Distributed System Design

COMP 6231 – Winter 2023

Concordia University

Department of Computer Science and Software Engineering

Instructor: R. Jayakumar

Distributed Movie Ticket Booking System (DMTBS) using Java RMI

Assignment – 1

By: Shivam Patel

ID : 40226428

Index

Overview

The Distributed Movie Ticket Booking System(DMTBS) is widely used by theatres, using this system a user can book their movie ticket for particular movie at particular locations. Moreover, Admin can also add the movie slots for given movies. The aim of this system is to connect the movie theatre servers which are located distributedly and help user to book and manage their tickets from any location.

The DMTBS system have three different locations for which user can book the movie tickets. The locations are as follow : 1. Atwater (ATW) , 2. Verdun (VER) and 3. Outremont(OUT). This locations are the movie theatre location for any famous movie theatre branches. The servers are located at this three locations and that are accessible only by the admins to add information regarding movie and by customer to perform operations such as booking of movie tickets.

The Admin and customer have their unique user ID for login purposes. The user ID is constructed by the acronym of their area and a 4 digit number (e.g. ATWA2345 and ATWC2345), where the 4^{th} letter indicates where the user is customer or admin.

Here customer and admin has different tasks to do.

The task associated with Customer is as follows :

1) Book Movie
2) Get Movie Schedule
3) Cancel Movie

The admin can also be a customer for ticket booking so, admin can also perform the operation of the customer , whereas the customer cannot perform the admin operations.

The task associated with admin is as follows:

1) Add movie
2) Remove Movie
3) List Movie Availability
4) Book Movie

5) Get Movie Schedule
6) Cancel Movie

Customer can book unlimited number of tickets at the area from where they belong. In addition, Customer can also book their movie in other areas but this bookings are limited to 3 in a week. Eg. Atwater Customer can book their movie ticket in Verdun and Outremont, but the maximum number of movie they can watch is three per week.

Furthermore, there are three movies for this system, namely Avatar , Avengers , Titanic. There are three various timeslot (Morning(M) , Afternoon(A) , Evening(E)) as well. MovieID is given in one format and is a combination of server , timeslot and date. For Ex, ATWA121022 means Atwater server (From substring 0 to 2) , afternoon show (Char At 3), 12/10/2022 (Rest substring) date.

Here Customer can only book movie, if it is not full yet. System should also count the value of booking of each customer. If Customer wants her can book as many tickets as want but not for same show same Id in different location.

The purpose of this system is to explain Java RMI fundamentals. RMI, or Remote Method Invocation, is used here. You can create distributed applications based on Java technology that can interact with one another using Java Remote Method Invocation (Java RMI). It is possible for other Java virtual machines (JVMs), which may be on different hosts, to call remote Java object methods.

