PollutionWatch: An Advanced System for Real-Time Monitoring and Reporting of Environmental Pollution in Urban Settings

Distributed Systems
March 2021

Luis Navarro Torres
ID: 22103902
Subject: Smart water/air pollution tracking

Contents

1 Service Definitions

The aim of this project is to develop a RPC communication system for the advanced system for real-time monitoring and reporting of environmental pollution in urban settings. The proposed system will simulate the use of sensors in order to take data in the different rivers of a stablished city as well as sensors in different points of the city to gather sample of the air quality to be analysed and determine the levels of pollution in real-time.

The system will also incorporate a user-friendly GUI that would enable citizens, policymakers, and environmental agencies to access pollution data and reports easily. By providing real-time information on pollution levels.

The proposed system will show examples of the four kinds of service methods of RPC divided in three different Services.

2 Service 1: River water control + Proto

This service deals with the control of the pollution of a river.

2.1 Methods

2.1.1 RPC Method 1 – Quick water oxygen testing

This **Unary RPC** method works when the client sends a single oxygen level value from a single point, the server compares it to the healthy oxygen values that the water sample should have and return a single answer depending on this comparison, the answer will be “the water is acceptable” or “the water is polluted”.

rpc oxigenLevel(testRequest) **returns** (testResponse);

2.1.2 RPC Method 2 – Deep water testing

This **client streaming RPC** method can give the client a better, more reliable answer to the water pollution status of a single point. The server will wait until it gets the ph levels, the temperature, and the oxygen level, from the client before following an algorithm that will determine the answer that it will send back to the client, this answer could be “All levels are correct, water is healthy” or “Some levels are correct, water acceptable” or “All levels are out of bounds, water polluted”.

rpc manyValues(stream deepTestRequest) **returns** (deepTestResponse);

2.1.3 RPC Method 3 – River status check

This **client streaming RPC** method works by waiting for the client to send the name of a river, the oxygen, temperature and ph readings of it and the server will save this data and answer a confirmation.

rpc manyReadings(stream riverDataRequest) **returns** (riverDataResponse);

2.1.4 Proto file

```
syntax = "proto3";

// Options for the GRPC

option java_multiple_files = true;

option java_package = "grpc.service1";
```

```
option java_outer_classname = "RiverWaterControlImpl";

package PollutionWatch;

// Service Definition

service RiverWaterControl{

rpc oxigenLevel(testRequest) returns (testResponse){}

rpc manyValues(stream deepTestRequest) returns (deepTestResponse){}

rpc manyReadings(stream riverDataRequest) returns (riverDataResponse){}

}

// Define the messages for oxigenLevel method

//(Unary)The client sends one float oxygen level value and the server
answers with a string.

message testRequest {

float testReq = 1;

}

message testResponse {

string testRes = 1;

}

// Define the messages for manyValues method

//(Client Streaming) The client sends 3 values taken from a river (oxygen,
temperature and ph)

// The server will check this values and answer with the quality of the
water.

message deepTestRequest {

float deepTestReqOxy = 1;

float deepTestReqTemp = 2;

float deepTestReqPh = 3;

}

message deepTestResponse {

string deepTestRes = 1;
```

```
}

// Define the messages for manyReadings method

//(Client Streaming) The client sends the name of a river and 3
readings(oxygen, temperature and ph)

// The server will save this data and answer a confirmation.

message riverDataRequest {

string riverDataReq = 1;

float riverDataReqOxy = 2;

float riverDataReqTemp = 3;

float riverDataReqPh = 4;

}

message riverDataResponse {

string riverDataRes = 1;

}
```

3 Service 2: Air quality check

This service is in charge of monitoring the air pollution of a neighbourhood.

3.1 Methods

3.1.1 RPC Method 1 – Neighbourhood air quality

In this **unary RPC**, the client will send the name of a neighbourhood, the server will respond if it has the info about that area either good air quality or poor air quality or if it does not have the information, information not available.

rpc neighbourhoodName(statusRequest) **returns** (statusResponse);

3.1.2 RPC Method 2 - Burette method oxygen%

This is a **bidirectional streaming** method that calculates de percentage of oxygen in the air of a determinate place using the burette method. The client will send the 2 variables and the server will perform the calculation and send back the answer and if it is healthy for a human.

rpc BidiOxygen(stream airOxygenRequest) **returns** (stream airOxygenResponse);

3.1.3 RPC Method 3 – Air pollution check

This **client streaming RPC** method works when the client sends a carbon monoxide value and a ground level ozone reading to the server which compares it to the healthy values that the air sample should have and return a single answer depending on this comparison, the answer will be “the air is acceptable” or “the air is polluted”.

rpc manyAirReadings(stream airDataRequest) returns (airDataResponse);

3.1.4 Proto file

```
syntax = "proto3";

// Options for the GRPC

option java_multiple_files = true;

option java_package = "grpc.service2";

option java_outer_classname = "AirQualityCheckImpl";

package PollutionWatch;

// Service Definition

service AirQualityCheck{

rpc neighbourhoodName(statusRequest) returns (statusResponse){}

rpc BidiOxygen(stream airOxygenRequest) returns (stream
airOxygenResponse){}

rpc manyAirReadings(stream airDataRequest) returns (airDataResponse){}

}

// Define the messages for neighbourhoodName method

//(Unary)The client sends one neighbourhood's name, and the server

// answers with the quality of the air in that place.

message statusRequest {

string statusReq = 1;

}

message statusResponse {

string statusRes = 1;

}

// Define the messages for BidiOxygen method

//(Bidirectional) The client sends 2 values (water height before and after
in a 50cm3 burette)
```

```
// The server will calculate the percentage oxygen, send it and also send
if it is healthy or not.

message airOxygenRequest {

float airOxygenReqBef = 1;

float airOxygenReqAft = 2;

}

message airOxygenResponse {

float airOxygenResPer = 1;

string airOxygenResHea = 2;

}

// Define the messages for manyAirReadings method

//(Client Streaming) The client sends a carbon monoxide value and a ground
level ozone reading

// the server compares them to the healthy values that the air sample
should have and return a single answer.

message airDataRequest {

float airDataReqMon = 1;

float airDataReqOz = 2;

}

message airDataResponse {

string airDataRes = 1;

}
```

4 Service 3: Area status control

This service deals with the recollection of data from the server

4.1 RPC Methods

4.1.1 RPC Method 1 – Historic monthly record

This **server streaming** method consi sts in the client sending the name of the river they want the information to be sent, the server will then start sending the results of the pollution analysis of each day of the last 10 days, these answers will have the format: **Day** (1….10) **pollution levels** (acceptable or unacceptable).

rpc riverHistoric(riverDataRequest) **returns** (stream riverDataResult);

4.1.2 RPC Method 2 – River pollution status

This is a **bidirectional streaming** RPC where the client will send 2 o mas river names and the server will send back as many data as it has about that river water.

rpc BidiRiver(stream waterStatusRequest) **returns** (stream waterStatusResponse);

4.1.3 RPC Method 3 – Neighbourhood air pollution status

This is a **bidirectional streaming** RPC where the client will send 2 o mas neighbourhood names and the server will send back as many data as it has about that neighbourhood's air pollution status.

rpc BidiNeighbourhood(stream neighborhoodRequest) **returns** (stream neighborhoodStatusResponse);

