Assignment 1

Distributed Health Care Management System (DHMS) using Java RMI

COMP 6231 – Winter 2024

DISTRIBUTED SYSTEM DESIGN

Concordia University

Department of Computer Science and Software Engineering

Instructor: - R. Jayakumar

Submitted By: - Janmitsinh Gajendrasinh Panjrolia

Student ID: - 40294468

Table of Contents

Overview

Goal: - The goal of a Distributed Health Care Management System (DHMS) is to manage medical appointments.

Here, we have two types of users of the system:

1] By Hospital admins: - Admins use DHMS to manage the information about medical appointments. To perform add, remove, list appointment and also patient actions.

2] By Patients: - Patients use DHMS to book, cancel and get Appointment Schedules across different hospitals within the Medicare System.

Here, we have three servers which run in three different cities: -

1] Montreal (MTL)

2] Sherbrooke (SHE)

3] Quebec (QUE)

Admin and Patient Functionalities: -

Admin Specific Functions: -

1] addAppoinment(appointmentID, appointmentType, capacity): - Checks if an appointment of the same type already exists, if it does, it should fail and if not, the appointment will add.

2] removeAppointment(appointmentID, appointmentType): - Admin can only remove appointments from their server. If an Appointment was removed, we must book another closest appointment for the patient registered for that doctor.

3] listAppointmentAvailability(appointmentType): - List out all appointments of a given type from all three servers.

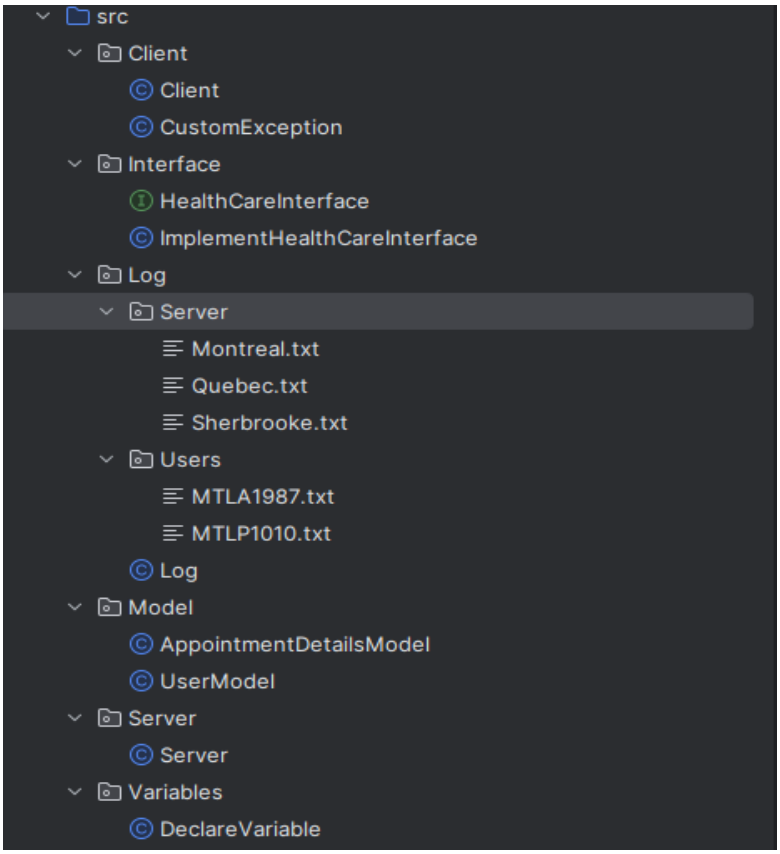
Patient-specific Functions: -

1] bookAppointment(patientID, appointmentID, appointmentType): The patient can also book from other servers with a weekly 3 limit.

2] getAppointmentSchedule(patientID): - Give the appointment schedule.

3] cancelAppointment(patientID, appointmentID): - Patient can remove an appointment from their schedule.

Folder Structure: -



src: The whole code is implemented under the src folder.

Client: The client folder contains two files:

1) Client.java: It implements the client-side functionality of a DHMS. It contains socket initialization, input/output streams, sending/receiving data, handling communication, patient Client methods, admin Client methods and error handling.

