Distributed Systems

Report

Name: Muhammad Abu Bakar Sani

Student No: x17112044

Course: BSHCSD4

***Smart Buildings***

Contents

[Introduction 3](#_Toc66985610)

[Services 3](#_Toc66985611)

[Smart Lights 3](#_Toc66985612)

[Smart Heating 4](#_Toc66985613)

[Smart Security 4](#_Toc66985614)

[Naming Services 5](#_Toc66985615)

[jmDNS 5](#_Toc66985616)

[zeroconf 5](#_Toc66985617)

[gRPC 5](#_Toc66985618)

[Remote Error Handling 5](#_Toc66985619)

[Graphical User Interface (GUI) 6](#_Toc66985620)

[GitHub 6](#_Toc66985621)

# Introduction

For Distributed systems project, my given topic is Smart Buildings because my student number last at 4. I have made my revolving around it, using services that I felt would be useful. These are the 3 services that I have implemented:

* Smart Heating
* Smart Lighting
* Smart Security

I have completed this project using gRPC for 2 java services and 1 Python service. I also used mDNS to discover and register these services. Separate GUI’s are also developed to control them and making it more user friendly.

# Services

## Smart Lights

This service was developed in Java using Eclipse IDE and had implemented 3 RPC’s. The RPC’s for this service was specified in light.proto file. These are the RPC’s for this service.

The powerSwitch is the first RPC. It’s an implementation of unary RPC. The powerSwitch RPC takes in “PowerRequest”, which has Boolean variable ‘power’ and returns “PowerResponse” which has also a Boolean variable named ‘power’. This feature was developed to give the user the ability to turn ON and OFF their smart lights. When the server receives the request, it prints out the request made and return the same variable in the response and when the client gets the response it will print out either “Server response: Light turned ON” or “Server response: Light turned OFF”, depending on whether or not the response is true or false.

The changeColour is the second RPC. Its an implementation of Client-side streaming RPC. This RPC takes in stream of “ColourRequest”, which has String variable ‘colour’ and returns “ColourResponse” which has also String variable ‘colour’. This feature was developed to give user the ability to change different colours as stream input. When the server receives the request, it prints out the requested colour. Once the client stops sending the requests, server will set the colour to the last one in the queue and display this message “Colour has been set to +colour”. This RPC was created to allow the user to dynamically change the colour of the smart lights.

The calculateBill is the third RPC. Its an implementation of Bidirectional streaming RPC. This RPC takes in stream of “BillRequest”. It has following variables:- ‘reading:int32’, ‘price:double’ and ‘discount:double’ and returns stream of “BillResponse” which has ‘reading:int32’ and ‘bill:double’ variables. This feature enables the users to calculate their bills. When clients stop sending requests, the server will response with the steam of calculated bill and its reading.

## Smart Heating

This service was also developed in Java using Eclipse IDE and had implemented 3 RPC’s. The RPC’s for this service was specified in heat.proto file. These are the RPC’s for this service.

The heatSwitch is the first RPC. It’s an implementation of unary RPC. The heatSwitch RPC takes in “HeatRequest”, which has Boolean variable ‘heat and returns “HeatResponse” which has also a Boolean variable named ‘heat. This feature was developed to give the user the ability to turn ON and OFF their smart heating. When the server receives the request, it prints out the request made and return the same variable in the response and when the client gets the response it will print out either “Server response: Heating turned ON” or “Server response: Heating turned OFF”, depending on whether or not the response is true or false.

The changeTemperature is the second RPC. It’s an implementation of Server-side streaming RPC. This RPC takes in request “TemperatureRequest”, which has following variable:- ‘number:int32’, ‘start:int32 and ‘increment:int32’ and returns stream of “TemperatureResponse” which has int32 variable ‘temperature’. This feature was developed to give user the ability to change temperature automatically. When the server receives request, it prints out the received input and starts incrementing the temperature according to user input. Client displays this message “Next temperature: ‘temp’ °C".

The suggestTemperature is the third RPC. It’s an implementation of Bidirectional streaming RPC. This RPC takes in stream of “SuggestRequest”. It has following variables:- ‘temp:int32’ and ‘date:string’ and returns stream of “SuggestResponse” which has ‘temp:int32’, ‘guess:string’ and ‘date:string’ variables. This feature enables the users to check their guessed temperature is high, low, or moderate. When client stop sending requests, the server will respond with the steam of results.

## Smart Security

This service was developed in Python using PyCharm IDE and had implemented 3 RPC’s. The RPC’s for this service was specified in security.proto file. These are the RPC’s for this service.

The securitySwitch is the first RPC. It’s an implementation of unary RPC. The securitySwitch RPC takes in “SecurityRequest”, which has Boolean variable ‘security’ and returns “SecurityResponse” which has also a Boolean variable named ‘security’. This feature was developed to give the user the ability to turn ON and OFF their smart heating. When the server receives the request, it prints out the request made and return the same variable in the response and when the client gets the response it will print out either “Server response: Security turned ON” or “Server response: Security turned OFF”, depending on whether or not the response is true or false.

The Liststaff is the second RPC. It’s an implementation of Server-side streaming RPC. This RPC takes in request “ListRequest”, which has variable:- ‘ask:string and returns stream of “ListResponse” which has string variable ‘ans’. This feature was developed to give user the ability to check people present in building. When the server receives request, it prints out the received request and start streaming the names. Client displays this message “Name: ‘name’".

The grantAccess is the third RPC. It’s an implementation of Client-side streaming RPC. This RPC takes in stream of “AccessRequest”, which has String variables ‘name’ & ‘id’ and returns “AccessResponse” which has also String variable ‘reply’. This feature was developed to give user the ability to add people details who need to grant access to the building. Once the client stops sending the requests, server will grant access and display this message “Today’s access granted to above names”.

# Naming Services

## jmDNS

jmDNS was implemented and used to discover the services. I set up a SmartBuildingServiceDiscovery.java and SmartBuildingServiceRegistration.java. I set up each service with a unique ID and with the discover and register files the jmDNS was able to pick up and register my services

## zeroconf

zeroconf was used to implement mDNS in Python.

## gRPC

gRPC was implemented and used to create all services. Each service has a unique proto file, and all four types of RPC were implemented.

# Remote Error Handling

Remote Error Handling was implemented using try catch methods on call to ensure errors are caught, these included RuntimeException handling.

# Graphical User Interface (GUI)

I have implemented 3 different GUI’s for 3 different services. The GUI’s were developed in java and the file names are HeatGUI.java, LightGUI.java and SecurityGUI.java. So, user have 2 options to use those services: 1st with command line and 2nd with GUI.

# GitHub

A GitHub repository was maintained from the beginning of the project with regular commits and version control. Inside is my Java and python codes along with this document. As the repository is private so no one can access it except me.