Distributed System Design

COMP 6231 – Winter 2023

Concordia University

Department of Computer Science and Software Engineering

Instructor: R. Jayakumar

**Web Service Implementation of the Distributed Movie Ticket Booking System - Assignment-3**

By: Mir Pasad

Student ID: 40253287

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Sr No. | Topic | Page No. |
| 1 | Overview | 3 |
| 2 | Implementation | 5 |
| 3 | WEB Interface | 6 |
| 4 | Data Structure | 7 |
| 5 | Test Case | 8 |

1. **OVERVIEW:**

Three separate servers spread over three different stations make up the Distributed Movie Ticket Booking system (DMTBS):

* Atwater (ATW)
* Verdun (VER)
* Outremont (OUT)

There are two categories of clients for this system:

• Admin

• Customer

We must ensure that these clients are connected to their own servers with Java RMI, and also the connection between our three servers are done through UDP/IP socket programming.

**Admin Specific functions:**

* InsertMovie(): Only the admin server can insert movies.
* RemoveMovie(): admin can only remove movies from their own servers; if a movie was removed, we would need to arrange another movie that was close by for the customers who had previously booked tickets for it. For a server-server connection, UDP is required.
* DisplayAvailability(): This function must be used to collect all movies of a specific type from all three servers. For a server-server connection, UDP is required.
* BookMovie()- This function allows the admin to book movie tickets specified by the customers requirement.
* BookingSchedule(): it allows the admin to obtain the booking schedule for all the movies and work accordingly for further bookings.
* CancelTickets(): The admin can also cancel particular ticket bookings if there was an error.
* Admin Login/Logout(): Allows the admin to login/logout.

**Customer/Admin Specific Functions:**

* PurchaseMovieTickets(): which allows users to reserve events from other servers up to three times per week. For a server-server connection, UDP is required.
* SchedulingReservation(): it shows the customer booking schedule.
* CancelMovieTicket(): Customers can remove or cancel any tickets booked from their own schedules. For a server-server connection, UDP is required.
* swapMovieTicket() allows customers to switch a reserved event for another event. (a cancelMovieTicket Plus a PurchaseMovieTickets) -> must be atomic

CustomerID (8 characters): serverID (3 characters) + clientType (C/M) + 4 digit identifier is used to identify clients.

Avatar, Avengers, Titanic, along with their movieType and movieID (10 characters)—serverID (3 characters), movieSlot (M, A, and E), and movieDate—are used to identify events (DDMMYY).

1. **IMPLEMENTATION:**

* Client – Server communication is done by SOAP based Webservice.
* Atwater service address: http://localhost:8080/atwater?wsdl
* Outremont service address: http://localhost:8080/outremont?wsdl
* Verdun service address: http://localhost:8080/verdun?wsdl
* @WebService (endpointInterface = "com.web.service.WebInterface")
* UDP/IP Socket programming is used for server-server communication.

The following ports are used:

* ATWATER UDP port: 8888
* VERDUN UDP port: 7777
* Outremont UDP port: 6666
* We utilised a single server implementation file and a single interface implementation file to reduce code duplication and to make updates and debugging easier.
* The Server and Client each keep their own log files.
* The serverName.txt file can be found in the project directory's src Logs Server directory.
* The project directory's srcLogsClientClientID.txt file is where you can get client logs.
* To achieve the highest level of concurrency, we employed concurrent Hash Maps to store the data.
* The most crucial aspect of the implementation was to keep UDP calls out of infinite loops, particularly in the removeMovie() and DisplayAvailability() methods.
* The removeMovie() method was the hardest to develop when clients were enrolled in the event and some of them came from other servers.
* Because the swap method must be atomic, we scheduled the newMovie first (in a manner akin to a reservation), and if it was successful, we cancelled our oldMovie. If the cancellation attempt was unsuccessful, we cancelled our previously scheduled newMovie(cancelReservation).
* Added a test concurrency to client to check how our database is thread-safe or not? We concurrently request 5 book/cancel events to an event with 2 capacities.