2) CustomException.java: This file is used to handle customized exceptions which will be thrown during communication over the network for better readability.

Interface: Interface folder contains two files:

1) HealthCareInterface.java: - It defines the remote interface. It declares the methods that can be invoked remotely by clients on remote objects.

2) ImplementHealthCareInterace.java: - It is responsible for implementing the admin and patient methods declared in the remote interface. It contains actual logic and functionality behind the remote methods. It is Server-Side Processing.

Log: The server folder contains the server log, and the Client folder contains client information.

We must store server and client information about the execution of a program in the log file.

It enhances the security, performance and reliability of the system by giving visibility into its operation.

Model:

- User Model is for encapsulating the data, Interaction with the remote server and logic specific to the client side of the application.
- AppointmentDetailsModel is the core model layer of the application, encapsulating the shared data and business logic used by both client and server files.
- Models manage and manipulate data of an application.

Server:

- It implements server-side functionality for DHMS. It does request handling, concurrency, scalability, and UDP/IP connection.
- Implementation of the remote interface and handling remote method invocations from clients. It enables remote communication between clients and servers.

Variables -> DeclareVariables: Declare Variables file stores the common public variables which are used all over the project.

Important/Difficult Part of this Assignment: -

1. **Implementing Java RMI:** Need to design and implement the Java RMI interface for the server and ensure that remote access calls between the client and the server are handled correctly and reliably.

2. **Managing distributed data:** Need to manage appointment records in HashMap, ensuring consistency and consistency of data for multiple servers for different cities (Montreal, Quebec, Sherbrooke). This requires an understanding of data structures.

3. **Inter-Server Communication:** Use functionality for servers to communicate with each other, using UDP/IP sockets, especially for applications such as booking appointments with hospitals.

Addressing these challenges will require a solid understanding of Java, network programming, data structures, and system design principles.

Architecture:

Distributed Health Care Management System (DHMS) is like a network of interconnected computers designed to handle healthcare-related tasks across multiple cities.

DHMS contains a network of computers, with each computer representing a hospital in a different city.

PatientClient: Hospital patients interact with the system. It lets them do things like booking appointments, checking their appointment schedules, and cancelling appointments.

AdminClient: This is another functionality, but it's for hospital administrators. They use it to manage appointment slots and perform other administrative tasks. Admin can also perform the patient methods.

Each server manages its operations. It listens to requests from patients and administrators, handles those requests, and keeps track of appointments.

There's also a central hub, sort of like a control center, that stores information about all the hospitals in the network. When a patient or administrator wants to do something, they first check in with this central hub. It then directs them to the right hospital based on what they need to do.

Servers (One per City):

- Each server manages the operations for its corresponding server.
- It handles functionalities for appointment management (booking, cancellation, listing availability, etc.).
- Patients and staff use them to book appointments and update them.

Central Repository:

- Store information about all the servers.
- Facilitates the connection between clients and the server based on the operation and user role.

PatientUser:

- Interface for patients to interact with the system.
- Request operations like booking, viewing, and cancelling appointments.
- Communicates with the server via the central repository to perform these operations.

AdminUser:

- Interface for hospital admins to manage appointment slots.

- Performing administrative tasks such as adding or removing appointment slots and checking appointment availability.
- Interacts with the server for all admin-related operations.

Data Storage and Management:

- Utilize HashMap to store key-value pairs and provide insert, delete and other operations.
- Each server maintains a database, and I use hash maps for managing appointment details.
- Handles concurrent access and ensures data consistency.

