Distributed Systems

April 2023

SMART HEALTH SYSTEM

REPORT

Name: Hélio Bispo Santana

Student ID: x22100431

**Service Definitions.................................................................................................................3**

# **Service Definitions**

In this system, I implemented 3 services: Smart Appointment Scheduling, Triage, and Visitor Checkin.

## **Smart appointment service**

With this service, appointments can be recorded. Patient details and appointment times are provided. We have a gRPC service called "AppointmentService" which consists of three methods: "CreateAppointment", "ModifyAppointment", and "CancelAppointment". The service is defined using the proto3 syntax.

The "CreateAppointment" and "ModifyAppointment" methods both take a stream of "Appointment" messages as input and return a single "Appointment" message as output. These methods are used for creating and modifying appointments between a patient and a doctor. The "Appointment" message contains several fields, such as "patient\_id", "doctor\_id", "start\_time", and "start\_date", which describe the appointment details.

The "CancelAppointment" method takes a "CancelAppointmentRequest" message as input and returns a "CancelAppointmentResponse" message as output. This method is used for canceling an appointment. The "CancelAppointmentRequest" message contains the "appointment\_id" field, which specifies the ID of the appointment to be canceled.

Both the "CreateAppointmentResponse" and "ModifyAppointmentResponse" messages contain a "success" field, which indicates whether the operation was successful or not, and a "message" field, which contains a human-readable message that explains the result of the operation. syntax = "proto3";

service AppointmentService {

rpc CreateAppointment(stream Appointment) returns (Appointment) {} rpc ModifyAppointment(stream Appointment) returns (Appointment) {} rpc CancelAppointment(CancelAppointmentRequest) returns

(CancelAppointmentResponse) {}

}

message Appointment {

string patient\_id = 1; string doctor\_id = 2; string name = 5; string gender = 6; string start\_time = 3; string start\_date = 4;

}

message CreateAppointmentResponse {

bool success = 1; string message = 2;

}

message ModifyAppointmentResponse {

bool success = 1; string message = 2;

}

message CancelAppointmentRequest {

string appointment\_id = 1;

}

message CancelAppointmentResponse {

bool success = 1; string message = 2;

}

## **Triage service**

The TriageService is a service that provides procedures for checking in patients and setting their status, and adjusting procedure steps.

The protofile defines the service with three RPCs: CheckInTriageProcedure, SetStatusDiseasePatient, and AdjustProcedureSteps.

CheckInTriageProcedure takes in a patient ID and returns a message.

SetStatusDiseasePatient takes in a patient ID, temperature, blood pressure, and symptoms, and returns a message.

AdjustProcedureSteps takes in a stream of ProcedureSteps (patient ID and step) and returns a message.

The response messages for all RPCs contain a simple message. syntax = "proto3";

service TriageService {

rpc CheckInTriageProcedure(CheckInTriageProcedureRequest) returns

(CheckInTriageProcedureResponse) {} rpc SetStatusDiseasePatient(stream SetStatusDiseasePatientRequest) returns

(SetStatusDiseasePatientResponse) {} rpc AdjustProcedureSteps(stream ProcedureStep) returns

(AdjustProcedureStepsResponse) {}

}

message CheckInTriageProcedureRequest {

string patient\_id = 1;

}

message CheckInTriageProcedureResponse {

string message = 1;

}

message SetStatusDiseasePatientRequest {

string patient\_id = 1; double temperature = 2; double blood\_pressure = 3; string symptoms = 4;

}

message SetStatusDiseasePatientResponse {

string message = 1;

}

message ProcedureStep {

string patient\_id = 1; int32 step = 2;

}

message AdjustProcedureStepsResponse {

string message = 1;

}

## **Checkin service**

The VisitorsCheckInService is a service that allows visitors to check-in at a hospital. It is defined in the provided protofile using the proto3 syntax. The service offers three remote procedure calls: CreateNewVisitor, VisitorStatus, and InpatientVisitors.