4.1.4 Proto file

```
syntax = "proto3";

// Options for the GRPC

option java_multiple_files = true;

option java_package = "grpc.service3";

option java_outer_classname = "AreaStatusControlImpl";

package PollutionWatch;

// Service Definition

service AreaStatusControl{

rpc riverHistoric(riverDataRequest) returns (stream riverDataResponse){}

rpc BidiRiver(stream waterStatusRequest) returns (stream
waterStatusResponse){}

rpc BidiNeighbourhood(stream neighborhoodStatusRequest) returns (stream
neighborhoodStatusResponse){}

}

// Define the messages for riverHistoric method

//(Server streaming)The client sends one river's name, and the server

// answers with the information of the last 10 days.

message riverDataRequest {

string riverDataReq = 1;

}

message riverDataResponse {
```

```
string riverDataRes = 1;

}

// Define the messages for BidiRiver method

//(Bidirectional) The client sends 2 river names

// The server will answer with data from this two rivers.

message waterStatusRequest {

string waterStatusReq = 1;

}

message waterStatusResponse {

string waterStatusRes = 1;

}

// Define the messages for BidiNeighbourhood method

//(Bidirectional) The client sends 2 neighborhood names

// The server will answer with data from this two neighborhood.

message neighborhoodStatusRequest {

string neighborhoodStatusReq = 1;

}

message neighborhoodStatusResponse {

string neighborhoodStatusRes = 1;

}
```

5 Services Implementations

5.1 Properties

To help maintain the order in the files a properties.java file was created for each of the 3 services:

5.1.1 River Water Control Properties file

```
service_type=_GRPCServ1._tcp.local.

service_name=River water control

service_description= This service deals with the control of the pollution
of rivers.
```

```
service_port=50051
```

5.1.2 Air Quality Check Properties file

```
service_type=_GRPCServ2._tcp.local.

service_name=Air Quality Check

service_description= This service is in charge of monitoring the air
pollution of a neighbourhood.

service_port=50052
```

5.1.3 Area StatusControl Properties file

```
service_type=_GRPCServ3._tcp.local.

service_name=Area Status Control

service_description= This service deals with the recollection of data from
the server for River water and neighbohuds air pollution.

service_port=50053
```

5.2 GRPC for River water control

5.2.1 Server

```
package grpc.service1;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.InetAddress;

import java.util.ArrayList;

import java.util.Properties;

import javax.jmdns.JmDNS;

import javax.jmdns.ServiceInfo;

import grpc.service1.RiverWaterControlGrpc.RiverWaterControlImplBase;
```

```
import io.grpc.Server;

import io.grpc.ServerBuilder;

import io.grpc.stub.StreamObserver;

public class RiverWaterControlServer extends RiverWaterControlImplBase{

public static void main(String[] args) {

RiverWaterControlServer serv1Riverserver = new RiverWaterControlServer();

//int port = 50051;

// Registration

//

Properties prop = serv1Riverserver.getProperties();

serv1Riverserver.registerService(prop);

int port = Integer.valueOf(prop.getProperty("service_port"));// #.50051;

Server server;

//

try {

server =
ServerBuilder.forPort(port).addService(serv1Riverserver).build().start();

System.out.println("Server 1 working");

server.awaitTermination();

} catch (IOException | InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}//

private Properties getProperties() {

Properties prop = null;

try (InputStream input = new
FileInputStream("src/main/resources/Serv1_RiverWaterControl.properties")) {
```

```
prop = new Properties();

// load a properties file

prop.load(input);

// get the property value and print it out

System.out.println("Service 1 - Pollution WatchArea: ");

System.out.println("\t service_type: " + prop.getProperty("service_type"));

System.out.println("\t service_name: " + prop.getProperty("service_name"));

System.out.println("\t service_description: " +
prop.getProperty("service_description"));

System.out.println("\t service_port: " + prop.getProperty("service_port"));

} catch (IOException ex) {

ex.printStackTrace();

}

return prop;

}

private void registerService(Properties prop) {

try {

// Create a JmDNS instance

JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

String service_type = prop.getProperty("service_type");//
"_GRPCServ1._tcp.local.";

String service_name = prop.getProperty("service_name");// "River water
control";

int service_port = Integer.valueOf(prop.getProperty("service_port"));//
#.50051;

String service_description_properties =
prop.getProperty("service_description");//description

// Registration service information

ServiceInfo serviceInfo = ServiceInfo.create(service_type, service_name,
service_port,
```

```
service_description_properties);

jmdns.registerService(serviceInfo);

System.out.printf("registrering service with type %s and name %s \n",
service_type, service_name);

// Wait a bit for the service to register

Thread.sleep(1000);

} catch (IOException e) {

System.out.println(e.getMessage());

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}//

@Override

public void oxigenLevel(testRequest request, StreamObserver<testResponse>
responseObserver) {

// TODO Auto-generated method stub

//super.oxigenLevel(request, responseObserver);

System.out.println("----- Receiving oxigen level (mg/L) from client -----
");

String value1 = "";

if(request.getTestReq()>=6.5 && request.getTestReq()<=8) {

value1 = "HEALTHY";

}else {

value1 = "UNHEALTHY";

}

testResponse response = testResponse.newBuilder().setTestRes("--- RPC 1 ---
\nThe status of the water is: "+ value1).build();

responseObserver.onNext( response );
```

```
responseObserver.onCompleted();

}

@Override

public StreamObserver<deepTestRequest>
manyValues(StreamObserver<deepTestResponse> responseObserver) {

// TODO Auto-generated method stub

//return super.manyValues(responseObserver);

System.out.println("----- Receiving river information from client -----");

final int[] deepTestCount = {0};

final String[] value2 = {""};

return new StreamObserver<deepTestRequest>() {

@Override

public void onNext(deepTestRequest value) {

// TODO Auto-generated method stub

if (value.getDeepTestReqOxy()>=6.5 && value.getDeepTestReqOxy()<=8) {

deepTestCount[0]++;

}

if (value.getDeepTestReqTemp()>=20 && value.getDeepTestReqTemp()<=31.67) {

deepTestCount[0]++;

}

if (value.getDeepTestReqPh()>=6 && value.getDeepTestReqPh()<=8) {

deepTestCount[0]++;

}

if(deepTestCount[0]<=1) {

value2[0] = "All levels are out of bounds, water polluted";

}

if(deepTestCount[0]==2){

value2[0] = "Some levels are correct, water acceptable";
```

```
}

if(deepTestCount[0]==3){

value2[0] = "All levels are correct, water is healthy";

}

}

@Override

public void onError(Throwable t) {

// TODO Auto-generated method stub

//Log the error

System.err.println("Error in manyValues: " + t.getMessage());

// Send an error response to the client

responseObserver.onError(t);

}

@Override

public void onCompleted() {

// TODO Auto-generated method stub

String message = "... " + value2[0];

deepTestResponse response =
deepTestResponse.newBuilder().setDeepTestRes(message).build();

responseObserver.onNext(response);

responseObserver.onCompleted();

}

};

}

@Override

public StreamObserver<riverDataRequest>
manyReadings(StreamObserver<riverDataResponse> responseObserver) {

// TODO Auto-generated method stub
```

```
//return super.manyReadings(responseObserver);

System.out.println("----- Receiving river name and it's information to be
stored -----");

final ArrayList<String> riverNames = new ArrayList<String>();

final ArrayList<Float> riverOxy = new ArrayList<Float>();

final ArrayList<Float> riverTemp = new ArrayList<Float>();

final ArrayList<Float> riverPh = new ArrayList<Float>();

return new StreamObserver<riverDataRequest>() {

@Override

public void onNext(riverDataRequest value) {

// TODO Auto-generated method stub

riverNames.add(value.getRiverDataReq());

riverOxy.add(value.getRiverDataReqOxy());

riverTemp.add(value.getRiverDataReqTemp());

riverPh.add(value.getRiverDataReqPh());

}

@Override

public void onError(Throwable t) {

// TODO Auto-generated method stub

//Log the error

System.err.println("Error in manyReadings: " + t.getMessage());

// Send an error response to the client

responseObserver.onError(t);

}

@Override

public void onCompleted() {

// TODO Auto-generated method stub
```

```
riverDataResponse response =
riverDataResponse.newBuilder().setRiverDataRes("River data saved,
thanks").build();

responseObserver.onNext(response);

responseObserver.onCompleted();

}

};

}

}
```