Access Control:

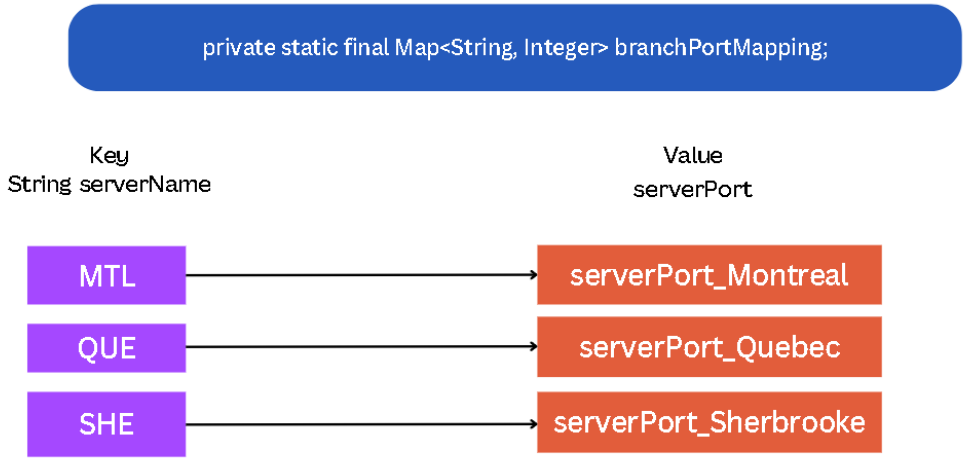
- The system enforces strict role-based access to ensure that patients and admins can only perform operations pertinent to their roles.

Network Communication:

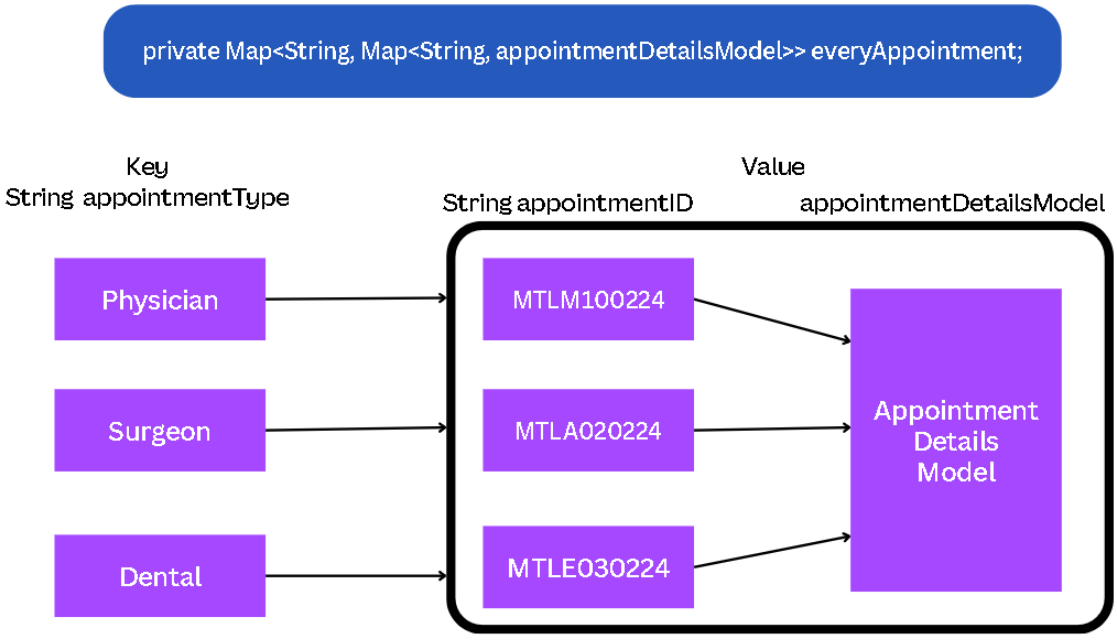
- Utilizes Java RMI for communication between clients and servers.
- Utilizes UDP/IP for server-server communication.
- Ensures efficient and secure data exchange across the distributed environment.

