Distributed Systems

March 2023

Higher Diploma in Science in Computing

21219842

Chungman Lee

1. Service Definitions

This pollution tracker system is made to help users monitor the pollution. The aim of the services and methods are following.

1.1 Service 1: Waterpllutiontracker

1.1.1 GetWaterPollutionHistory(Server-side streaming RPC)

rpc GetWaterPollutionHistory (WaterPollutionHistoryRequest) **returns** (**stream** WaterPollutionLevel) {}

This method will receive the start time, end time and location from the client and show the water pollution information about the location such as the location, pollution type, pollution level, and time.

1.1.2 GetWaterPollutionAlerts(Unary RPC)

rpc GetWaterPollutionAlerts (WaterPollutionAlertsRequest) **returns** (WaterPollutionAlertsResponse) {}

This method will receive the location and threshold from client and show the alert message if the location has water pollution level larger than the threshold. The message has location, pollution type, pollution level and timestamp.

1.1.3 MonitorWaterPollution (Bidirectional streaming RPC)

rpc MonitorWaterPollution (**stream** WaterPollutionMonitorRequest) **returns** (**stream** WaterPollutionLevel) {}

This method will receive the locations from the client multiply as a stream with two text boxes, and it will return the stream of the water pollution information of the monitored locations.

1.2 Service 2: Air Pollution Tracker

This service is basically twins with Water Pollution Tracker.

1.2.1 GetAirPollutionHistory (Server- side streaming RPC0

rpc GetAirPollutionHistory (AirPollutionHistoryRequest) **returns** (**stream**

AirPollutionLevel) {}

This method will receive the start time, end time and location from the client and show the air pollution information about the location such as the location, pollution type, pollution level, and time.

1.2.2 GetAirPollutionAlerts (Unary RPC)

rpc GetAirPollutionAlerts (AirPollutionAlertsRequest) **returns** (AirPollutionAlertsResponse) {}

This method will receive the location and threshold from client and show the alert message if the location has air pollution level larger than the threshold. The message has location, pollution type, pollution level and timestamp.

1.2.3 MonitorWaterPollution (Bidirectional streaming RPC)

rpc MonitorAirPollution (**stream** AirPollutionMonitorRequest) **returns** (**stream** AirPollutionLevel) {}

This method will receive the locations from the client multiply as a stream with two text boxes, and it will return the stream of the air pollution information of the monitored locations.

1.3 Service 3: Data Visualization Service

1.3.1 GetPollutionStatistics (Server-side streaming RPC)

rpc GetPollutionStatistics (PollutionStatisticsRequest) **returns** (**stream** PollutionStatistics) {}

This method receive location from the client and return the calculated pollution statistics especially average pollution level and highest pollution level of the region.

1.3.2 2 FilterDataByLocation (Bidirectional streaming RPC)

rpc FilterDataByLocation (**stream** LocationFilterRequest) **returns** (**stream** PollutionLevel) {}

This method will receive maximum locations from the client and filter the data by location and show it to the client returning a stream of pollution data such as location, pollution type, pollution level, and time.

1.3.3 SetFavoriteLocation (Client-side streaming service)

rpc SetFavoriteLocation (**stream** FavoriteLocationRequest) **returns** (**stream** FavoriteLocationResponse) {}

This method allows the client to do the streaming sending locations. The server will return the success response.

2. Service Implementations

I made 3 server and 3 clients as 3 pairs. Combining all the GUI makes the code too long so just separated.

WaterPollutionTrackerServer

The main method starts with creating an instance of server and start it. And I registered it with JmDNS. It will wait until the server terminates.

In terms of generateSampleDataHistory method, it makes some sample data. will receive a start time and end time and location as inputs and creates a list of WaterPollutionLevel objects with random pollution levels for Dublin 1-10. generateSmapleAlerts and generateSampleDataMonitor are also similar with the generateSampleAlerts. It will generates sample data for GetWaterPollutionAlerts and MonitorWaterPollutionrpc.

getWaterPollutionHistory method is implementing the GetWaterPolluionHistory rpc. It takes a WaterPollutionHistoryRequest as input and a StreamObserver<WaterPollutionLevel> as output. It receives pollution levels from generateSampleDataHistory, filters the data by location and time, and sends the filtered data to the client using the StreamObserver.

getWaterPollutionAlerts method is also one of the main methods of this class. This method implements the GetWaterPollutionAlerts rpc and it takes request as input which is generated by the information based on the specifics provided by the client.

monitorWaterPollution is the third main method of this class. This method will receive

StreamObserver<WaterPollutionLevel> as input and returns StreamObserver<WaterPollutionMonitorRequest>.