1. **WEB INTERFACE:**

**A screenshot of a computer

Description automatically generated**

1. **DATA STRUCTURES:**

Each server maintains all the data using the three Map structures depicted in the diagram below.

Diagram

Description automatically generated

Diagram

Description automatically generated

Chart, diagram, box and whisker chart

Description automatically generated

1. **TEST CASE:**

|  |  |  |  |
| --- | --- | --- | --- |
| Sr No. | Test | Scenario | Test Case |
| 1 | Login | Username | * Admin ID * Customer ID |
| 2 | Menu | Logout | Menu for logging out |
| 3 | ADMIN | InsertMovie() | * Invalid MovieID-not added. * New MovieID-added. * Already existing MovieID-to increase the cap server. * Duplicate Movie-not allowed. * MovieID of other servers-not allowed to add |
| 4 | RemoveMovie() | * Not a valid MovieID * Non existing MovieID * Movie with no registration-remove movie * Movie with few registration- remove movie + register to same movieType if possible (UDP if needed) * Other server MovieID-not allowed to add |
| 5 | DisplayAvailability() | * Display all movies of a given type from all the servers. (UDP Needed) |
| 6 | Purchase and Cancel Movie Tickets | * On own server- allowed to purchase and cancel movie ticket after asking for customerID * If movie full-not allowed to purchase or cancel movie ticket. * Other server-Purchase tickets only thrice a week(UDP needed). * Invalid movieID-not allowed to purchase or cancel movie tickets. |
| 7 | ADMIN + CUSTOMER | PurchaseMovieTicket() | * On own server-allowed to purchase tickets * Movie capaserver is full- not allowed to purchase tickets. * On other server- only three in a week. (UDP needed) * Not a valid MovieID- not allowed to purchase tickets. |
| 8 | SchedulingReservation() | * Display the movie booking schedule of customer * Not a valid customerID- not allowed to login and view anything * Customer does not exits-ok |
| 9 | CancelMovieTicket() | * Cancel tickets on own server-allowed to cancel * Cancel tickets on other servers-allowed to cancel (UDP needed) * Cancel a movie that is not registered - error. * Not a valid MovieID- not allowed to login or Cancel tickets. |
| 10 | SwapMovieTickets() | * New movie has no capaserver – not allowed. * Old Movie doesn’t exist and given new Movie ID exists- not allowed. * Old Movie exists and given new Movie ID doesn’t exist- not allowed. * old MovieID server and new MovieID server equals to user’s server happening in same week - allowed. * old MovieID server and new MovieID server equals to user’s server Not happening in same week – allowed. * old MovieID server does not equal to user’s server, new MovieID server equals to user’s server happening in same week - allowed. * old MovieID server does not equal to user’s server, new MovieID server equals to user’s server not happening in same week – allowed. * old MovieID server equals to user’s server, new MovieID server does not equal to user’s server happening in the same week. Limit == 3 – not allowed to do so. * old MovieID server equals to user’s server, new MovieID server does not equal to user’s server happening in the same week. Limit < 3 – allowed. * old MovieID server equals to user’s server, new MovieID server does not equal to user’s server not happening in same week. limit == 3 – not allowed. * old MovieID server equals to user’s server, new MovieID server does not equal to user’s server not happening in same week. limit <3 - allowed. * old MovieID server does not equal to user’s server, new MovieID server does not equal to user’s server happening in the same week. limit < 3 – allowed. * old MovieID server does not equal to user’s server, new MovieID server does not equal to user’s server happening in the same week. limit == 3 - allowed * old MovieID server does not equal to user’s server, new MovieID server does not equal to user’s server not happening in the same week. limit < 3 - allowed. * old MovieID server does not equal to user’s server, new MovieID server does not equal to user’s server not happening in the same week. limit == 3 – not allowed |