Distributed System Design

COMP 6231 – Winter 2024

Concordia University

Department of Computer Science and Software Engineering

Instructor: R. Jayakumar

Distributed Health Care Management System (DHMS) using JAVA IDL
Assignment – 3

By: Rushi Donga

ID: 40269583

INDEX

OVERVIEW

One of the most important needs in the healthcare industry that the Distributed Health Care Management System (DHMS) helps with is the coordination of medical staff from different hospitals that are part of the Medicare system.

The main goal of DHMS is to develop a platform that simplifies the administration of medical professionals' availability and gives administrators a user-friendly interface for scheduling appointments. Patients also gain from a smooth experience that makes it simple for them to make appointments across multiple cities and specializations.

DHMS operates across three distinct hospitals situated in Montreal (MTL), Quebec (QUE), and Sherbrooke (SHE). The system caters to two primary stakeholders:

1. Admins: Each hospital is equipped with an administrator responsible for overseeing appointment management tasks such as opening or closing appointment slots.
2. Patients: Patients have the flexibility to book or cancel appointments across any hospital within the network.

DHMS uses cutting-edge technologies, most notably Java Webservices, to guarantee efficient and safe client-server communication. This technology adds to the efficacy of the system by enabling real-time updates and improving appointment management overall.

Essentially, DHMS meets a crucial need in the healthcare industry by enhancing accessibility to healthcare services and streamlining the coordination of medical professionals.

SYSTEM ARCHITECTURE

The DHMS operates across three distinct hospital servers:

- Montreal (MTL)
- Quebec (QUE)
- Sherbrooke (SHE)

There is only one Admin user per hospital that is in charge of the scheduling of appointments. The hospital's area code (MTL, QUE, or SHE), the admin user type (A), and a four-digit number (e.g., MTLA1234) are combined to create the admin users' user IDs, which allow them to be uniquely identified.

Patients within the DHMS are identified by their eight-digit user IDs, which consist of the hospital's area code, the user type (P for Patient), and a four-digit number unique to each patient (e.g., QUEP1234).

Appointment scheduling within the DHMS offers three distinct types of appointments: Physician, Surgeon, and Dental. Patients have the flexibility to book appointments in three different time slots: Morning (M), Afternoon (A), and Evening (E).

The 10-digit code format used to create appointment IDs consists of the three-digit area code of the hospital, one-digit code designating the time slot (M, A, or E), and six-digits representing the appointment date in the pattern DDMMYY. For example, the appointment ID MTLM100224 might be associated with a scheduled morning appointment at the Montreal hospital on February 10, 2024.

The DHMS enforces specific limitations in order to preserve system integrity and maximize appointment scheduling. Patients are only able to schedule up to three appointments in other regions in a single week, but they are free to schedule as many appointments as they like inside their own zone. This restriction avoids overbooking and maximizes resource allocation while guaranteeing equitable access to appointments for all hospitals connected to the DHMS network.

FUNCTIONALITY

- **Admin-Specific Functions:**

1. addAppointment():

- Allows admins to add appointment slots for a particular hospital, date, and time slot.
- Admins can only add appointment slots to their respective hospitals.

2. removeAppointment():

- Enables admins to remove appointment slots for a specific day.
- If a patient has booked an appointment for the slot being removed, the system automatically transfers their appointment to the next available slot.

3. listAppointmentAvailability():

- Provides admins with a list of all appointment slots, along with their capacity and remaining slot size.
- Implemented using UDP/IP server-server communication for efficient data exchange between servers.

- **Patient-Specific Functions:**

1. bookAppointment():

- Allows patients to book appointments in any hospital where slots are available.
- Patients can book any number of appointments in their own city, but are limited to a maximum of three appointments in other cities within a week.
- Patients cannot book more than one appointment of the same type in a single day.

2. getAppointmentSchedule():

- Enables patients to view all appointments booked by them.

3. cancelAppointment():

- Allows patients to cancel appointments they have previously booked.

4. SwapAppointment():

- Patients can swap a booked appointment with another appointment. But first it will check these two conditions:
- 1) The old appointment is booked, 2) New appointment slot is available. If this both conditions are satisfied, add and cancel operation will be performed.

Note: Admins have access to client operations, but patients do not have access to admin operations.