Other methods are for JmDNS.

In the discoverServiceWithJmdns, I made the function using JmDNS livrary to discover the server's IP address and port number. It adds service listener that listens for service events, and when a service is resolved, it use resolvedIP and port as variables.

In the runClientGui class, I made some GUI using swings for clients which is containing three methods buttons.

Mian method calls discoverServiceWithJmDNS() to discover the server, waits for the serviceResolvedLatch countdown latch to be triggered, and then calls runClientGui(resolvedIP, port) to launch the client GUI.

2.1 AirPollutionTracker

It is basically same logic with the Water PollutionTrackerServer.

It is basically same logic with the Water WaterPollutionTrackerClient.

2.2 DataVisualization

getPollutionStatistics method accepts a PollutionStatisticsRequest message that contains a location, and responds with a PollutionStatistics message that contains the average and highest pollution levels for that location.

filterDataByLocation method accepts a stream of LocationFilterRequest messages that contain a location and responds with a stream of PollutionLevel messages that match that location.

setFavoriteLocation method accepts a stream of FavoriteLocationRequest messages that contain a location and responds with a single FavoriteLocationResponse message indicating that the location has been saved.

The most different thing from other servers is it cultulates the statistics and has Client-side streaming service.

3. Naming Services.

I used JmDNS for naming service basically. Wrote some dependency in the pom file and added cods in server side and client side. sometimes

4. Remote Error Handling & Advanced Features

I used try and catch sentence to deal with exceptions and imported IOException and so on. And used Deadline function also to deal with error. I will return error message if the deadline is over. It will limit the due time of response. It will be useful when interacting with other computer or system because the client don't have to wait forever.

5. Client - Graphical User Interface (GUI)

Basically I mae 3 gui and each gui is containing 3 parts for each method. And it pisplay the contends of the GUI in a row. Some of the methods has multiple boxes to receive multiple information for the streaming services.

6. GitHub

https://github.com/ChungmanLee/CA_Distributed_System

7. Source Code

i. Waterpollutiontracker.proto

```
syntax = "proto3";
//Options for the GRPC
option java_multiple_files = true;
option java_package = "ds.waterpollutiontracker";
option java_outer_classname = "WaterPollutionTrackerImpl";
package waterpollutiontracker;

service WaterPollutionTracker {
// RPC to retrieve historical water pollution data
	rpc GetWaterPollutionHistory (WaterPollutionHistoryRequest) returns
(stream
	WaterPollutionLevel) {
	}
	// RPC to retrieve water pollution alerts triggered by the system
	rpc GetWaterPollutionAlerts (WaterPollutionAlertsRequest) returns
	(WaterPollutionAlertsResponse) {
	}
	// Bidirectional streaming RPC to monitor water pollution in real-
time
	rpc MonitorWaterPollution (stream WaterPollutionMonitorRequest)
returns (stream
	WaterPollutionLevel) {
	}
}

message WaterPollutionHistoryRequest {
	int64 start_time = 1;
	int64 end_time = 2;
	string location = 3;
}

message WaterPollutionAlertsRequest {
	int32 threshold = 1;
	string location = 2;
}

message WaterPollutionAlertsResponse {
	repeated WaterPollutionAlert alerts = 1;
}

message WaterPollutionAlert {
	string location = 1;
	string pollution_type = 2;
	float pollution_level = 3;
	string timestamp = 4;
}

message WaterPollutionMonitorRequest {
	string location = 1;
}

message WaterPollutionLevel {
	string location = 1;
	string pollution_type = 2;
	float pollution_level = 3;
```

```
        string timestamp = 4;
}
```