The CreateNewVisitor RPC creates a new visitor in the system by taking a request containing the visitor's username, password, and other relevant fields. The response indicates whether the visitor was successfully created or not. The VisitorStatus RPC checks the status of a visitor by taking a request containing the visitor's ID and returning a boolean indicating whether the visitor is allowed to visit.

The InpatientVisitors RPC allows hospital staff to onboard visitors for inpatient visits. It takes a stream of requests containing the inpatient and visitor IDs and returns a boolean indicating whether the visitor has been successfully onboarded. The protofile also includes the relevant message types for each RPC request and response.

syntax = "proto3";

service VisitorsCheckInService {

rpc CreateNewVisitor(CreateNewVisitorRequest) returns (CreateNewVisitorResponse) {}

rpc VisitorStatus(stream VisitorStatusRequest) returns (VisitorStatusResponse) {} rpc InpatientVisitors(stream InpatientVisitorRequest) returns (InpatientVisitorResponse) {}

}

message CreateNewVisitorRequest {

string username = 1; string password = 2;

// other relevant fields

}

message CreateNewVisitorResponse {

bool success = 1; string message = 2;

}

message VisitorStatusRequest {

string visitor\_id = 1;

}

message VisitorStatusResponse {

bool allowed = 1;

}

message InpatientVisitorRequest {

string inpatient\_id = 1; string visitor\_id = 2;

}

message InpatientVisitorResponse {

bool onboarded = 1;

}

# **Service implementations**

The AppointmentService provides methods for creating and modifying appointments, as well as canceling them. The service keeps track of all created appointments in an ArrayList.

The CheckinService provides methods for creating new visitors and checking their status. Additionally, there is a method for getting information about inpatient visitors.

The TriageService provides methods for performing triage procedures on patients, setting their disease status, and adjusting the steps of the triage procedure.

All three services define startServer and stop methods for starting and stopping the gRPC server. The startServer method takes a port parameter and starts the server with insecure server credentials on that port. The stop method shuts down the server after waiting for up to 30 seconds for any ongoing requests to complete.

Each service implements a set of gRPC methods defined in a proto file. The

StreamObserver interface is used for handling requests and responses that involve streams of data.

# Naming services

Text

Description automatically generated

The ServiceRegistration class implements the Runnable interface, which means it can be executed by a thread. The run() method is the main method that will be executed when the thread is started.

In the run() method, the Logger class is used to log any messages. Then, an instance of JmDNS is created by calling the create() method and passing in the local host address as an argument. JmDNS is a class that provides a way to register and discover network services using the DNS Service Discovery (DNS-SD) protocol.

After creating the JmDNS instance, three ServiceInfo objects are created, one for each of the three services (AppointmentService, CheckinService, and TriageService). Each ServiceInfo object is created by calling the create() method and passing in the service type

(\_grpc.\_tcp.local.), service name, port number, and path of the service as arguments. The ServiceInfo class is used to encapsulate information about a network service, such as its name, port, and protocol.

Finally, the three services are registered by calling the registerService() method on the JmDNS instance and passing in the corresponding ServiceInfo object as an argument. This tells the JmDNS instance to advertise the service on the network using the DNS-SD protocol.

After registering the services, the startServer() method is called on each of the three services (AppointmentService, CheckinService, and TriageService) to start their respective servers and listen for incoming connections.