IMPLEMENTATION

- Client-Server Communication is done by SOAP based JAVA Webservices:
  o Montreal service address:
  http://localhost:8080/montreal?wsdl o Quebec service
  address: http://localhost:8080/quebec?wsdl o Shaerbrooke
  service address: http://localhost:8080/sherbrooke?wsdl o
  @WebService(endpointInterface =
  "com.web.service.WebInterface")

- Server-Server Communication:
- Implemented using UDP/IP socket programming.
- UDP ports:
  - Montreal: 6666 o Quebec: 7777
  - Sherbrooke: 8888

• Concurrency:

- Each server runs in a separate thread to ensure concurrency and efficient handling of multiple requests.
- Added a concurrency test to check whether system is thread-safe or not?

• Code Organization:

- Implemented a single interface file to reduce code redundancy across servers.
- Utilized a helper.java file to list commonly used methods and properties, reducing code duplication.

• Logging:

- Each server maintains a separate log file to track all user operations, ensuring accountability and traceability.

• Challenges:

- Handling the transfer of patient appointments to the next available slot when an admin invokes the `removeAppointment()` method.
- Ensuring the rollback operation when 'swapAppointment()' operation fails.

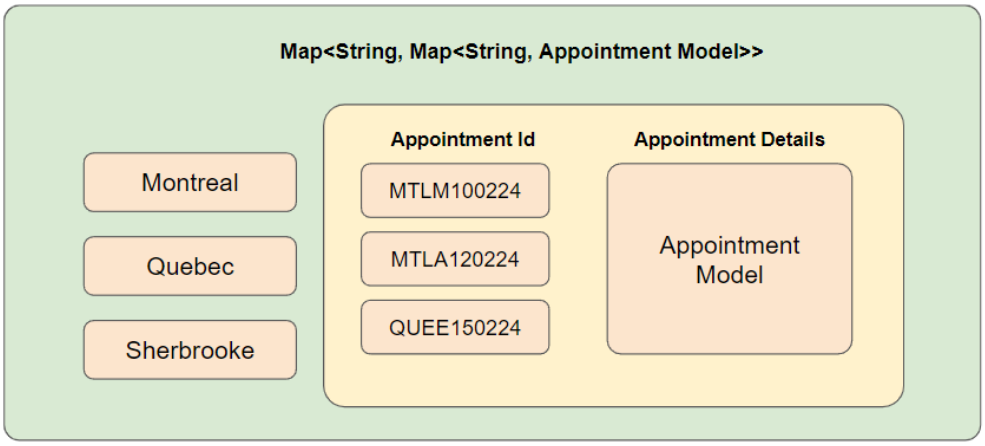
• Key Considerations:

- The implementation focused on robust server-server communication to facilitate accurate appointment management across hospitals.
- Addressing concurrency issues and minimizing code redundancy were integral aspects of the implementation process.

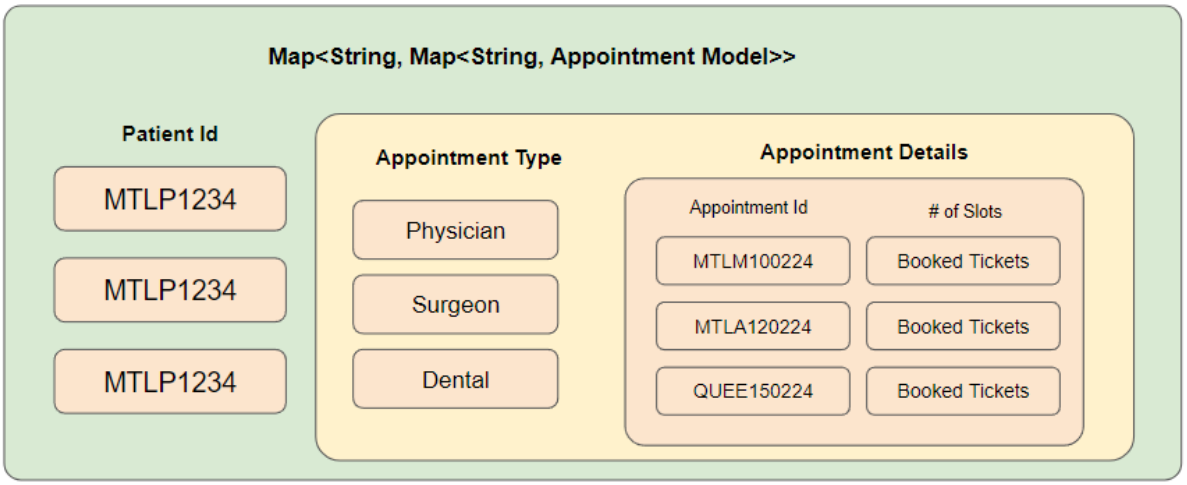
DATA STRUCTURE

All the data is stored in the form of concurrent HashMap.