ii. Airpollutiontracker.proto

```
syntax = "proto3";
//Options for the GRPC
option java_multiple_files = true;
option java_package = "ds.airpollutiontracker";
option java_outer_classname = "AirPollutionTrackerImpl";
package airpollutiontracker;

service AirPollutionTracker {
// RPC to retrieve historical air pollution data
        rpc GetAirPollutionHistory (AirPollutionHistoryRequest) returns
(stream
        AirPollutionLevel) {
        }
        // RPC to retrieve air pollution alerts triggered by the system
        rpc GetAirPollutionAlerts (AirPollutionAlertsRequest) returns
        (AirPollutionAlertsResponse) {
        }
        // Bidirectional streaming RPC to monitor air pollution in real-time
        rpc MonitorAirPollution (stream AirPollutionMonitorRequest) returns
(stream
        AirPollutionLevel) {
        }
}

message AirPollutionHistoryRequest {
        int64 start_time = 1;
        int64 end_time = 2;
        string location = 3;
}

message AirPollutionAlertsRequest {
        int32 threshold = 1;
        string location = 2;
}

message AirPollutionAlertsResponse {
        repeated AirPollutionAlert alerts = 1;
}

message AirPollutionAlert {
        string location = 1;
        string pollution_type = 2;
        float pollution_level = 3;
        string timestamp = 4;
}

message AirPollutionMonitorRequest {
        string location = 1;
}

message AirPollutionLevel {
```

```
        string location = 1;
        string pollution_type = 2;
        float pollution_level = 3;
        string timestamp = 4;
}
```

iii. Datavisualizer.proto

```
syntax = "proto3";
//Options for the GRPC
option java_multiple_files = true;
option java_package = "ds.datavisualizer";
option java_outer_classname = "DataVisualizerImpl";
package datavisualizer;

service DataVisualization {
// Server-side streaming RPC to retrieve pollution statistics
        rpc GetPollutionStatistics (PollutionStatisticsRequest) returns
(stream
        PollutionStatistics) {
        }
        // Bidirectional streaming RPC to filter pollution data by location
        rpc FilterDataByLocation (stream LocationFilterRequest) returns
(stream
        PollutionLevel) {
        }
        // Client-side streaming RPC to set favorite locations
        rpc SetFavoriteLocation (stream FavoriteLocationRequest) returns
(stream
        FavoriteLocationResponse) {
        }
}

message PollutionStatisticsRequest {
    string location = 1;
}

message PollutionStatistics {
    float averagePollutionLevel = 1;
    float highestPollutionLevel = 2;
}

message LocationFilterRequest {
    string location = 1;
}

message FavoriteLocationRequest {
    string location = 1;
}

message FavoriteLocationResponse {
    string status = 1;
}

message PollutionLevel {
    string location = 1;
```

```
    string pollutionType = 2;
    float pollutionLevel = 3;
    int64 timestamp = 4;
}
```