Data Structure [Hash Map]:

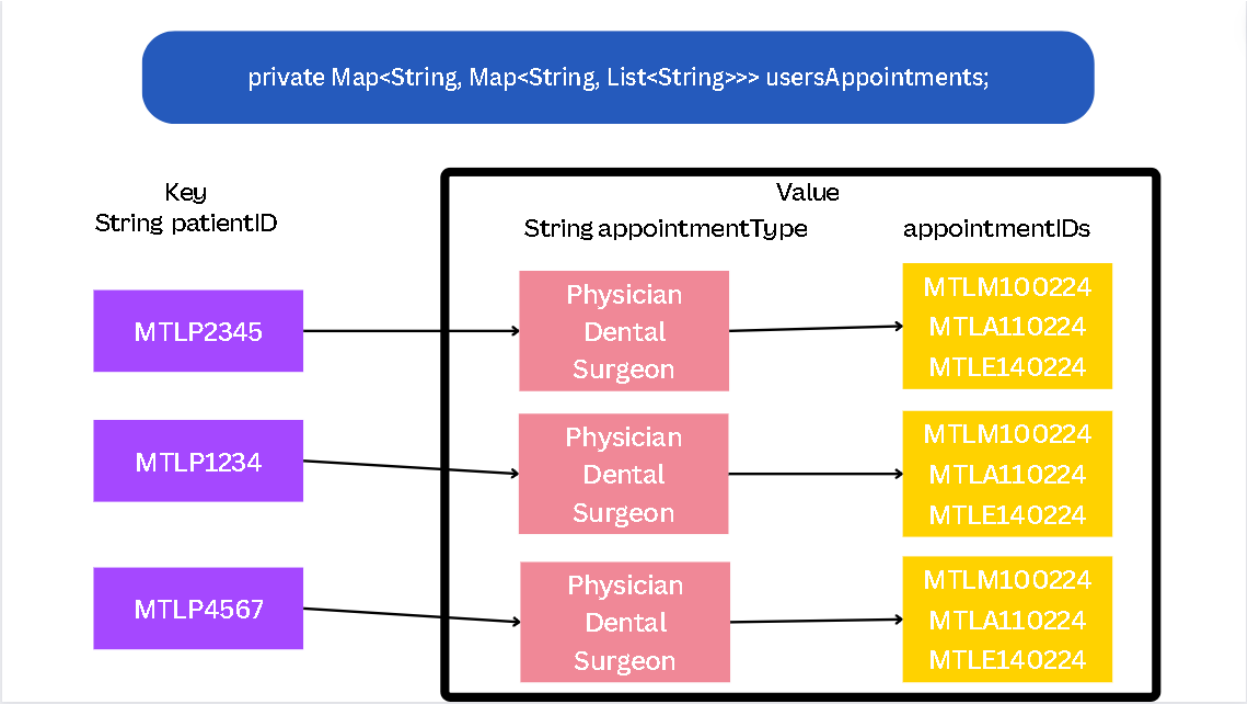
1) key: serverName value: serverPort



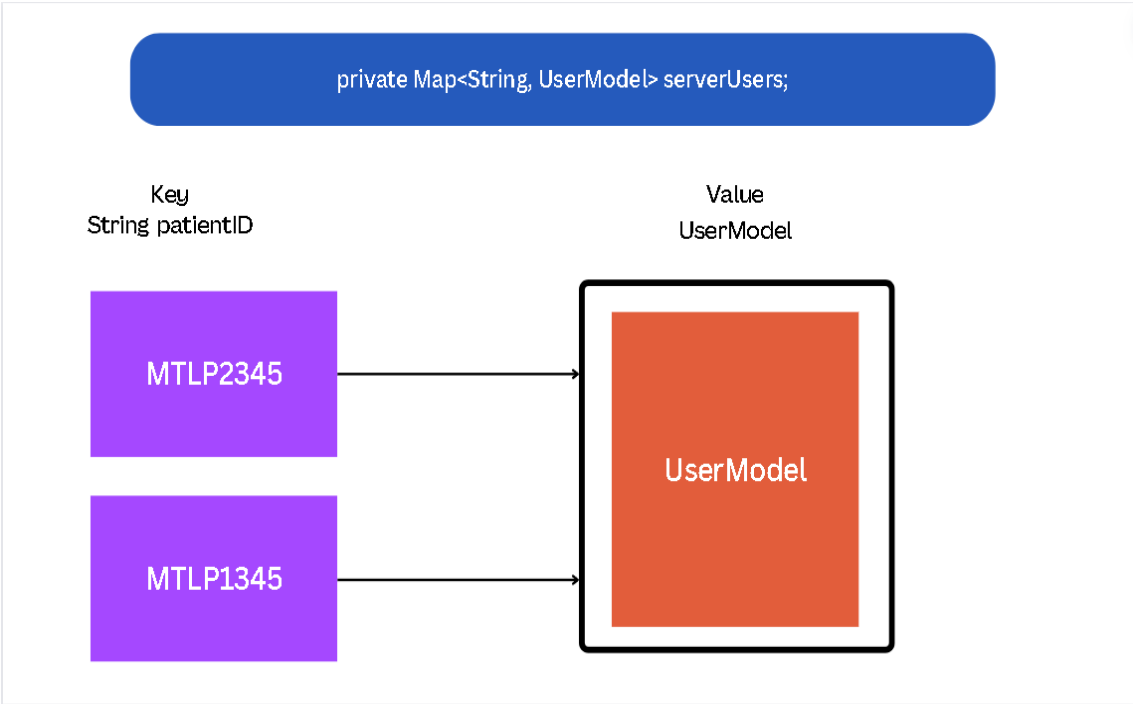
2) Key: AppointmentType Value: <appointmentID, appointmentDetailsModel>



