Distributed System Design

Assignment 3

Kasra Karaji
40255099

Overview

This project mainly consists of two sections a Server section and a Client section.

Server

The main functionalities of the server are implemented in DHMS.java that implements DHMSInterface.java which is the interface that includes the appropriate annotations like @WebMethod for the functions and @WebService and @SOAPBinding for the interface and the DHMS class.
DHMS object is instantiated inside HospitalServer.java which makes the object accessible in the name space.
There are three hospital servers in total so we have three instances of HospitalServer inside the Server class:

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

Client

On the other hand, we have the client section which handles inputs from the user and invokes the appropriate methods using web services. On the client side we use the following commands to get the binded object to the address. Then we can access it on the client-side.

```
URL url = new URL( spec: "http://localhost:8080/"+ user.getCity().toString().toLowerCase()+"?wsdl");
QName qName = new QName( namespaceURI: "http://DHMS/", localPart: "DHMSService");
Service service = Service.create(url, qName);
DHMSInterface dhms = service.getPort(DHMSInterface.class);
```

Communications

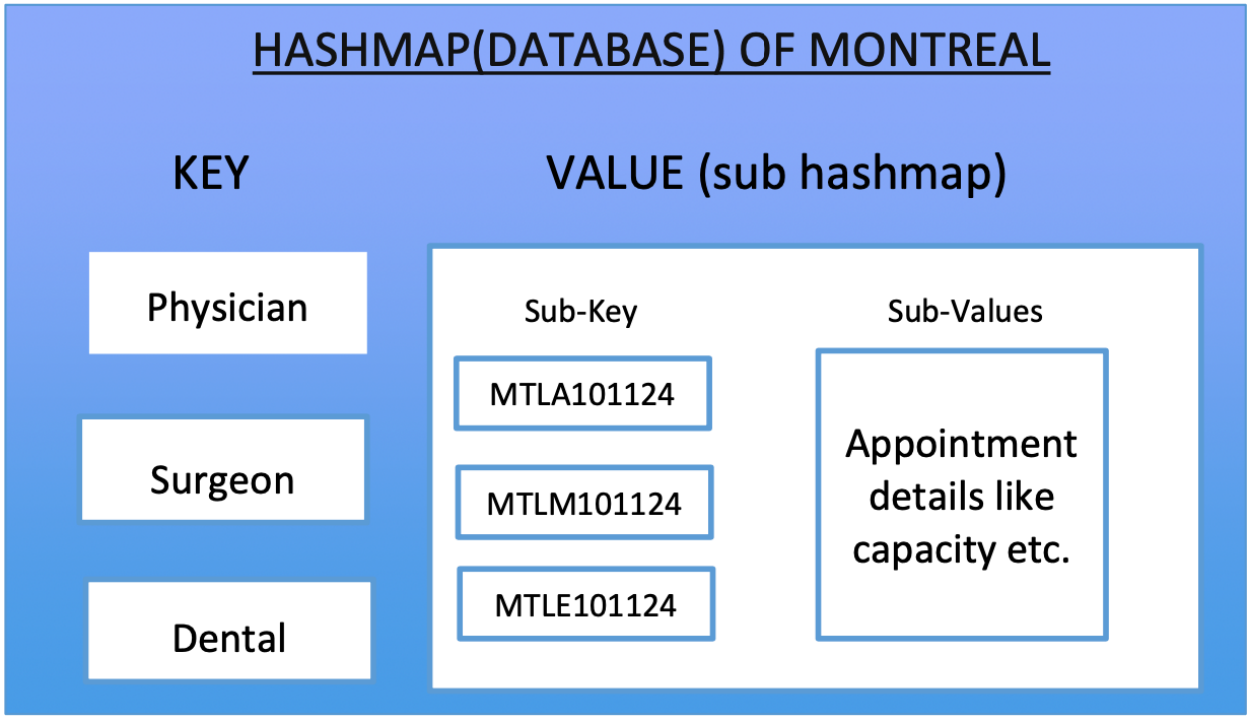
As mentioned earlier, the Client-Server communication is facilitated by web services but the inter-server communication is enabled using UDP socket programming.

Design Techniques

- Separate thread is used for listening to UDP addresses to enable the server functionalities to run concurrently.
- The hardest method to implement was swapAppointment since it had to be atomic also it had many possible outcomes that needed handling.
- Object-Oriented Programming is used for better readability, Modularity and encapsulation. (For example we have different classes and enums for appointments and users)
- Concurrent Hashmaps are used to ensure maximum concurrency.
- A separate thread is assigned for each server so they can run concurrently.

Data Structures

A class is defined for appointments and users to store their values.
There are three things that needed to be stored the appointments, the patients and the admins. Appointments are stored on the server that they belong to using concurrent hashmaps as shown by the structure bellow:



Patients and Admins are stored in separate arrayLists in the respective servers for easy access.

Functions

There are two types of functions implemented on this server ones only accessible by the admin and the other's accessed by both the admins and the patients.

Admin Functions

- ***addAppointment* (*appointmentID, appointmentType, capacity*):** This function adds a new appointment to the available appointments for booking.
- ***removeAppointment* (*appointmentID, appointmentType*):** This function removes an appointment from the existing appointment hashmap and also removes the appointments from the patients that have booked it and books the earliest appointment available after that appointment in the appointment city for the patient.
- **listAppointmentAvailibility(appointmentType):** This function lists all the available appointments of a certain type from all the cities.

Assignment 3

Patient Functions

- ***bookAppointment* (*patientID, appointmentID, appointmentType*):** This appointment is for booking an appointment for a patient.
- ***getAppointmentSchedule* (*patientID*):** This function gets all the appointments for a certain patient and displays them to the client.
- **cancelAppointment(patientID, appointmentID, appointmentType):** This function is used to cancel an appointment which a patient had previously booked.
- **swapAppointment(patientID, oldAppointmentID, oldAppointmentType, newAppointmentID, newAppointmentType):** This function is for swaping an appointment for another which is atomic meaning that either the old appointment is removed and the new appointment is added or none are done.

Also there is the login function which is used by both the admins and the users and for a user to get registered in the system they have to log in one time.

Test Cases

Here are some test scenarios that are described to test the functionality of the code:

login:

- Admin login: mtla1234
- Patient login: mtlp1234
- Invalid city: thrp1234 -> error
- Invalid role: mtlk1234 -> error
- Invalid numericid: mtlajjjj -> error

addAppointment:

- Admin adds appointment for his own city
- Admin adds appointment for another city -> error

removeAppointment:

- Appointment doesn't exist -> error
- Appointment exists and is wiped from all the patients and the server
- Appointment of the same type exists in the city after the removed appointment and the patients for that appointment get rebooked for that appointment.

listAppointmentAvailibility:

- Appointments from all servers are shown.
- Invalid appointment type -> error

bookAppointment:

- The patient can book appointment from his city
- The admin can book an appointment for a patient from his city
- The admin can't book an appointment for a patient from another city

- The patient can book from other cities
- Admin can book from other cities
- Patient cannot book with conflict with his other appointments
- Patient cannot book more that 3 appointments outside his city

getAppointmentSchedule:

- Patient can view his appointments from all the cities
- Admin can view appointments for his city's patients

cancelAppointment:

- Patient can cancel appointments that he has
- Patient can't cancel appointments that he doesn't have
- Admin can cancel appointments for the patients of his own city

swapAppointment:

- Swap appointment for an appointment in the same city as the patient
- Swap appointment for an appointment in a different city as the patient
- Swap appointment for non-existent old appointment
- Swap appointment for full new appointment
- Swap appointment for non-existent new appointment