5.2.2 Client

```
package grpc.service1;

import java.net.InetAddress;

import java.util.concurrent.TimeUnit;

import javax.jmdns.JmDNS;

import javax.jmdns.ServiceEvent;

import javax.jmdns.ServiceInfo;

import javax.jmdns.ServiceListener;

import grpc.service1.RiverWaterControlGrpc.RiverWaterControlBlockingStub;

import grpc.service1.RiverWaterControlGrpc.RiverWaterControlStub;

import io.grpc.ManagedChannel;

import io.grpc.ManagedChannelBuilder;

import io.grpc.StatusRuntimeException;

import io.grpc.stub.StreamObserver;

public class RiverWaterControlCient {

private static RiverWaterControlBlockingStub blockingstub;

private static RiverWaterControlStub asyncStub;

static String host= "_GRPCServ1._tcp.local.";//"localhost";
```

```
static int port; //= 50051;

static String resolvedIP;

public static void main(String[] args) {

testClientJMDNS();

ManagedChannel channel = ManagedChannelBuilder.forAddress(resolvedIP,
port).usePlaintext().build();

blockingstub = RiverWaterControlGrpc.newBlockingStub(channel);

asyncStub = RiverWaterControlGrpc.newStub(channel);

//Method calls and channel shutdown

oxigenLevel();

manyValues();

manyReadings();

try {

channel.shutdown().awaitTermination(5,TimeUnit.SECONDS);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

private static void testClientJMDNS() {

// TODO Auto-generated method stub

try {

// Create a JmDNS instance

JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

// Add a service listener

jmdns.addServiceListener(host, new SampleListener());

// Wait a bit

Thread.sleep(20000);
```

```
} catch (Exception e) {
System.out.println(e.getMessage());
}
}
private static class SampleListener implements ServiceListener {
public void serviceAdded(ServiceEvent event) {
System.out.println("Service added: " + event.getInfo());
}
public void serviceRemoved(ServiceEvent event) {
System.out.println("Service removed: " + event.getInfo());
}
@SuppressWarnings("deprecation")
public void serviceResolved(ServiceEvent event) {
System.out.println("Service resolved: " + event.getInfo());
ServiceInfo info = event.getInfo();
port = info.getPort();
resolvedIP = info.getHostAddress();
System.out.println("IP Resolved - " + resolvedIP + ":" + port);
}
}
public static void oxigenLevel() {
// TODO Auto-generated method stub
try {
testRequest request = testRequest.newBuilder().setTestReq(7).build();
testResponse response = blockingstub.oxigenLevel(request);
System.out.println(response.getTestRes());
}catch (StatusRuntimeException e) {
```

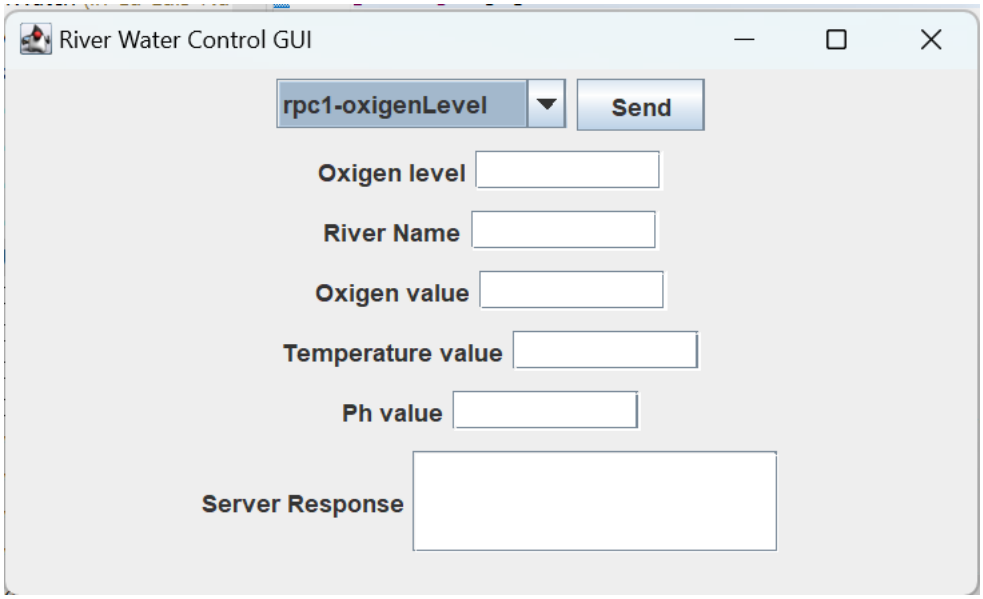
```
e.getStatus();

}

}

public static void manyValues() {

// TODO Auto-generated method stub

StreamObserver<deepTestResponse> deepReply = new
StreamObserver<deepTestResponse>() {

@Override

public void onNext(deepTestResponse value) {

// TODO Auto-generated method stub

System.out.println("--- RPC 2 --- \nThe status of the deep water test is: "
+ value.getDeepTestRes());

}

@Override

public void onError(Throwable t) {

// TODO Auto-generated method stub

t.printStackTrace();

}

@Override

public void onCompleted() {

// TODO Auto-generated method stub

System.out.println("Deep test finished succesfully");

}

};

StreamObserver<deepTestRequest> deepResponse =
asyncStub.manyValues(deepReply);

try {

deepTestRequest request =
deepTestRequest.newBuilder().setDeepTestReqOxy(7).setDeepTestReqTemp(25).se
tDeepTestReqPh(70).build();
```

```
deepResponse.onNext(request);

Thread.sleep(500);

deepResponse.onCompleted();

Thread.sleep(3000);

} catch (RuntimeException | InterruptedException e) {

e.printStackTrace();

}

}

public static void manyReadings() {

// TODO Auto-generated method stub

StreamObserver<riverDataResponse> riverReply = new
StreamObserver<riverDataResponse>() {

@Override

public void onNext(riverDataResponse value) {

// TODO Auto-generated method stub

System.out.println("--- RPC 3 --- \nStoring data... " +
value.getRiverDataRes());

}

@Override

public void onError(Throwable t) {

// TODO Auto-generated method stub

t.printStackTrace();

}

@Override

public void onCompleted() {

// TODO Auto-generated method stub

System.out.println("River info storing ended");

}
```

```
};

StreamObserver<riverDataRequest> riverResponse =
asyncStub.manyReadings(riverReply);

try {

riverDataRequest request =
riverDataRequest.newBuilder().setRiverDataReq("Lifey").setRiverDataReqOxy(1
0).setRiverDataReqTemp(30).setRiverDataReqPh(9).build();

riverResponse.onNext(request);

Thread.sleep(500);

riverResponse.onCompleted();

Thread.sleep(10000);

} catch (RuntimeException | InterruptedException e) {

e.printStackTrace();

}

}

}
```

5.2.3 GUI

In this GUI first we have a rpc method selector and the button to send the data to the server, depending on the service that the client wants to run he/she needs to fill those spaces, it looks like this:



Code

```
package grpc.service1;

import java.awt.EventQueue;

import java.awt.FlowLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.BoxLayout;

import javax.swing.DefaultComboBoxModel;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.JTextField;

public class GUIserv1 {

public JFrame frame;

private JTextField textOxigenLevel;

private JTextField textOxigenValue;

private JTextField textTempValue;

private JTextField textPhValue;

private JTextField textRiverName;

private JTextArea textResponseRiverControl ;

/**

* Launch the application.

*/
```

```
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
public void run() {
try {
GUIserv1 window = new GUIserv1();
window.frame.setVisible(true);
} catch (Exception e) {
e.printStackTrace();
}
}
});
}
public GUIserv1() {
initialize();
}
/**
* Initialize the contents of the frame.
*/
private void initialize() {
frame = new JFrame();
frame.setTitle("River Water Control GUI");
frame.setBounds(100, 100, 500, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
BoxLayout bl = new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS);
frame.getContentPane().setLayout(bl);
//panel 1
JPanel panel_service_1 = new JPanel();
```

```
frame.getContentPane().add(panel_service_1);
panel_service_1.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
//panel 2
JPanel panel_service_2 = new JPanel();
frame.getContentPane().add(panel_service_2);
panel_service_2.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel lblNewLabel_2 = new JLabel("Oxigen level");
panel_service_2.add(lblNewLabel_2);
textOxigenLevel = new JTextField();
panel_service_2.add(textOxigenLevel);
textOxigenLevel.setColumns(10);
//panel 3
JPanel panel_service_3 = new JPanel();
frame.getContentPane().add(panel_service_3);
panel_service_3.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel lblNewLabel_3 = new JLabel("River Name");
panel_service_3.add(lblNewLabel_3);
textRiverName = new JTextField();
panel_service_3.add(textRiverName);
textRiverName.setColumns(10);
//panel 4
JPanel panel_service_4 = new JPanel();
frame.getContentPane().add(panel_service_4);
panel_service_4.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel lblNewLabel_4 = new JLabel("Oxigen value");
panel_service_4.add(lblNewLabel_4);
textOxigenValue = new JTextField();
```

```
panel_service_4.add(textOxigenValue);

textOxigenValue.setColumns(10);

//panel 5

JPanel panel_service_5 = new JPanel();

frame.getContentPane().add(panel_service_5);

panel_service_5.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));

JLabel lblNewLabel_5 = new JLabel("Temperature value");

panel_service_5.add(lblNewLabel_5);

textTempValue = new JTextField();

panel_service_5.add(textTempValue);

textTempValue.setColumns(10);

//panel 6

JPanel panel_service_6 = new JPanel();

frame.getContentPane().add(panel_service_6);

panel_service_6.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));

JLabel lblNewLabel_6 = new JLabel("Ph value");

panel_service_6.add(lblNewLabel_6);

textPhValue = new JTextField();

panel_service_6.add(textPhValue);

textPhValue.setColumns(10);

//

final JComboBox comboOperation = new JComboBox();

comboOperation.setModel(new DefaultComboBoxModel(new String[] {"rpc1-
oxigenLevel", "rpc2-manyValues", "rpc3-manyReadings"}));

panel_service_1.add(comboOperation);

JButton btnCalculate = new JButton("Send");

btnCalculate.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {
```

```
float num1 = Float.parseFloat(textOxigenLevel.getText());

float num2 = Float.parseFloat(textOxigenValue.getText());

float num3 = Float.parseFloat(textTempValue.getText());

float num4 = Float.parseFloat(textPhValue.getText());

String name = textRiverName.getText();

int index = comboOperation.getSelectedIndex();

if (index == 0) {

if(num1>=6.5 && num1<=8) {

name="Result of Test: The river is HEALTY";

}else {

name="Result of Test: The river is UNHEALTY";

}

}

if (index == 1) {

textResponseRiverControl.append("Result of Subtraction" +
String.valueOf(num1 - num2));

}

if (index == 2) {

textResponseRiverControl.append("Result of Multiplication" +
String.valueOf(num1 * num2));

}

}

});

panel_service_1.add(btnCalculate);

textResponseRiverControl = new JTextArea(3, 20);

textResponseRiverControl .setLineWrap(true);

textResponseRiverControl.setWrapStyleWord(true);

JScrollPane scrollPane = new JScrollPane(textResponseRiverControl);
```

```
//textResponse.setSize(new Dimension(15, 30));

JPanel panel_service_7 = new JPanel();

JLabel lblNewLabel_7 = new JLabel("Server Response");

panel_service_7.add(lblNewLabel_7);

frame.getContentPane().add(panel_service_7);

panel_service_7.add(scrollPane);

JPanel panel_service_8 = new JPanel();

frame.getContentPane().add(panel_service_8);

JPanel panel_service_9 = new JPanel();

frame.getContentPane().add(panel_service_9);

}

}
```

5.3 GRPC for Air Quality Check

5.3.1 Server

```
package grpc.service2;

import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.InetAddress;

import java.util.Properties;

import java.util.logging.Logger;

import javax.jmdns.JmDNS;

import javax.jmdns.ServiceInfo;

import grpc.examples.bidirectionstreamstrings.StringServer;

import grpc.service2.AirQualityCheckGrpc.AirQualityCheckImplBase;

import io.grpc.Server;
```

```
import io.grpc.ServerBuilder;

import io.grpc.stub.StreamObserver;

public class AirQualityCheckServer extends AirQualityCheckImplBase {

private static final Logger logger =
Logger.getLogger(StringServer.class.getName());

public static void main(String[] args) {

AirQualityCheckServer serv2Airserver = new AirQualityCheckServer();

//int port = 50052;

// Registration

//

Properties prop = serv2Airserver.getProperties();

serv2Airserver.registerService(prop);

int port = Integer.valueOf(prop.getProperty("service_port"));// #.50052;

Server server;

//

try {

server =
ServerBuilder.forPort(port).addService(serv2Airserver).build().start();

System.out.println("Server 2 working");

server.awaitTermination();

} catch (IOException | InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}//

private Properties getProperties() {

Properties prop = null;

try (InputStream input = new
FileInputStream("src/main/resources/Serv2_AirQualityCheck.properties")) {
```

```
prop = new Properties();

// load a properties file

prop.load(input);

// get the property value and print it out

System.out.println("Service 2 - Pollution WatchArea: ");

System.out.println("\t service_type: " + prop.getProperty("service_type"));

System.out.println("\t service_name: " + prop.getProperty("service_name"));

System.out.println("\t service_description: " +
prop.getProperty("service_description"));

System.out.println("\t service_port: " + prop.getProperty("service_port"));

} catch (IOException ex) {

ex.printStackTrace();

}

return prop;

}

private void registerService(Properties prop) {

try {

// Create a JmDNS instance

JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

String service_type = prop.getProperty("service_type");//
"_GRPCServ2._tcp.local.";

String service_name = prop.getProperty("service_name");// "Air Quality
Control";

int service_port = Integer.valueOf(prop.getProperty("service_port"));//
#.50052;

String service_description_properties =
prop.getProperty("service_description");//description

// Registration service information

ServiceInfo serviceInfo = ServiceInfo.create(service_type, service_name,
service_port,
```

```
service_description_properties);

jmdns.registerService(serviceInfo);

System.out.printf("registrering service with type %s and name %s \n",
service_type, service_name);

// Wait a bit for the service to register

Thread.sleep(1000);

} catch (IOException e) {

System.out.println(e.getMessage());

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}//

@Override

public void neighbourhoodName(statusRequest request,
StreamObserver<statusResponse> responseObserver) {

// TODO Auto-generated method stub

//super.neighbourhoodName(request, responseObserver);

System.out.println("----- Receiving name of the neighbourhood from client -
----");

String answer = "The neighbourhood is not yet on file";

String neigh1 = new String("Churchtown");

String neigh2 = new String("Temple bar");

String neigh3 = new String("Smithfield");

String neigh4 = new String("Ranelagh");

String neigh5 = new String("Portobello");

if(request.getStatusReq().equals(neigh1)) {

answer = "Air quality in Churchtown is GOOD (every individual can stay)";

}
```

```
if(request.getStatusReq().equals(neigh2)) {

answer = "Air quality in Temple bar is UNHEALTHY (avoid prolonged
exposure)";

}

if(request.getStatusReq().equals(neigh3)) {

answer = "Air quality in Smithfield is MODERATE (extremely sensitive
individuals limit stay)";

}

if(request.getStatusReq().equals(neigh4)) {

answer = "Air quality in Ranelagh is GOOD (every individual can stay)";

}

if(request.getStatusReq().equals(neigh5)) {

answer = "Air quality in Portobello is GOOD (every individual can stay)";

}

statusResponse response = statusResponse.newBuilder().setStatusRes("--- RPC
1 --- \nAcording to the data base: "+ answer).build();

responseObserver.onNext( response );

responseObserver.onCompleted();

}

@Override

public StreamObserver<airOxygenRequest>
bidiOxygen(StreamObserver<airOxygenResponse> responseObserver) {

// TODO Auto-generated method stub

//return super.bidiOxygen(responseObserver);

return new StreamObserver<airOxygenRequest>() {

@Override

public void onNext(airOxygenRequest value) {

// TODO Auto-generated method stub

// In bidirectional stream, both server and client would be sending the
stream of messages.
```

```
// Here, for each message in stream from client, server is sending back one
response.

// To calculate if the oxygen is healthy or not we use the next equation:

// % O2 in air = (change in height of water in the burette/initial volume
of air in the burette) x 100

StringBuilder value1 = new StringBuilder();

float bef;

float aft;

float change;

float initial;

float calculation;

String message = "";

bef = value.getAirOxygenReqBef();

aft = value.getAirOxygenReqAft();

change = bef-aft;

initial = 50-bef;

calculation = (change/initial)*100;

if(calculation<=23.55 && calculation>=19.5) {

message = "HEALTHY for humans" ;

}else {

message = "UNHEALTHY for humans" ;

}

value1.append((calculation));

float value1AsFloat = Float.parseFloat(value1.toString());

value1 = value1.reverse();

// Preparing and sending the reply for the client. Here, response is build
and with the value (input1.toString()) computed by above logic.

airOxygenResponse response =
airOxygenResponse.newBuilder().setAirOxygenResHea(message).setAirOxygenResP
er(value1AsFloat).build();
```

```
responseObserver.onNext(response);

}

@Override

public void onError(Throwable t) {

// TODO Auto-generated method stub

//Log the error

System.err.println("Error in BidiOxygen: " + t.getMessage());

// Send an error response to the client

responseObserver.onError(t);

}

@Override

public void onCompleted() {

// TODO Auto-generated method

responseObserver.onCompleted();

}

};

}

@Override

public StreamObserver<airDataRequest>
manyAirReadings(StreamObserver<airDataResponse> responseObserver) {

// TODO Auto-generated method stub

//return super.manyAirReadings(responseObserver);

System.out.println("----- Waiting for the carbon monoxide and a ground
level ozone readings from client -----");

final int[] airCount = {0};

final String[] value2 = {""};

return new StreamObserver<airDataRequest>() {

@Override
```

```
public void onNext(airDataRequest value) {
// TODO Auto-generated method stub
if (value.getAirDataReqMon()>=6.5 && value.getAirDataReqMon()<=8) {
airCount[0]++;
}
if (value.getAirDataReqOz()>=20 && value.getAirDataReqOz()<=31.67) {
airCount[0]++;
}
if(airCount[0]==2) {
value2[0] = "The air is clean";
}else{
value2[0] = "The air is polluted";
}
}
@Override
public void onError(Throwable t) {
// TODO Auto-generated method stub
//Log the error
System.err.println("Error in manyAirReadings: " + t.getMessage());
// Send an error response to the client
responseObserver.onError(t);
}
@Override
public void onCompleted() {
// TODO Auto-generated method stub
String message = "Calculating... " + value2[0];
airDataResponse response =
airDataResponse.newBuilder().setAirDataRes(message).build();
```

```
responseObserver.onNext(response);
responseObserver.onCompleted();
}
};
}
}
```

