**CA**

**Distributed**

**Systems**

**Mengzhao Xu**

**x22198491**

1. Introduction

The ‘SmartHealth’ monitoring system utilizes gRPC for efficient RPC communication. It provides functionality for patient monitoring, prescription management, and device monitoring services.

1. Service Definitions

2.1 Patient Monitoring Service

Purpose: Monitor the health status of patients.

Methods: getPatientStatus(PatientRequest) returns (PatientStatus)

2.2 Prescription Service

Purpose: Manage patient prescriptions.

Methods: addPrescription(Prescription) returns (PrescriptionResponse)

2.3 Equipment Monitoring Service

Purpose: Monitor the operational status of medical equipment.

Methods: getEquipmentStatus(EquipmentRequest) returns (EquipmentStatus)

1. Service Implementations

**PatientMonitoringServiceImpl:** Implements the method to get the health status of a patient. The demo returns a "Stable" status for every request.

**PrescriptionServiceImpl:** Implements the method to add a prescription. The demo returns a "Prescription Added!" for every message.

**EquipmentMonitoringServiceImpl:** Implements the method to get the status of medical equipment. The demo returns an "Operational" status for every request.

1. Naming Services

When we use gRPC, each service you create has a default way of being found and named. In my code, every service has a special tag or identifier. For example, the Patient Monitoring Service has a tag: ‘PatientMonitoringServiceGrpc.PatientMonitoringServiceImplBase’. This tag helps our project know which service we're talking about and makes sure we're communicating with the right one.

1. Remote Error Handling & Advanced Features

5.1 Error Handling

My project uses the ‘StatusRuntimeException’ provided by gRPC to handle RPC related errors. This exception provides a detailed status and description that helps in diagnosing the cause of the failure.

5.2 Advanced Features

Streaming: Our project uses gRPC’s streaming capability via StreamObserver to send and receive multiple messages within a single RPC call.

Concurrency: Services are designed to handle multiple requests concurrently, ensuring optimal resource utilization and performance.

1. Client - Graphical User Interface (GUI)

**Design:** A simple Swing-based GUI is provided for end-users to interact with the services.

**Functionalities:**

Check Patient Status: Queries the ‘CheckPatientStatus’ service.

Add Prescription: Utilizes the ‘AddPrescription’ service.

Check Equipment Status: Queries the ‘CheckEquipmentStatus’ service.

Error Display: The GUI displays errors directly on the interface, to ensure immediate feedback to the user.

1. GitHub

The full set of files and instructions for the 'SmartHealth' monitoring system can be found on the GitHub. To ensure efficient development, we consistently using techniques like: regular updates, creating separate lines of development and suggesting code changes.

1. Conclusion

My "SmartHealth" monitoring system demonstrates the capabilities of gRPC for remote interaction. Its well-organized structure and user-friendly interface make it easy to use.

If the system continues to be improved in the future, I hope it can provide real-time updates of health data, connect with smart devices, and so on.