# **Remote Error Handling and advanced features**

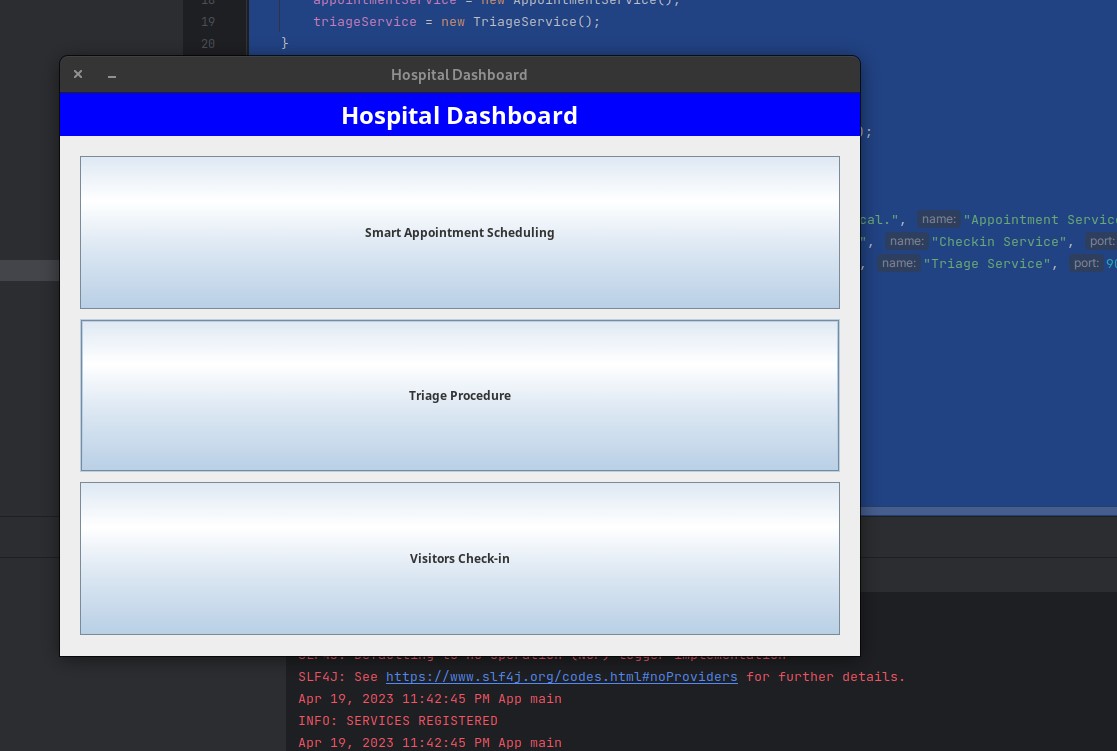
In this project, remote error handling was considered an essential aspect of the distributed system to prevent errors from propagating throughout the system. To achieve this, gRPC's built-in error handling mechanisms were used to provide more detailed error messages and better communication between services.

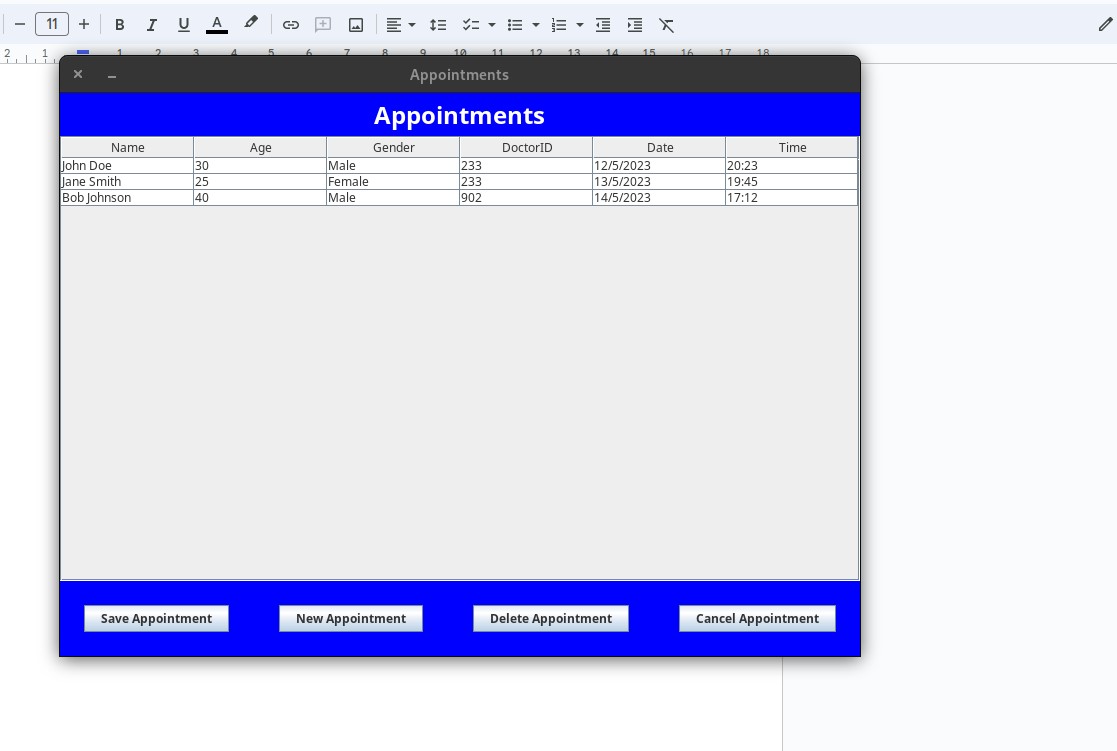
To handle authentication and authorization, gRPC's built-in support for SSL/TLS was utilized. This helped to ensure that only authorized users could access certain services, enhancing the overall security of the system.