5.3.2 Client

```
package grpc.service2;
import java.net.InetAddress;
import java.util.Random;
import java.util.concurrent.TimeUnit;
import javax.jmdns.JmDNS;
import javax.jmdns.ServiceEvent;
import javax.jmdns.ServiceInfo;
import javax.jmdns.ServiceListener;
import grpc.service2.AirQualityCheckGrpc.AirQualityCheckBlockingStub;
import grpc.service2.AirQualityCheckGrpc.AirQualityCheckStub;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.StatusRuntimeException;
import io.grpc.stub.StreamObserver;
public class AirQualityCheckClient {
private static AirQualityCheckBlockingStub blockingstub;
private static AirQualityCheckStub asyncStub;
static String host="_GRPCServ2._tcp.local.";//"localhost";
static int port;// = 50052;
```

```
static String resolvedIP;

public static void main(String[] args) {

testClientJMDNS();

ManagedChannel channel = ManagedChannelBuilder.forAddress(resolvedIP,
port).usePlaintext().build();

blockingstub = AirQualityCheckGrpc.newBlockingStub(channel);

asyncStub = AirQualityCheckGrpc.newStub(channel);

//Method calls and channel shutdown

neighbourhoodName();

BidiOxygen();

manyAirReadings();

try {

channel.shutdown().awaitTermination(5,TimeUnit.SECONDS);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

private static void testClientJMDNS() {

// TODO Auto-generated method stub

try {

// Create a JmDNS instance

JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

// Add a service listener

jmdns.addServiceListener(host, new SampleListener());

// Wait a bit

Thread.sleep(20000);

} catch (Exception e) {
```

```
System.out.println(e.getMessage());

}

}

private static class SampleListener implements ServiceListener {

public void serviceAdded(ServiceEvent event) {

System.out.println("Service added: " + event.getInfo());

}

public void serviceRemoved(ServiceEvent event) {

System.out.println("Service removed: " + event.getInfo());

}

@SuppressWarnings("deprecation")

public void serviceResolved(ServiceEvent event) {

System.out.println("Service resolved: " + event.getInfo());

ServiceInfo info = event.getInfo();

port = info.getPort();

resolvedIP = info.getHostAddress();

System.out.println("IP Resolved - " + resolvedIP + ":" + port);

}

}

public static void neighbourhoodName() {

// TODO Auto-generated method stub

try {

statusRequest request =
statusRequest.newBuilder().setStatusReq("Churchtown").build();

statusResponse response = blockingstub.neighbourhoodName(request);

System.out.println(response.getStatusRes());

}catch (StatusRuntimeException e) {

e.getStatus();
```