vi. WaterPollutionTrackerServer.java

```
package ds.waterpollutiontracker;

import java.io.IOException;
import java.net.InetAddress;
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

import javax.jmdns.JmDNS;
import javax.jmdns.ServiceInfo;

import ds.waterpollutiontracker.WaterPollutionTrackerGrpc.WaterPollutionTrackerImplBase;
import io.grpc.Server;
import io.grpc.ServerBuilder;
import io.grpc.stub.StreamObserver;

public class WaterPollutionTrackerServer extends WaterPollutionTrackerImplBase {
    static int port = 50084;
```

```
    public static void main(String[] args) throws InterruptedException, IOException {
        WaterPollutionTrackerServer wTracker = new WaterPollutionTrackerServer();

        Server server;
        try {
            server = ServerBuilder.forPort(port)
                    .addService(wTracker)
                    .build()
                    .start();

            System.out.println("WaterPollutionTracker started, listening on " + port);

            registerWithJmDNS();
            server.awaitTermination();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void getWaterPollutionHistory(WaterPollutionHistoryRequest request,
                                         StreamObserver<WaterPollutionLevel>
responseObserver) {
        Instant startTime = Instant.ofEpochMilli(request.getStartTime());
```

```
        Instant endTime = Instant.ofEpochMilli(request.getEndTime());
        String location = request.getLocation();

        // Pass startTime and endTime to generateSampleData()
        List<WaterPollutionLevel> pollutionLevels =
generateSampleDataHistory(startTime, endTime);

        List<WaterPollutionLevel> filteredData = pollutionLevels.stream()
                .filter(level -> level.getLocation().equals(location))
                .filter(level -> Instant.parse(level.getTimestamp()).isAfter(startTime))
                .filter(level -> Instant.parse(level.getTimestamp()).isBefore(endTime))
                .collect(Collectors.toList());

        for (WaterPollutionLevel level : filteredData) {
            responseObserver.onNext(level);
        }
        responseObserver.onCompleted();
    }

    @Override
    public void getWaterPollutionAlerts(WaterPollutionAlertsRequest request,
StreamObserver<WaterPollutionAlertsResponse> responseObserver) {
        // TODO: Replace this with actual data retrieval from the database
        List<WaterPollutionAlert> alerts = generateSampleAlerts();
```

```
        int threshold = request.getThreshold();
        String location = request.getLocation();

        List<WaterPollutionAlert> filteredAlerts = alerts.stream()
                .filter(alert -> alert.getLocation().equals(location))
                .filter(alert -> alert.getPollutionLevel() > threshold)
                .collect(Collectors.toList());

        WaterPollutionAlertsResponse response =
WaterPollutionAlertsResponse.newBuilder()
                .addAllAlerts(filteredAlerts)
                .build();

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

    @Override
    public StreamObserver<WaterPollutionMonitorRequest> monitorWaterPollution(
            StreamObserver<WaterPollutionLevel> responseObserver) {
        // TODO: Replace this with actual real-time data monitoring implementation
        return new StreamObserver<WaterPollutionMonitorRequest>() {
            @Override
            public void onNext(WaterPollutionMonitorRequest request) {
                String location = request.getLocation();
```

```
                // Generate sample data for demonstration purposes
                List<WaterPollutionLevel> pollutionLevels = generateSampleDataMonitor();

                for (WaterPollutionLevel level : pollutionLevels) {
                    if (level.getLocation().equals(location)) {
                        responseObserver.onNext(level);
                    }
                }
            }

            @Override
            public void onError(Throwable t) {
                System.out.println("Error encountered in WaterPollutionTrackerServer: " + t.getMessage());
            }

            @Override
            public void onCompleted() {
                responseObserver.onCompleted();
            }
        };
    }

    private List<WaterPollutionLevel> generateSampleDataHistory(Instant startTime, Instant endTime) {
```

```
        // This method generates sample data for demonstration purposes
        List<WaterPollutionLevel> pollutionLevels = new ArrayList<>();
        Random random = new Random();
        long duration = endTime.toEpochMilli() - startTime.toEpochMilli();
        for (int i = 1; i <= 10; i++) {
            for (int j = 1; j <= 3; j++) {
                long randomTimestamp = startTime.toEpochMilli() + (long) (random.nextDouble() * duration);
                Instant sampleTimestamp = Instant.ofEpochMilli(randomTimestamp);
                WaterPollutionLevel level = WaterPollutionLevel.newBuilder()
                        .setLocation("Dublin " + i)
                        .setPollutionType("Pollution type " + i)
                        .setPollutionLevel(random.nextFloat() * 200)
                        .setTimestamp(sampleTimestamp.toString())
                        .build();
                pollutionLevels.add(level);
            }
        }
        return pollutionLevels;
    }

    private List<WaterPollutionLevel> generateSampleDataMonitor() {
        // This method generates sample data for demonstration purposes
        List<WaterPollutionLevel> pollutionLevels = new ArrayList<>();
        Random random = new Random();
        for (int i = 1; i <= 10; i++) {
```

```
            WaterPollutionLevel level = WaterPollutionLevel.newBuilder()
                    .setLocation("Dublin " + i)
                    .setPollutionType("Pollution type " + i)
                    .setPollutionLevel(random.nextFloat() * 200)
                    .setTimestamp(Instant.now().minusSeconds(random.nextInt(3600)).to
String())
                    .build();
            pollutionLevels.add(level);
        }
        for (int i = 1; i <= 10; i++) {
            WaterPollutionLevel level = WaterPollutionLevel.newBuilder()
                    .setLocation("Dublin " + i)
                    .setPollutionType("Pollution type " + i)
                    .setPollutionLevel(random.nextFloat() * 200)
                    .setTimestamp(Instant.now().minusSeconds(random.nextInt(3600)).to
String())
                    .build();
            pollutionLevels.add(level);
        }
        for (int i = 1; i <= 10; i++) {
            WaterPollutionLevel level = WaterPollutionLevel.newBuilder()
                    .setLocation("Dublin " + i)
                    .setPollutionType("Pollution type " + i)
                    .setPollutionLevel(random.nextFloat() * 200)
                    .setTimestamp(Instant.now().minusSeconds(random.nextInt(3600)).to
String())
```

```
                .build();
            pollutionLevels.add(level);
        }
        return pollutionLevels;
    }

    private List<WaterPollutionAlert> generateSampleAlerts() {
        // This method generates sample alerts for demonstration purposes
        List<WaterPollutionAlert> alerts = new ArrayList<>();
        Random random = new Random();
        for (int i = 1; i <= 5; i++) {
            WaterPollutionAlert alert = WaterPollutionAlert.newBuilder()
                .setLocation("Dublin " + i)
                .setPollutionType("Pollution type " + i)
                .setPollutionLevel(random.nextFloat() * 300)
                .setTimestamp(Instant.now().minusSeconds(random.nextInt(3600)).to
String())
                .build();
            alerts.add(alert);
        }
        return alerts;
    }

 // Add the JmDNS registration method here
    public static void registerWithJmDNS() {
        try {
```

```
            // Create a JmDNS instance
            JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

            // Register a service
            ServiceInfo serviceInfo = ServiceInfo.create("_http._tcp.local.", "water-pollution-tracker", port, "WaterPollutionTracker service");
            jmdns.registerService(serviceInfo);

            // Wait a bit
            Thread.sleep(20000);

            // Unregister all services
            // jmdns.unregisterAllServices();
            Runtime.getRuntime().addShutdownHook(new Thread(() -> {
                jmdns.unregisterAllServices();
            }));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

vii. WaterPollutionTrackerClient.java