System Architecture

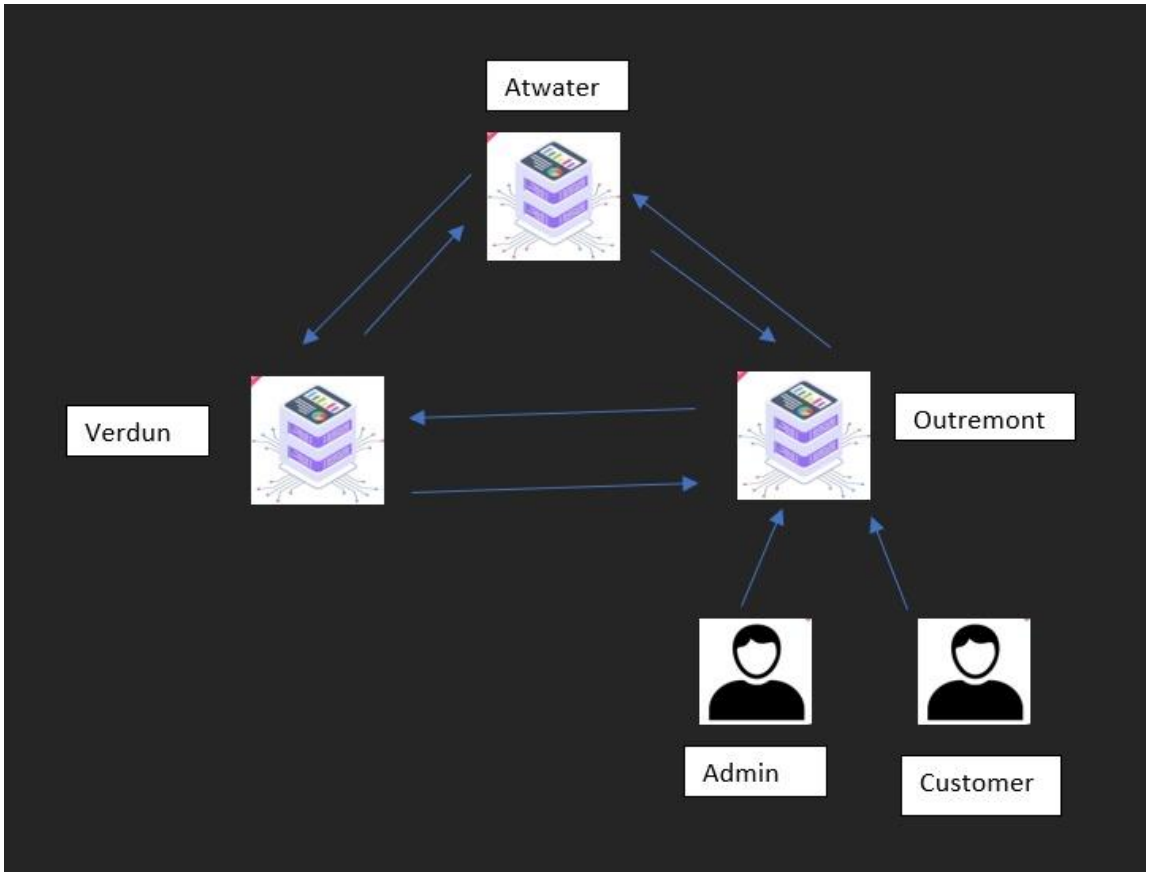


Fig1 . System Architecture

Three servers make up this system: Atwater , Verdun , Outremont. RMI is used for user (Admin or Customer) and server communication. where servers talk to one another using UDP/IP.

UDP/IP port are given below where the hospital server are running:

- Atwater : 8888
- Verdun : 7777
- Outremont : 6666

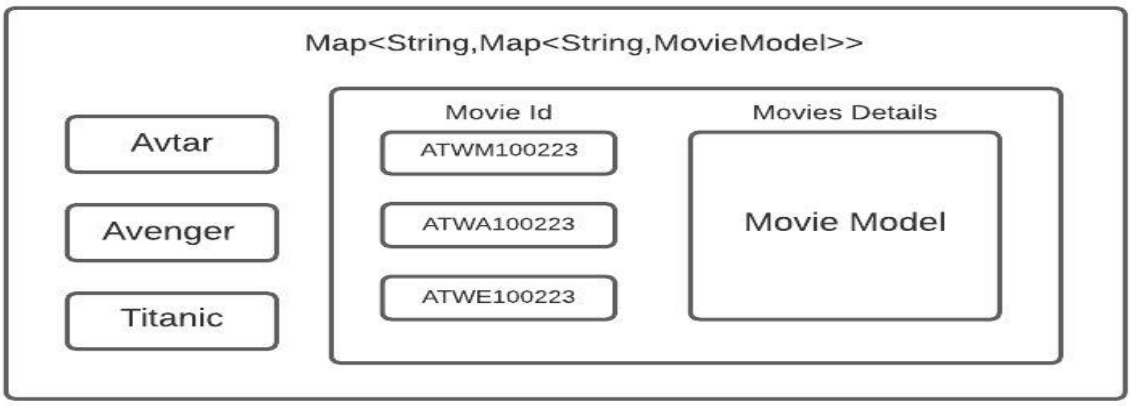
Each server has an own database that is made up of HashMap. The Movie type and related Movie id are both included in this database. Each Movie ID comprises information about the Movie's capacity and the Customer ID that made the booking. By obtaining the user's prefix from their user id, the system will allocate a server. Each server has a designated admin who alone is able to manage that server's activities. Each admin needs a password to do tasks.