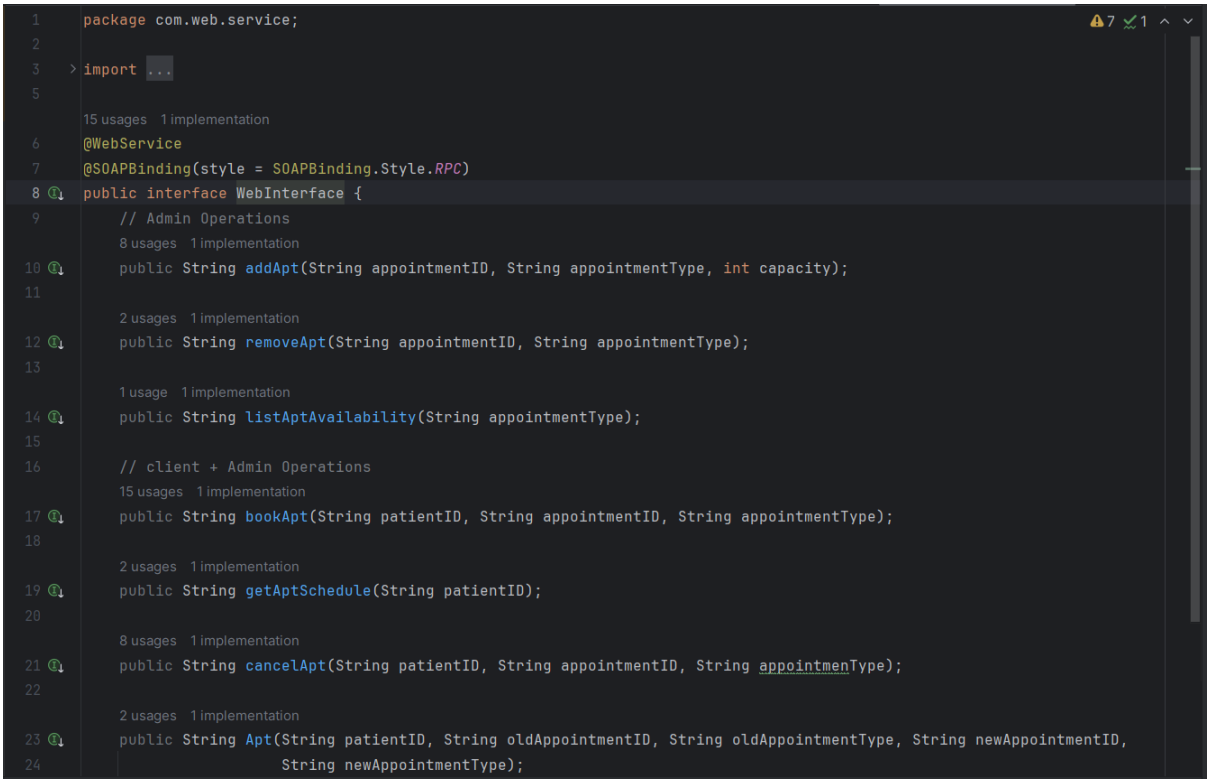
1. All Appointments:



2. User Appointments:



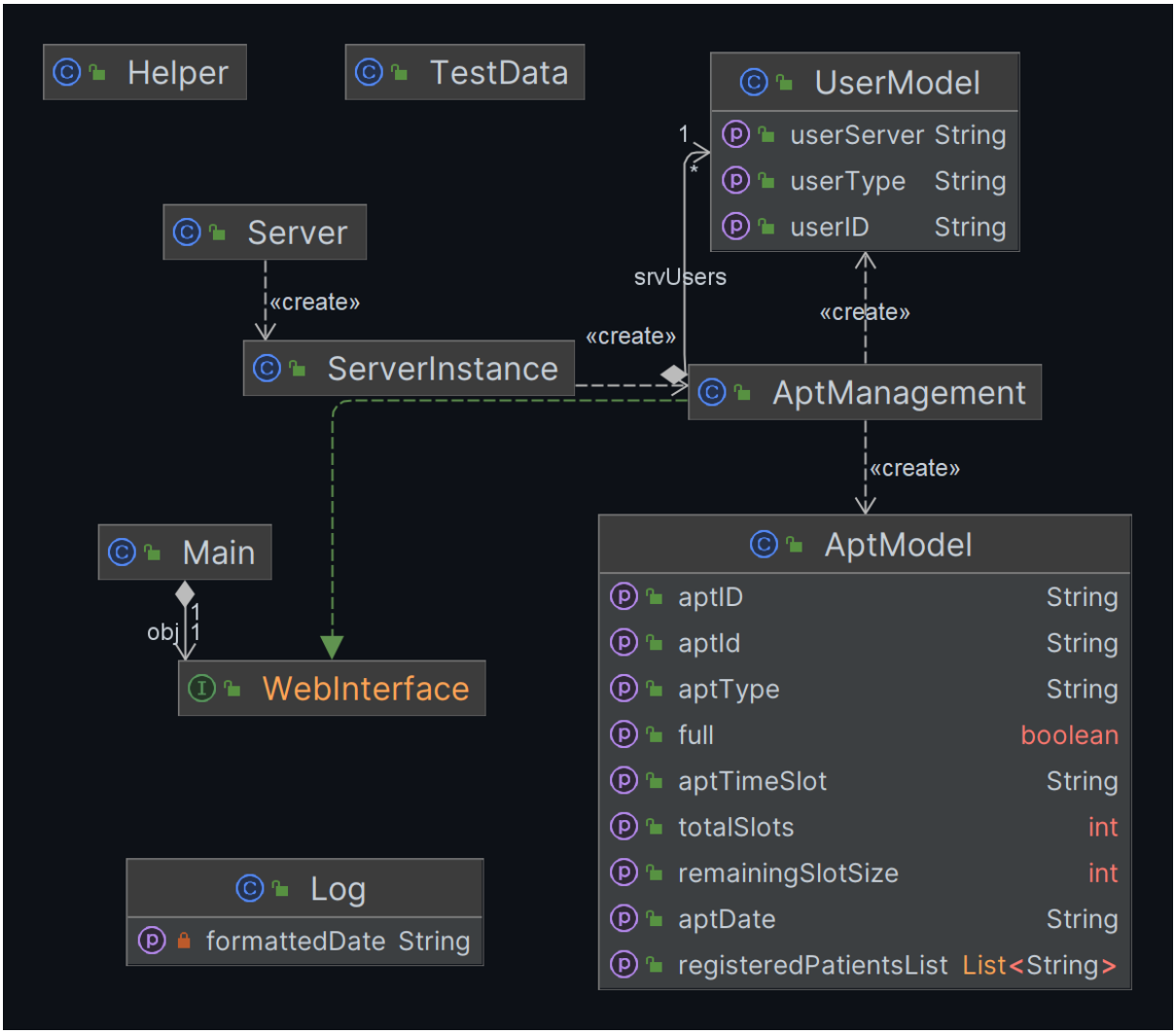
IDL INTERFACE DEFINITION



```java
package com.web.service;

import ...

15 usages  1 implementation
@WebService
@SOAPBinding(style = SOAPBinding.Style.RPC)
public interface WebInterface {
    // Admin Operations
    8 usages  1 implementation
    public String addApt(String appointmentID, String appointmentType, int capacity);

    2 usages  1 implementation
    public String removeApt(String appointmentID, String appointmentType);

    1 usage  1 implementation
    public String listAptAvailability(String appointmentType);

    // client + Admin Operations
    15 usages  1 implementation
    public String bookApt(String patientID, String appointmentID, String appointmentType);

    2 usages  1 implementation
    public String getAptSchedule(String patientID);

    8 usages  1 implementation
    public String cancelApt(String patientID, String appointmentID, String appointmenType);

    2 usages  1 implementation
    public String Apt(String patientID, String oldAppointmentID, String oldAppointmentType, String newAppointmentID,
                      String newAppointmentType);
```