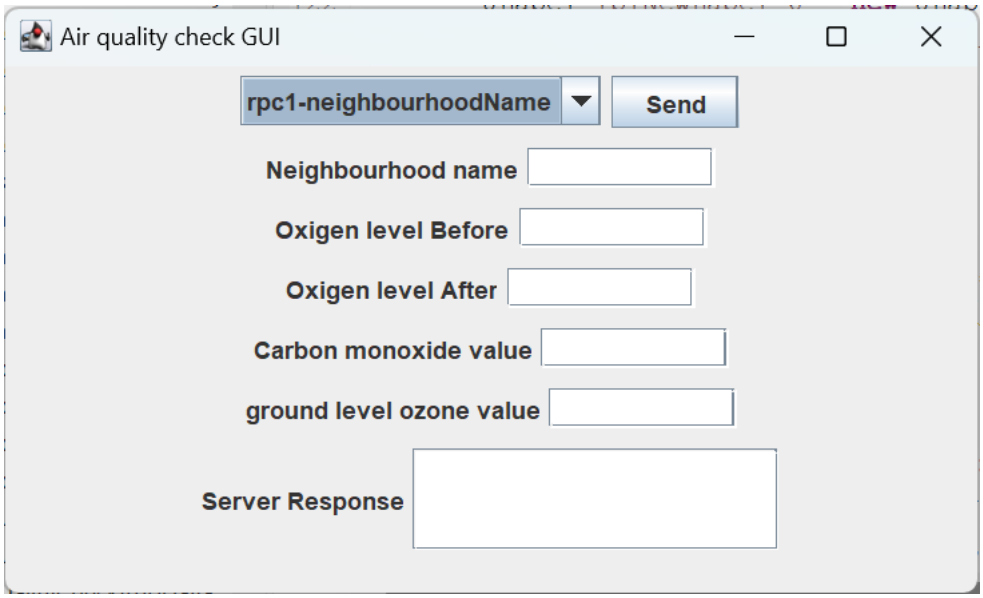
```
}

}

public static void BidiOxygen() {

// TODO Auto-generated method stub

// Handling the server stream for client using onNext (logic for handling
each message in stream), onError, onCompleted (logic will be executed after
the completion of stream)

StreamObserver<airOxygenResponse> airOxyReply = new
StreamObserver<airOxygenResponse>() {

@Override

public void onNext(airOxygenResponse value) {

System.out.println("The percentage of the oxygen in the air is: " +
value.getAirOxygenResPer() + "% which means it is: " +
value.getAirOxygenResHea());

}

@Override

public void onError(Throwable t) {

// TODO Auto-generated method stub

}

@Override

public void onCompleted() {

// TODO Auto-generated method stub

System.out.println("Burette method test finished succesfully");

}

};

// Here, we are calling the Remote reverseStream method. Using onNext,
client sends a stream of messages.

StreamObserver<airOxygenRequest> requestObserver =
asyncStub.bidiOxygen(airOxyReply);

try {

requestObserver.onNext(airOxygenRequest.newBuilder().setAirOxygenReqBef(20)
.setAirOxygenReqAft(14).build());
```

```
requestObserver.onNext(airOxygenRequest.newBuilder().setAirOxygenReqBef(25)
.setAirOxygenReqAft(15).build());

requestObserver.onNext(airOxygenRequest.newBuilder().setAirOxygenReqBef(16)
.setAirOxygenReqAft(8).build());

System.out.println("--- RPC 2 ---");

// Mark the end of requests

requestObserver.onCompleted();

// Sleep for a bit before sending the next one.

Thread.sleep(new Random().nextInt(1000) + 500);

} catch (RuntimeException e) {

e.printStackTrace();

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public static void manyAirReadings() {

// TODO Auto-generated method stub

StreamObserver<airDataResponse> airReply = new
StreamObserver<airDataResponse>() {

@Override

public void onNext(airDataResponse value) {

// TODO Auto-generated method stub

System.out.println("--- RPC 3 --- \nThe status of the air is: " +
value.getAirDataRes());

}

@Override

public void onError(Throwable t) {

// TODO Auto-generated method stub

t.printStackTrace();
```

```
}

@Override

public void onCompleted() {

// TODO Auto-generated method stub

System.out.println("Server finnished");

}

};

StreamObserver<airDataRequest> airRequest =
asyncStub.manyAirReadings(airReply);

try {

airDataRequest request =
airDataRequest.newBuilder().setAirDataReqMon(7).setAirDataReqOz(30).build()
;

airRequest.onNext(request);

Thread.sleep(500);

airRequest.onCompleted();

Thread.sleep(3000);

} catch (RuntimeException | InterruptedException e) {

e.printStackTrace();

}

}

}
```

5.3.3 GUI

In this GUI first we have a rpc method selector and the button to send the data to the server, depending on the service that the client wants to run he/she needs to fill those spaces, it looks like this:



Code

```
package grpc.service2;

import java.awt.EventQueue;

import java.awt.FlowLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.BoxLayout;

import javax.swing.DefaultComboBoxModel;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTextArea;

import javax.swing.JTextField;

import grpc.service2.GUIserv2;

public class GUIserv2 {
```

```
private JFrame frame;
private JTextField textNeigStatus;
private JTextField textOxiBefValue;
private JTextField textOxiAftValue;
private JTextField textDataMonValue;
private JTextField textDataOzValue;
private JTextArea textResponseAirQuality;
/**
* Launch the application.
*/
public static void main(String[] args) {
EventQueue.invokeLater(new Runnable() {
public void run() {
try {
GUIserv2 window = new GUIserv2();
window.frame.setVisible(true);
} catch (Exception e) {
e.printStackTrace();
}
}
});
}
public GUIserv2() {
initialize();
}
/**
* Initialize the contents of the frame.
```

```
*/
private void initialize() {
frame = new JFrame();
frame.setTitle("Air quality check GUI");
frame.setBounds(100, 100, 500, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
BoxLayout bl = new BoxLayout(frame.getContentPane(), BoxLayout.Y_AXIS);
frame.getContentPane().setLayout(bl);
//panel 1
JPanel panel_service_1 = new JPanel();
frame.getContentPane().add(panel_service_1);
panel_service_1.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
//panel 2
JPanel panel_service_2 = new JPanel();
frame.getContentPane().add(panel_service_2);
panel_service_2.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel lblNewLabel_2 = new JLabel("Neighbourhood name");
panel_service_2.add(lblNewLabel_2);
textNeigStatus = new JTextField();
panel_service_2.add(textNeigStatus);
textNeigStatus.setColumns(10);
//panel 3
JPanel panel_service_3 = new JPanel();
frame.getContentPane().add(panel_service_3);
panel_service_3.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel lblNewLabel_3 = new JLabel("Oxigen level Before");
panel_service_3.add(lblNewLabel_3);
```

```
textDataOzValue = new JTextField();
panel_service_3.add(textDataOzValue);
textDataOzValue.setColumns(10);
//panel 4
JPanel panel_service_4 = new JPanel();
frame.getContentPane().add(panel_service_4);
panel_service_4.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel lblNewLabel_4 = new JLabel("Oxigen level After");
panel_service_4.add(lblNewLabel_4);
textOxiBefValue = new JTextField();
panel_service_4.add(textOxiBefValue);
textOxiBefValue.setColumns(10);
//panel 5
JPanel panel_service_5 = new JPanel();
frame.getContentPane().add(panel_service_5);
panel_service_5.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel lblNewLabel_5 = new JLabel("Carbon monoxide value");
panel_service_5.add(lblNewLabel_5);
textOxiAftValue = new JTextField();
panel_service_5.add(textOxiAftValue);
textOxiAftValue.setColumns(10);
//panel 6
JPanel panel_service_6 = new JPanel();
frame.getContentPane().add(panel_service_6);
panel_service_6.setLayout(new FlowLayout(FlowLayout.CENTER, 5, 5));
JLabel lblNewLabel_6 = new JLabel("ground level ozone value");
panel_service_6.add(lblNewLabel_6);
```

```
textDataMonValue = new JTextField();

panel_service_6.add(textDataMonValue);

textDataMonValue.setColumns(10);

//

final JComboBox comboOperation = new JComboBox();

comboOperation.setModel(new DefaultComboBoxModel(new String[] {"rpc1-
neighbourhoodName", "rpc2-BidiOxygen", "rpc3-manyAirReadings"}));

panel_service_1.add(comboOperation);

JButton btnCalculate = new JButton("Send");

btnCalculate.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

float num1 = Float.parseFloat(textNeigStatus.getText());

float num2 = Float.parseFloat(textOxiBefValue.getText());

float num3 = Float.parseFloat(textOxiAftValue.getText());

float num4 = Float.parseFloat(textDataMonValue.getText());

float num5 = Float.parseFloat(textDataOzValue.getText());

String name = textResponseAirQuality.getText();

int index = comboOperation.getSelectedIndex();

if (index == 0) {

textResponseAirQuality.append("Result of Subtraction" + String.valueOf(num1
- num2));

}

if (index == 1) {

textResponseAirQuality.append("Result of Subtraction" + String.valueOf(num1
- num2));

}

if (index == 2) {

textResponseAirQuality.append("Result of Multiplication" +
String.valueOf(num1 * num2));
```

```
}
}
});
panel_service_1.add(btnCalculate);
textResponseAirQuality = new JTextArea(3, 20);
textResponseAirQuality .setLineWrap(true);
textResponseAirQuality.setWrapStyleWord(true);
JScrollPane scrollPane = new JScrollPane(textResponseAirQuality);
//textResponse.setSize(new Dimension(15, 30));
JPanel panel_service_7 = new JPanel();
JLabel lblNewLabel_7 = new JLabel("Server Response");
panel_service_7.add(lblNewLabel_7);
frame.getContentPane().add(panel_service_7);
panel_service_7.add(scrollPane);
JPanel panel_service_8 = new JPanel();
frame.getContentPane().add(panel_service_8);
JPanel panel_service_9 = new JPanel();
frame.getContentPane().add(panel_service_9);
}
}
```