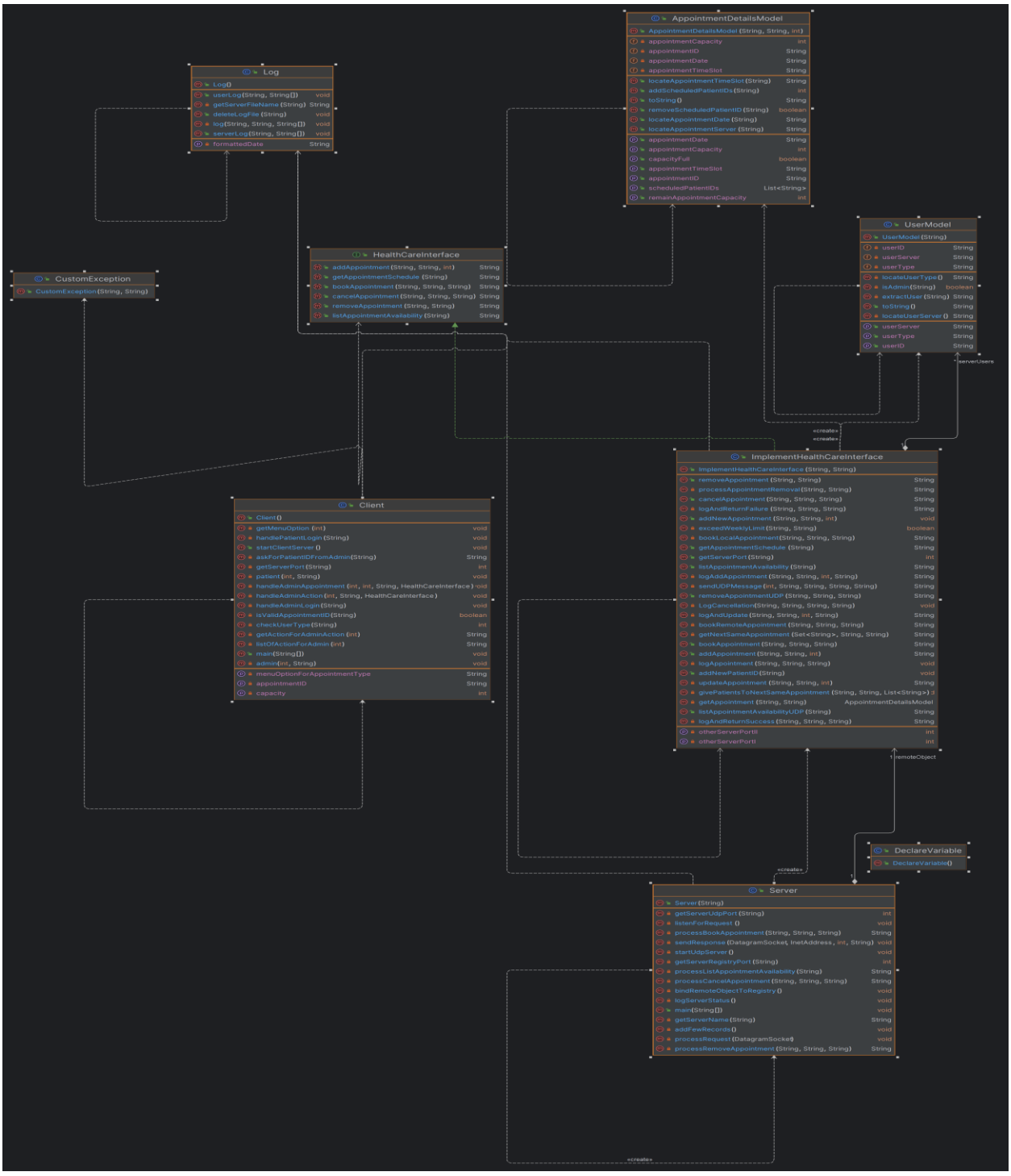
3) Key: patientID value: <appointmentType, appointmentIDs>