UML DIAGRAM


UserModel
detectUserServer() String
setUserID(String) void
setUserType(String) void
setUserServer(String) void
getUserID() String
getUserServer() String
detectUserType() String
toString() String
getUserType() String
srvUsers
«create»
Main
patient(String) void
startConcurrencyTest() void
getServerID(String) String
admin(String) void
start() void
main(String[]) void
init() void
1 obj
WebInterface
getAptSchedule(String) String
listAptAvailability(String) String
Apt(String, String, String, String, String) String
removeApt(String, String) String
bookApt(String, String, String) String
addApt(String, String, int) String
cancelApt(String, String, String) String
AptManagement
removeApt(String, String) String
listAptAvailability(String) String
removeAptIfExists(String, String, String) boolean
listAptAvailabilityUDP(String) String
isAptofThisServer(String) boolean
patientHasEvent(String, String, String) boolean
onTheSameWeek(String, String) boolean
addPatientAndApt(String, String, String) void
sendUDPMessage(int, String, String, String, String) String
getNextSameApt(Set<String>, String, String) String
bookApt(String, String, String) String
Apt(String, String, String, String, String) String
isUserofThisServer(String) boolean
addApt(String, String, int) String
cancelApt(String, String, String) String
addAptTypeAndApt(String, String, String) void
addPatientsToNextSameApt(String, String, List<String>) void
exceedWeeklyLimit(String, String) boolean
aptExists(String, String) boolean
getAptSchedule(String) String
checkUserExists(String) boolean
addNewPatientToUsers(String) void
addNewApt(String, String, int) void
removeAptUDP(String, String, String) String
Log
deleteALogFile(String) void
getFormattedDate() String
userLog(String, String, String, String) void
serverLog(String, String, String, String, String) void
userLog(String, String) void
getFileName(String, String) String
serverLog(String, String) void
Helper
getToDoOperation(String) void
getAppointmentType(Scanner) String
identifyTimeSlotFromAptID(String) String
getUserType(String) String
getUDPServerPort(String) int
identifyAptDate(String) String
identifyServer(String) String
getAptID(Scanner) String
getAppointmentCapacity(Scanner) int
TestData
main(String[]) void
addTestData() void
«create»
AptModel
toString() String
getRemainingSlotSize() int
getAptID() String
setTotalSlots(int) void
getAptDate() String
setAptDate(String) void
getRegisteredPatientsList() List<String>
addPatientToRegisteredPatientsList(String) int
getAptTimeSlot() String
getTotalSlots() int
removePatientFromRegisteredPatientsList(String) boolean
setAptType(String) void
getAptType() String
setAptTimeSlot(String) void
isFull() boolean
setAptId(String) void
«create»
Server
main(String[]) void
«create»
ServerInstance
listenForRequest(AptManagement, int, String, String) void


Helper
TestData
UserModel
userServer String
userType String
userID String
1
*
Server
«create»
srvUsers
«create»
«create»
ServerInstance
AptManagement
«create»
Main
obj
1
1
WebInterface
AptModel
aptID String
aptId String
aptType String
full boolean
aptTimeSlot String
totalSlots int
remainingSlotSize int
aptDate String
registeredPatientsList List<String>
Log
formattedDate String

TEST SCENARIOS

#	Type of Test	Scenario	Cases
1	Log In	Username	AdminID PatientID
2	Admin operation	addAppointment()	1.Invalid appointmentID => appointment not added 2.valid appointmentID => appointment created 3.duplicate appointID => appointment exist message 4.otehr server appointmentID => appointment not created
3		removeAppointment()	1.Invalid appointmentID => appointment not removed 2.valid appointmentID => appointment removed 3.appointmetID of other server => error message 4.appointmentID where patient registered => patient added to next appointment
4		listAppointmentAvailability()	1.when appointment slots are available => appointment list shown 2. when no slots available => blank list
5	Admin + Patient Operation	bookAppointment()	1.when slots are available => appointment booked 2.when slots are full=> slots full message 3.no slots open => no slot available error 4.in other server => appointment booked

			5.max. 3 appointments in other server test => successful 6. invalid appointmentID => no appointment booked
6		getAppointmetSchedule()	1.patient booked appointments => list of appointments 2.patient never booked appointment=> no appointment found message
			3.appointment from other servers => visible
7		cancelAppointment()	1.cancle on own server => success 2.cancle on other server => success 3.no appointment booked => no appointment found error message 4.invalid appointment id => no cancel operation
8	File log	clientLog()	1.user login => success 2.user operation => success 3.user logout => success
9		serverLog()	1.server starts => success 2.user operation => success 3.server-server communication => success

10	Admin + Patient Operation	SwapAppointment()	1.new appointment has no capacity => no swap 2.Old appointment doesn't exists and new Appointment exists => no swap 3.Old appointment exists, and new Appointment doesn't exists => no swap 4.old appointment city equals to patient's city and new appointment city equals to patient's city => success 5.old appointment city not equals to patient's city and new appointment city equals to patient's city. => success 6.old appointment city equals to patient's city and new appointment city not equals to Patient's city => success 7.old appointment city not equals to patient's city and new appointment city not equals to patient's => success 8.old appointment city not equals to patient's city and new appointment city not equals to patient's city (Limit 3 reached) => no swap

Table 4.0 Test Scenarios and results