However, he or she is only permitted to a maximum of three bookings of each type per week at Movies in other cities.

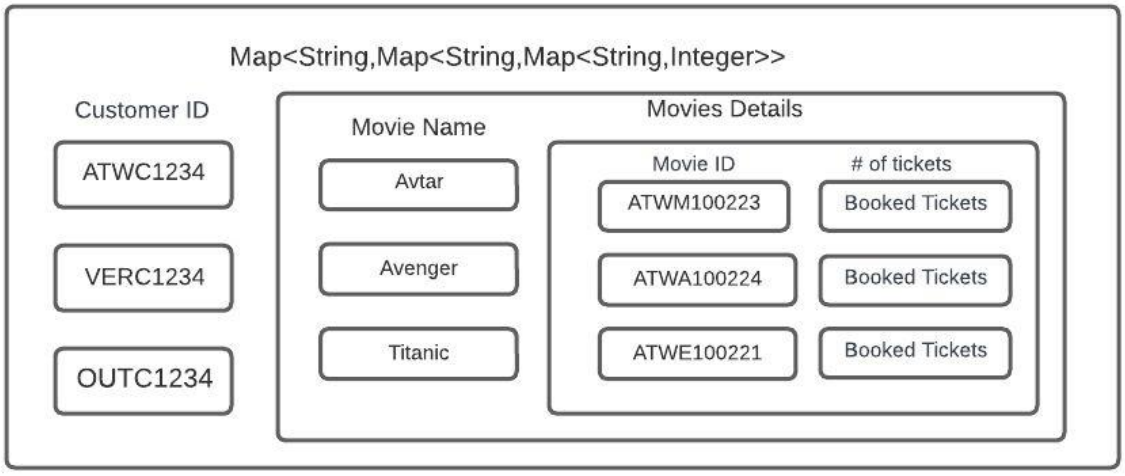
The database for Atwater is shown in this table. It consists of a HashMap with an Movie type key and a sub HashMap as its value. The appointment ids are the key in the sub Hashmap, and the values include array details like capacity and the Customer who made the Booking. The Booking's capacity is always indicated by the first member of the array. The Customer ids that have made the Booking are represented by the final element.

Data Structures

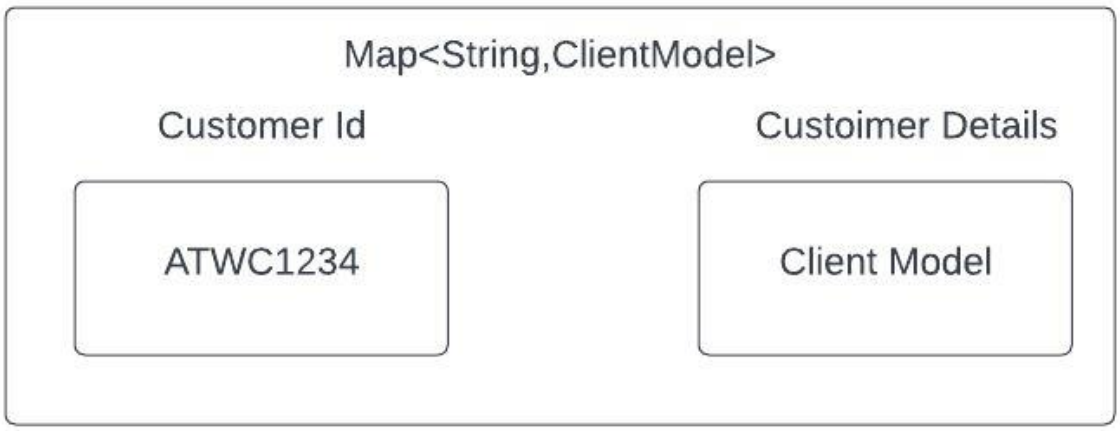
All the data is maintained within each server, using three Map Structures shown in the figure below.