5.4 GRPC for Area Status Control

5.4.1 Server

```
@Override
package grpc.service3;
```

```
import java.io.FileInputStream;

import java.io.IOException;

import java.io.InputStream;

import java.net.InetAddress;

import java.util.Properties;

import java.util.logging.Logger;


import javax.jmdns.JmDNS;

import javax.jmdns.ServiceInfo;


import grpc.examples.bidirectionstreamstrings.StringServer;

import grpc.service3.AreaStatusControlGrpc.AreaStatusControlImplBase;

import io.grpc.Server;

import io.grpc.ServerBuilder;

import io.grpc.stub.StreamObserver;


public class AreaStatusControlServer extends AreaStatusControlImplBase {

      private static final Logger logger =
Logger.getLogger(StringServer.class.getName());

      public static void main(String[] args) {

            AreaStatusControlServer serv3Areaserver = new
AreaStatusControlServer();

            //int port = 50053;


            // Registration

            //

            Properties prop = serv3Areaserver.getProperties();

            serv3Areaserver.registerService(prop);
```

```
            int port = Integer.valueOf(prop.getProperty("service_port"));//
#.50053;

            Server server;

            //

            try {

                  server =
ServerBuilder.forPort(port).addService(serv3Areaserver).build().start();

                  System.out.println("Server 3 working");

                  server.awaitTermination();

            } catch (IOException | InterruptedException e) {

                  // TODO Auto-generated catch block

                  e.printStackTrace();

            }

      }//

      private Properties getProperties() {


            Properties prop = null;


            try (InputStream input = new
FileInputStream("src/main/resources/Serv3_AreaStatusControl.properties")) {

                  prop = new Properties();


                  // load a properties file

                  prop.load(input);


                  // get the property value and print it out

                  System.out.println("Service 3 - Pollution WatchArea: ");
```

```
                  System.out.println("\t service_type: " +
prop.getProperty("service_type"));

                  System.out.println("\t service_name: " +
prop.getProperty("service_name"));

                  System.out.println("\t service_description: " +
prop.getProperty("service_description"));

                  System.out.println("\t service_port: " +
prop.getProperty("service_port"));



            } catch (IOException ex) {

                  ex.printStackTrace();

            }

            return prop;

      }



      private void registerService(Properties prop) {

            try {

                  // Create a JmDNS instance

                  JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());



                  String service_type = prop.getProperty("service_type");//
"_GRPCServ3._tcp.local.";

                  String service_name = prop.getProperty("service_name");//
"Area status control";

                  int service_port =
Integer.valueOf(prop.getProperty("service_port"));// #.50053;



                  String service_description_properties =
prop.getProperty("service_description");// "Description"



                  // Registration service information
```

```
                ServiceInfo serviceInfo =
ServiceInfo.create(service_type, service_name, service_port,

                                service_description_properties);

                jmdns.registerService(serviceInfo);



                System.out.printf("registrering service with type %s and
name %s \n", service_type, service_name);



                // Wait a bit for the service to register

                Thread.sleep(1000);



            } catch (IOException e) {

                System.out.println(e.getMessage());

            } catch (InterruptedException e) {

                // TODO Auto-generated catch block

                e.printStackTrace();

            }

      }//

      @Override

      public void riverHistoric(riverDataRequest request,
StreamObserver<riverDataResponse> responseObserver) {

            // TODO Auto-generated method stub

            //super.riverHistoric(request, responseObserver);



             String[] RiverShannon = {"unacceptable", "acceptable",
"acceptable", "acceptable", "acceptable", "acceptable", "unacceptable",
"acceptable", "acceptable", "acceptable"};

             String[] RiverBarrow = {"acceptable", "acceptable",
"unacceptable", "acceptable", "acceptable", "acceptable", "acceptable",
"unacceptable", "acceptable", "unacceptable"};
```

```
            String[] RiverLiffey = {"acceptable", "unacceptable",
"unacceptable", "acceptable", "unacceptable", "acceptable", "acceptable",
"acceptable", "unacceptable", "acceptable"};



            System.out.println("receiving name of the river...");



            if(request.getRiverDataReq().equalsIgnoreCase("Shannon")) {

                  System.out.println("Sending the result of the last 10
days of the RiverShannon");

                  for(int i=1; i<=RiverShannon.length; i++) {


responseObserver.onNext(riverDataResponse.newBuilder().setRiverDataRes("Day
" + i + " pollution levels are " + RiverShannon[i-1]).build());

                }



            }else if(request.getRiverDataReq().equalsIgnoreCase("Barrow"))
{

                  System.out.println("Sending the result of the last 10
days of the RiverShannon");

                  for(int i=1; i<=RiverBarrow.length; i++) {


responseObserver.onNext(riverDataResponse.newBuilder().setRiverDataRes("Day
" + i + " pollution levels are " + RiverBarrow[i-1]).build());

                }



            }else if(request.getRiverDataReq().equalsIgnoreCase("Liffey"))
{

                  System.out.println("Sending the result of the last 10
days of the RiverShannon");

                  for(int i=1; i<=RiverLiffey.length; i++) {


responseObserver.onNext(riverDataResponse.newBuilder().setRiverDataRes("Day
" + i + " pollution levels are " + RiverLiffey[i-1]).build());

                }
```

```
            }else {

                  System.out.println("Client send River " +
request.getRiverDataReq() + ", this one is not on the database");


responseObserver.onNext(riverDataResponse.newBuilder().setRiverDataRes("Riv
er " + request.getRiverDataReq() + " is not in the database").build());

            }

            responseObserver.onCompleted();

      }

      @Override

      public StreamObserver<waterStatusRequest>
bidiRiver(StreamObserver<waterStatusResponse> responseObserver) {

            // TODO Auto-generated method stub

            //return super.bidiRiver(responseObserver);

            return new StreamObserver<waterStatusRequest>() {


                  @Override

                  public void onNext(waterStatusRequest value) {

                        // TODO Auto-generated method stub


                              StringBuilder value2 = new StringBuilder();

                              String RiverName;

                              String message = "";

                              System.out.println("receiving name of the
river...");

                              RiverName = value.getWaterStatusReq();



                              if(RiverName.equalsIgnoreCase("Liffey")) {
```

```
                                    message = "River Liffey quick
info:\nLength: 132 km\nPollution: HIGH\nSwimming: Not advisable\nFishing:
LOW, only in some areas" ;

                                }else
if(RiverName.equalsIgnoreCase("Barrow")) {

                                      message = "River Barrow quick
info:\nLength: 192 km\nPollution: LOW\nSwimming: Yes, check for
areas\nFishing: HIGH, many areas" ;

                                }else
if(RiverName.equalsIgnoreCase("Shannon")) {

                                      message = "River Barrow quick
info:\nLength: 360 km\nPollution: MID depending on the area\nSwimming: Yes,
check for areas\nFishing: MID, some areas" ;

                                }else {

                                      message = "No information of this river
is available" ;

                                      System.out.println("Client input river
is not in database");

                                }



                               waterStatusResponse response =
waterStatusResponse.newBuilder().setWaterStatusRes(message).build();


                            responseObserver.onNext(response);

                  }



                  @Override

                  public void onError(Throwable t) {

                        // TODO Auto-generated method stub

                        //Log the error

                        System.err.println("Error in BidiRiver: " +
t.getMessage());

                  // Send an error response to the client

                  responseObserver.onError(t);
```

```
                }

                @Override
                public void onCompleted() {
                        // TODO Auto-generated method
                        responseObserver.onCompleted();

                }

        };
    }
    @Override
    public StreamObserver<neighborhoodStatusRequest> bidiNeighbourhood(
                StreamObserver<neighborhoodStatusResponse>
responseObserver) {
        // TODO Auto-generated method stub
        //return super.bidiNeighbourhood(responseObserver);
        return new StreamObserver<neighborhoodStatusRequest>() {

                @Override
                public void onNext(neighborhoodStatusRequest value) {
                        // TODO Auto-generated method stub

                                StringBuilder value2 = new StringBuilder();
                                String NeighborhoodName;
                                String message = "";
                                System.out.println("receiving name of the
neighborhood...");
```

```
                                NeighborhoodName =
value.getNeighborhoodStatusReq();




      if(NeighborhoodName.equalsIgnoreCase("Churchtown")) {

                                  message = "Neighborhood Churchtown quick
info:\nEirCode: D14\nSafety: High\nPollution: LOW\nPeople: 8,736";

                                }else
if(NeighborhoodName.equalsIgnoreCase("Temple bar")) {

                                      message = "Neighborhood Temple bar
quick info:\nEirCode: D02\nSafety: Mid, many bars\nPollution: High\nPeople:
3,000";

                                }else
if(NeighborhoodName.equalsIgnoreCase("Ranelagh")) {

                                      message = "Neighborhood Ranelagh quick
info:\nEirCode: D06\nSafety: High\nPollution: very Low\nPeople: 1,268" ;

                                }else {

                                      message = "No information of this
Neighborhood is available" ;

                                      System.out.println("Client input
Neighborhood is not in database");

                                }



                                neighborhoodStatusResponse response =
neighborhoodStatusResponse.newBuilder().setNeighborhoodStatusRes(message).b
uild();

                            responseObserver.onNext(response);

                  }



                  @Override

                  public void onError(Throwable t) {

                        // TODO Auto-generated method stub

                        //Log the error
```

```
                        System.err.println("Error in Bidineighborhood: " +
t.getMessage());
                    // Send an error response to the client
                    responseObserver.onError(t);
                    }
public void onCompleted() {
// TODO Auto-generated method
responseObserver.onCompleted();
}
};
}
}
```