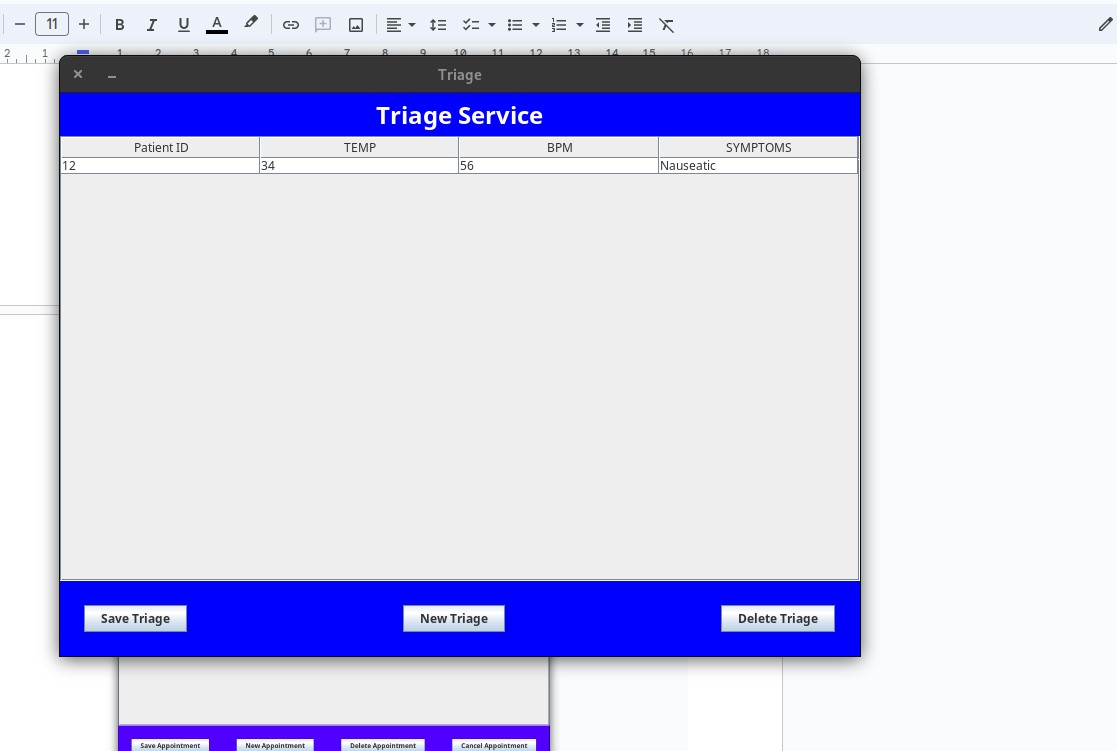
Custom error codes and messages were defined for each service to provide informative and helpful error messages to clients in the event of an error. This allowed clients to quickly identify the cause of the error and take appropriate action.