DataModel

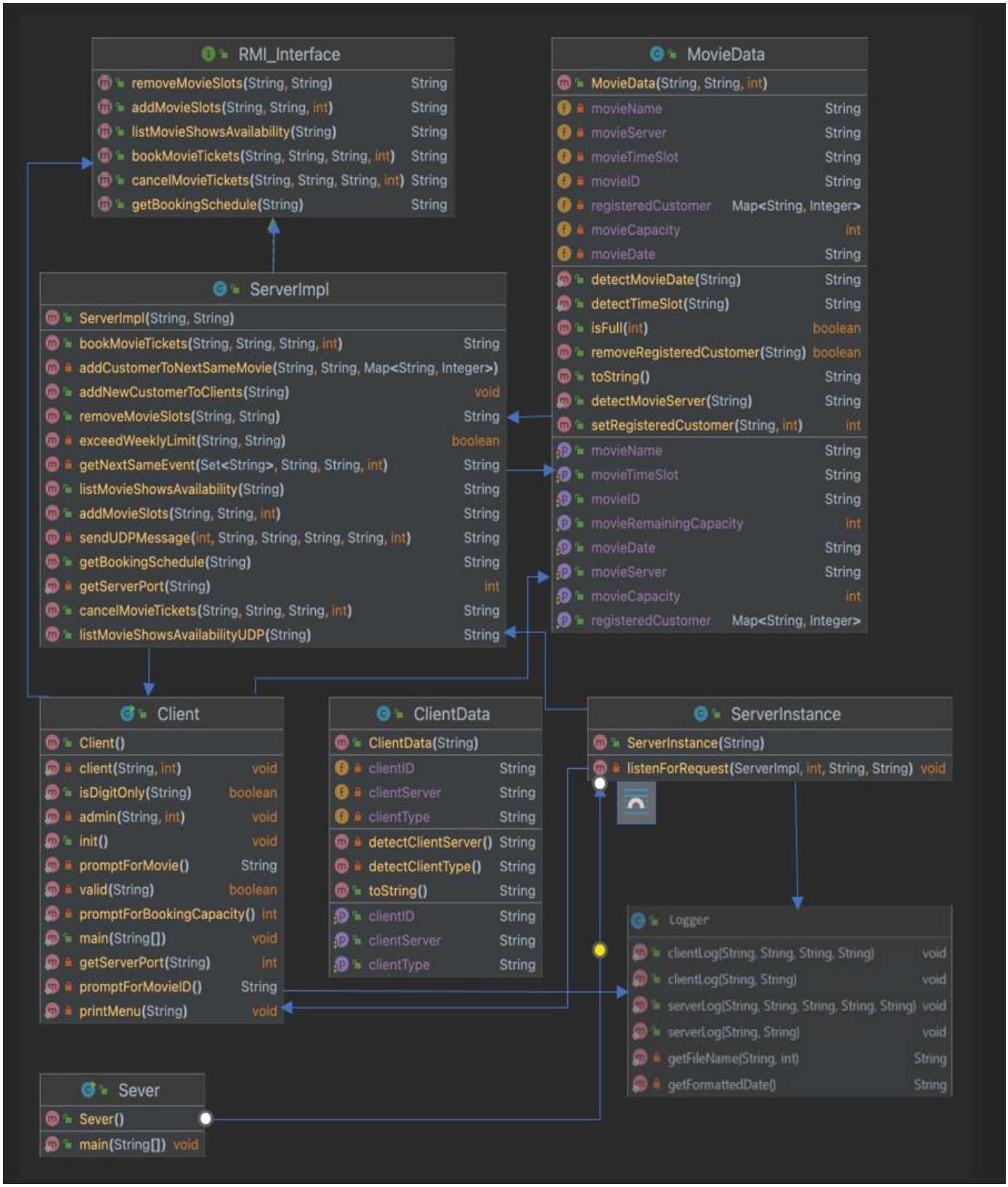


ClientServer



ServerClient

UML DIAGRAM


RMI_Interface
removeMovieSlots(String, String) String
addMovieSlots(String, String, int) String
listMovieShowsAvailability(String) String
bookMovieTickets(String, String, String, int) String
cancelMovieTickets(String, String, String, int) String
getBookingSchedule(String) String
MovieData
MovieData(String, String, int)
movieName String
movieServer String
movieTimeSlot String
movieID String
registeredCustomer Map<String, Integer>
movieCapacity int
movieDate String
detectMovieDate(String) String
detectTimeSlot(String) String
isFull(int) boolean
removeRegisteredCustomer(String) boolean
toString() String
detectMovieServer(String) String
setRegisteredCustomer(String, int) int
movieName String
movieTimeSlot String
movieID String
movieRemainingCapacity int
movieDate String
movieServer String
movieCapacity int
registeredCustomer Map<String, Integer>
ServerImpl
ServerImpl(String, String)
bookMovieTickets(String, String, String, int) String
addCustomerToNextSameMovie(String, String, Map<String, Integer>)
addNewCustomerToClients(String) void
removeMovieSlots(String, String) String
exceedWeeklyLimit(String, String) boolean
getNextSameEvent(Set<String>, String, String, int) String
listMovieShowsAvailability(String) String
addMovieSlots(String, String, int) String
sendUDPMessage(int, String, String, String, String, int) String
getBookingSchedule(String) String
getServerPort(String) int
cancelMovieTickets(String, String, String, int) String
listMovieShowsAvailabilityUDP(String) String
Client
Client()
client(String, int) void
isDigitOnly(String) boolean
admin(String, int) void
init() void
promptForMovie() String
valid(String) boolean
promptForBookingCapacity() int
main(String[]) void
getServerPort(String) int
promptForMovieID() String
printMenu(String) void
ClientData
ClientData(String)
