated. A he/she/other wishes to book a movie ticket, then the function *bookMovieTicket(String CustomerID, String movieID, String movieName, int Numberoftickets)* is called. When the function is called it checks for various parameters. First it checks whether the movie is exists using the movieName, then checks whether the slot is available using the movieID. Post validation, it checks if the number of tickets that the customer has requested for is not more than available slots. Once that is validated, it checks whether the customer is foreign customer or not, if he/she/other is then they can only book the ticket 3 times other than their local server. Then, the customer hashmap is checked whether the customer exists in the hashmap, or he/she/other have already booked this movie, with the same movieID before. If that returns true then the number of tickets are just incremented to the old number. Otherwise, it creates the customer using the ID and books the slot and puts all the information in the hashmap. Finally, the tickets are deducted from the movies hashmap accordingly.

3.2. Get Booking Schedule

When this option is invoked, the function *getBookingSchedule(String CustomerID)* is called. In this function, initially it is checked whether the customer exists in the hashmap. This is done with the help of CustomerID provided, if that fails then the appropriate message is sent to the client. Otherwise, it creates a temporary hashmap, fetching all the details from the sub-hashmap of the provided CustomerID. Then it is iterated and all the data is put in an ArrayList. After which, the values of the arraylist are returned using the toString method.

3.3. Cancel Movie Tickets

Post selection of this option, the request is sent to the server and the function cancelMovieTickets(String CustomerID, String movieID, String movieName, int Numberoftickets) is invoked. When the function is invoked, it first checks if the customer exists. Then it checks whether the number of tickets that the customer has booked is greater than equal to the amount he/she/other wishes to cancel. Post this, it checks the movies database. It decrements the bookingcount in the customer hashmap, totalbooking – bookingcancellation. After that the moviehashmap is updated, and the movies that were subtracted are added back to the main movies hashmap.

4. General Operations

There are 3 operations that are general operations that can be performed by either of the clients, i.e Customer or Administrator.

4.1. Use Different ID

When this option is selected, the while loop that is running for the current menu is broken and the you can re-enter the userID to proceed back into the program with a different ID.

4.2. EXIT

When this option is selected, the main program loop (while loop) is exited using the break function.
The conclusion message is displayed on the screen and the system exits.

B. Client Side

1. Data Management

The client side will not be storing any sensitive data on the client side. It only acts as a bridge or UI (User Interface) between the user and the Servers (Theaters for this case). The data management is handled by the servers themselves.

2. Logging Mechanism

The client side keeps a log of user's actions and takes in the responses from the servers. The logs include the date and time of when the event occurred, the type of request, the request parameter and whether the request parsed successfully or not. This logging mechanism will help keep a flow and track of user's actions and can be used to backtrack in case of an anomaly.

IV. TESTING

A. Test cases for Admin:

Test Case 1: Add movie slots
Expected Result: The movie slot should be added successfully, and the details should be reflected in the list of available movie shows.
Result:

```
MovieID:
ATWA130223
Enter Name of the Movie:
AVATAR
Enter the Capacity you wish to add:
199

AVATAR with MovieID ATWA130223 has been created with 199 capacity
```

Verifying:

```
3
Enter the Movie Name of which you wish to see the shows for:
AVATAR

Atwater : [ATWA130223 with 199 capacity]  Outremont : [] Verdun : []
```

Passed.

Test Case 2: Remove movie slots
Expected Result: The movie slot should be removed successfully and should no longer be present in the list of available movie shows.

Result:

```
3
Enter the Movie Name of which you wish to see the shows for:
AVATAR

Atwater : [ATWA130223 with 199 capacity, ATWM150223 with 400 capacity]  Outremont : [] Verdun : []
```

Deleting Slot:

```
MovieID:
ATWA130223
Enter Name of the Movie:
AVATAR

AVATAR with MovieID ATWA130223 has been removed
```

Verifying:

```
Enter the Movie Name of which you wish to see the shows for:
AVATAR

Atwater : [ATWM150223 with 400 capacity]  Outremont : [] Verdun : []
```

Passed.

Test Case 3: List movie shows availability
Expected Result: Movie ID's for the given movie Name with capacity from all the servers

Result:

```
Enter the Movie Name of which you wish to see the shows for:
AVATAR

Atwater : [ATWM150223 with 400 capacity]  Outremont : [OUTE170223 with 88 capacity]
 Verdun : [VERM130223 with 99 capacity]
```

Showing Results across all the servers.
UDP/IP working successfully.
Passed.

B. Test cases for customer actions

Test Case 1: Book a movie ticket

Expected Result: The movie ticket should be booked successfully, and the details of the booking should be displayed.

Result:

Booking Movie Tickets:

```
MovieID:
ATWM150223
Enter the nubmer of tickets to be booked:
233
233 Tickets have been booked to ATWM150223 show of AVATAR
```

Getting Booking Schedule:

```
2
ATWATER: [233 tickets for AVATAR:ATWM150223]
VERDUN: No shows booked in Verdun
OUTREMONT: No shows booked in Outremont
```

Checking from admin, the number of tickets left for the movie:

```
Enter the Movie Name of which you wish to see the shows for:
AVATAR

Atwater : [ATWM150223 with 167 capacity]  Outremont : [OUTE170223 with 88 capacity]
 Verdun : [VERM130223 with 99 capacity]
```

Capacity decreased. Slots booked by the customer.
Passed.

Test Case 2: Cancel a movie ticket: To test the functionality of cancelling a movie ticket.
Expected Result: The movie ticket should be cancelled successfully, and the details of the booking should be updated.

Result:

Cancelling Movie Ticket:

```
MovieID:
ATWM150223
Number of tickets you wish to cancel
34
34 tickets has been cancelled for AVATAR with MovieID ATWM150223
```

Getting Booking Schedule:

```
ATWATER: [199 tickets for AVATAR:ATWM150223]
VERDUN: No shows booked in Verdun
OUTREMONT: No shows booked in Outremont
```

Checking from admin, the number of tickets for the movie:

```
Enter the Movie Name of which you wish to see the shows for:
AVATAR

Atwater : [ATWM150223 with 201 capacity]  Outremont : [OUTE170223 with 88 capacity]
 Verdun : [VERM130223 with 99 capacity]
```

Slots got removed and got added back to the available slots of the movie successfully.
Passed.

Test Case 3: Get Booking Schedule
Expected Result: The details of all bookings made by the customer should be displayed.

Result:

```
ATWATER: [199 tickets for AVATAR:ATWM150223]
VERDUN: No shows booked in Verdun
OUTREMONT: No shows booked in Outremont
```

Showing the movies booked by the user.
Passed.

V. CONCLUSION

We conclude that the DMTBS is a complex and intertwined system but is robust system for theatre managers and also customers. The system functions with different servers as different theaters; namely: Atwater, Verdun, and Outremont. It implements unique admin and customer IDs for identification. It maintains logs on both client side and the server side. It logs all the function calls from the customer or the admin, and what functions took place in the server and if it succeeded or not. Administrators have been given the ability to add, remove and list the movie shows also from other servers using UDP/IP. Customers have been given the ability to book, cancel and view the show for which tickets they have booked for. System additionally also limits the customer from booking three movies from other servers than their own local server (based on the prefix). This system successfully provides an outline of an implementation of a movie ticket management system.