4) key: patientID value: userModel



Class Diagram: -



Test Scenarios: -

#	Test Module	Test ID	Test Description	Test Behavior
1	Login	Username	Verify that a user has a valid Appointment AdminID and PatientID	1. Successful login with valid Admin ID. 2. Successful login with valid Patient ID.
2	Menu Items	Logout	The user should be able to Log0ut when entering the number of the Logout button from a menu item	Logout through the menu Item
3	Admin	addAppointment()	1. Test with Invalid AppointmentID 2. Add a New AppointmentID 3. Add an existing appointment with a lower capacity 4. Add an existing appointment with a higher capacity 5. Add duplicate appointments to test duplication 6. Add an appointment from another server to check the error. 7. Add an appointment to the current server.	1. Invalid AppointmentID - Trying to add an Appointment with an invalid ID, which should fail 2. New AppointmentID - Successfully add new AppointmentID 3. Existing AppointmentID (Lower Capacity) -not allowed 4. Existing AppointmentID (Higher Capacity) - Capacity update for existing AppointmentID 5. Duplicate Appointment - Ensuring duplicate Appointments are not created. 6. AppointmentID from other Servers -> from other servers not permitted
4		removeAppointment()	1. Attempt to remove an Appointment with an invalid ID. 2. Trying to remove a non-existent Appointment. 3. Removing an Appointment without any registered participants. 4. Removing an Appointment with registered participants and handling reallocation. 5. Add Appointments from other servers.	1. Rejecting invalid AppointmentID 2. AppointmentID does not exist 3. Appointment not registered - Delete Appointments with no registrations. 4. Appointmentregistered - Removed Appointment + registered to same AppointmentType if possible 5. AppointmentID from other servers – return fails.
5		listAppointmentAvailability()	1. Give a list of available appointment	1. Showing a list of all Appointments of a given type from all three servers 2. Display only available Appointment types
6		Ask for patientID	Admin can perform a patient method.	1. Ensuring successful access to patient methods.
7		bookAppointment()	1. Book an appointment on the current server 2. Book appointments on other servers with a weekly limit of three.	1. booking only allowed on own server 2. Booking is not allowed if the Appointment is full

	Admin + Patient		3. Book the same appointmentID which has the same appointment type. 4. Book with an invalid AppointmentID. (not able to book) 5. Book the same appointmentID but a different appointmentType. (not able to book)	3. Limits booking on other servers to three per week 4. Invalid booking AppointmentID is not permitted.
8		getAppointmentSchedule()	1. Get the booking schedule 2. Try with an invalid patientID. 3. Should give all appointments of patients in the current server.	1. displaying the booking schedule of a patient. 2. errors are shown for an invalid patientID.
9		cancelAppointment()	1. Cancel a booked Appointment on the current server. 2. Cancel a booked Appointment on other servers. 3. Canceling a non-registered Appointment. 4. Try cancellation of Appointments with an invalid AppointmentID.	1. Successfully cancelling a booked Appointment allowed on own server 2. Successfully cancelling a booked Appointment on other servers 3. Display error for cancelling unregistered Appointment 4. Ensuring cancellation of Appointments with an invalid AppointmentID is not allowed.