clientID String
clientServer String
clientType String
detectClientServer() String
detectClientType() String
toString() String
clientID String
clientServer String
clientType String
ServerInstance
ServerInstance(String)
listenForRequest(ServerImpl, int, String, String) void
Logger
clientLog(String, String, String, String) void
clientLog(String, String) void
serverLog(String, String, String, String, String) void
serverLog(String, String) void
getFileName(String, int) String
getFormattedDate() String
Sever
Sever()
main(String[]) void

TEST SCANARIO

#	Type of Test	Scenario	Cases
1	Login	UserName	1.Admin ID 2.Customer ID
2	Menu Items		
3		Logout	1.Log out menu Item
4	Admin	*addMovieSlots* ()	1.invalid MovieID -> not added 2.new MovieID -> added 3.Existing MovieID (LowerCapacity) -> not allowed 4.Existing MovieID (HigherCapacity) -> capacity Updated 5.Duplicate MovieID -> not happening 6.MovieID of Other Servers -> not allowed
5		*removeMovieSlots* ()	1.invalid MovieID 2.MovieID not exist 3.Movie without anyone registered -> removed movie 4.movie with someone registered -> Removed movie + registered to same movieName if possible (UDP if needed) 5.MovieID in other servers -> not allowed
6		*listMovieShowsAvailability* ()	1.list all movies of a given type from all three servers (UDP needed) 2.Movie Name is forced correctly with showing only options available
7		Ask for customerID	1.Access Customer methods
8	Admin + Customer	*bookMovieTickets* ()	1.on own server -> allowed 2.if event is full -> not allowed 3.on other servers -> only three in a week (UDP needed) 4. invalid MovieID -> not allowed
9		getBookingSchedule()	1.Show booking schedule of customer 2.invalid customerID -> not allowed 3.customer not exist ->ok
10		*cancelMovieTickets* ()	1.cancel on own server -> ok 2.cancel on other server -> ok(UDP needed) 3.cancel a not registered Movie -> error shown 4.invalid MovieID -> not allowed