# **Client - Graphical User Interface (GUI**

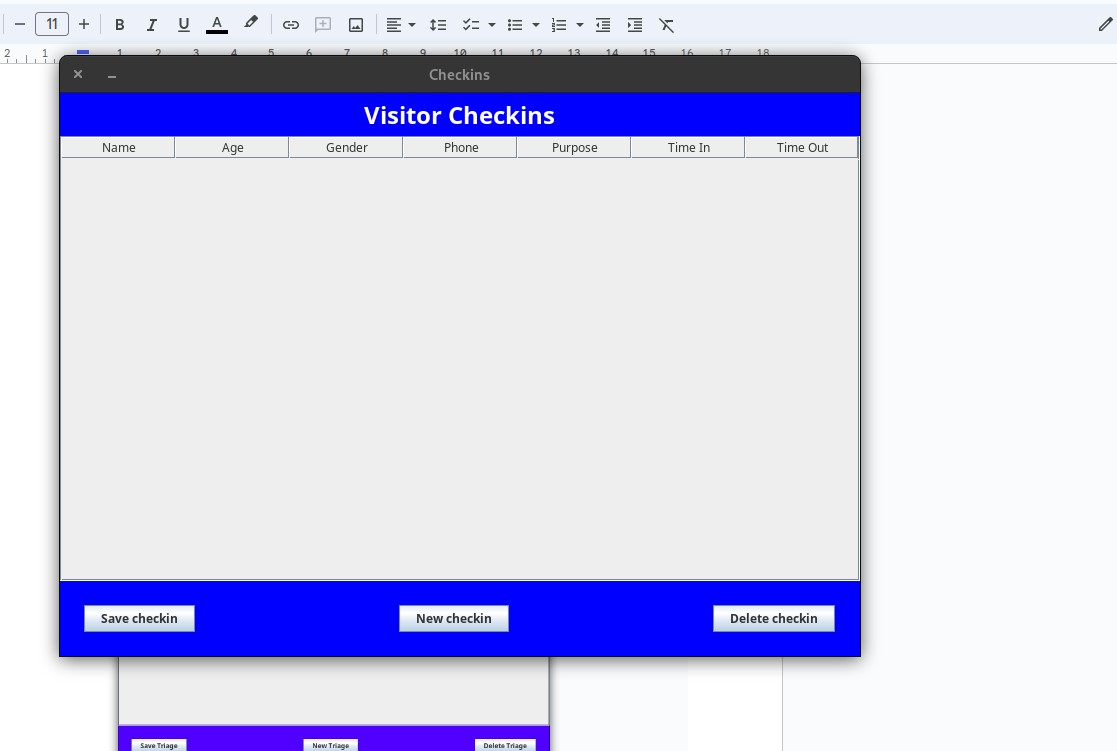
In order to interact with the services, I created a graphical user interface for reading and sending data to the services. The following is a snapshot







SMART HEALTH



**GitHub repository**

<https://github.com/heliobs>

**Video Presentation**

https://youtu.be/ubwido8zzq8

SMART HEALTH

## REFERENCES

"Distributed Systems: Concepts and Design" by George Coulouris, Jean Dollimore, and Tim Kindberg

"Microservices Patterns: With examples in Java" by Chris Richardson

"Building Microservices: Designing Fine-Grained Systems" by Sam Newman

"gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes" by Kasun Indrasiri and Danesh Kuruppu

"gRPC Official Documentation" available at <https://grpc.io/docs/>

"The Design of the UNIX Operating System" by Maurice J. Bach (for understanding distributed system design principles)

"Scalable Web Architecture and Distributed Systems" (free online course) available at https://www.educative.io/courses/scalable-web-architecture-and-distributed-systems