5.4.2 Client

```
// Mark the end of requests
requestObserver.onCompleted();
// Sleep for a bit before sending the next one.
Thread.sleep(new Random().nextInt(1000) + 500);
} catch (RuntimeException e) {
e.printStackTrace();
} catch (InterruptedException e) {
e.printStackTrace();
}
}
}
```

5.4.3 GUI

In this GUI first we have a rpc method selector and the button to send the data to the server, depending on the service that the client wants to run he/she needs to fill those spaces, it looks like this:

Code:

```
package grpc.service3;


import java.awt.EventQueue;

import java.awt.FlowLayout;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;


import javax.swing.BoxLayout;

import javax.swing.DefaultComboBoxModel;

import javax.swing.JButton;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;
```

```
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;

import grpc.service3.GUIserv3;

public class GUIserv3 {
	private JFrame frame;
	private JTextField textRiverData;
	private JTextField textWatStatValue;
	private JTextField textNeiStatValue;
	private JTextArea textResponseAreaStat;

	/**
	 * Launch the application.
	 */
	public static void main(String[] args) {
		EventQueue.invokeLater(new Runnable() {
			public void run() {
				try {
					GUIserv3 window = new GUIserv3();
					window.frame.setVisible(true);
				} catch (Exception e) {
					e.printStackTrace();
				}
			}
		});
```

```
	}
	public GUIserv3() {




		initialize();
	}



	/**
	 * Initialize the contents of the frame.
	 */
	private void initialize() {
		frame = new JFrame();
		frame.setTitle("Area Status Control GUI");
		frame.setBounds(100, 100, 500, 300);
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


		BoxLayout bl = new BoxLayout(frame.getContentPane(),
BoxLayout.Y_AXIS);


		frame.getContentPane().setLayout(bl);
		//panel 1
		JPanel panel_service_1 = new JPanel();
		frame.getContentPane().add(panel_service_1);
		panel_service_1.setLayout(new FlowLayout(FlowLayout.CENTER, 5,
5));
```

```
            //panel 2
            JPanel panel_service_2 = new JPanel();
            frame.getContentPane().add(panel_service_2);
            panel_service_2.setLayout(new FlowLayout(FlowLayout.CENTER, 5,
5));

            JLabel lblNewLabel_2 = new JLabel("River Name");
            panel_service_2.add(lblNewLabel_2);

            textRiverData = new JTextField();
            panel_service_2.add(textRiverData);
            textRiverData.setColumns(10);
            //panel 3

            //panel 4
            JPanel panel_service_4 = new JPanel();
            frame.getContentPane().add(panel_service_4);
            panel_service_4.setLayout(new FlowLayout(FlowLayout.CENTER, 5,
5));

            JLabel lblNewLabel_4 = new JLabel("River Name 2");
            panel_service_4.add(lblNewLabel_4);

            textWatStatValue = new JTextField();
            panel_service_4.add(textWatStatValue);
            textWatStatValue.setColumns(10);
            //panel 5
            JPanel panel_service_5 = new JPanel();
```

```
            frame.getContentPane().add(panel_service_5);

            panel_service_5.setLayout(new FlowLayout(FlowLayout.CENTER, 5,
5));



            JLabel lblNewLabel_5 = new JLabel("Neighbourhood Name");

            panel_service_5.add(lblNewLabel_5);



            textNeiStatValue = new JTextField();

            panel_service_5.add(textNeiStatValue);

            textNeiStatValue.setColumns(10);

            //panel 6



            //




            final JComboBox comboOperation = new JComboBox();

            comboOperation.setModel(new DefaultComboBoxModel(new String[]
{"rpc1-neighbourhoodName", "rpc2-BidiOxygen", "rpc3-manyAirReadings"}));

            panel_service_1.add(comboOperation);




            JButton btnCalculate = new JButton("Send");

            btnCalculate.addActionListener(new ActionListener() {

                  public void actionPerformed(ActionEvent e) {



                        float num1 =
Float.parseFloat(textRiverData.getText());

                        float num2 =
Float.parseFloat(textWatStatValue.getText());
```

```
                        float num3 =
Float.parseFloat(textNeiStatValue.getText());

                        String name = textResponseAreaStat.getText();



                        int index = comboOperation.getSelectedIndex();



                        if (index == 0) {

                              textResponseAreaStat.append("Result of
Subtraction" + String.valueOf(num1 - num2));

                              }

                        if (index == 1) {

                              textResponseAreaStat.append("Result of
Subtraction" + String.valueOf(num1 - num2));

                              }

                        if (index == 2) {

                              textResponseAreaStat.append("Result of
Multiplication" + String.valueOf(num1 * num2));

                              }



                  }

            });

panel_service_1.add(btnCalculate);

textResponseAreaStat = new JTextArea(3, 20);

textResponseAreaStat .setLineWrap(true);

textResponseAreaStat.setWrapStyleWord(true);

JScrollPane scrollPane = new JScrollPane(textResponseAreaStat);

//textResponse.setSize(new Dimension(15, 30));

JPanel panel_service_7 = new JPanel();

JLabel lblNewLabel_7 = new JLabel("Server Response");
```

```
panel_service_7.add(lblNewLabel_7);
frame.getContentPane().add(panel_service_7);
panel_service_7.add(scrollPane);
JPanel panel_service_8 = new JPanel();
frame.getContentPane().add(panel_service_8);
JPanel panel_service_9 = new JPanel();
frame.getContentPane().add(panel_service_9);
}
}
```

6 GitHub Repo

The project can be found in the public GitHub repository:

https://github.com/LuisNavarroT/CA-DistributedSystems-LuisNavarro-PollutionWatch