```
package ds.waterpollutiontracker;

import
```

```
ds.waterpollutiontracker.WaterPollutionTrackerGrpc.WaterPollutionTrackerBlockingStub;
import ds.waterpollutiontracker.WaterPollutionTrackerGrpc.WaterPollutionTrackerStub;
import io.grpc.Deadline;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.Status;
import io.grpc.StatusRuntimeException;
import io.grpc.stub.StreamObserver;

import javax.jmdns.JmDNS;
import javax.jmdns.ServiceEvent;
import javax.jmdns.ServiceInfo;
import javax.jmdns.ServiceListener;

import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.InetAddress;
import java.util.Iterator;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

public class WaterPollutionTrackerClient {
    private static CountDownLatch serviceResolvedLatch = new CountDownLatch(1);
    static String serviceType = "_http._tcp.local.";
    static String resolvedIP;
```

```
static int port;


public static void main(String[] args) {
    discoverServiceWithJmDNS();


    try {
        serviceResolvedLatch.await();
    } catch (InterruptedException e) {
        System.err.println("Service resolution interrupted: " + e.getMessage());
        return;
    }


    // Check if the service has been resolved
    if (resolvedIP != null && port > 0) {
        // Run the GUI
        runClientGui(resolvedIP, port);
    } else {
        System.out.println("Could not resolve the service.");
    }
}


public static void runClientGui(String ip, int port) {
    JFrame frame = new JFrame("Water Pollution Tracker");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(800, 600);
```

```
        JPanel panel = new JPanel();
        panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
        frame.add(panel);

        // Get Water Pollution History UI elements
        JLabel titleLabelHistory = new JLabel("Method 1: Water Pollution History");
        panel.add(titleLabelHistory);

        JLabel historyLocationLabel = new JLabel("Location:");
        panel.add(historyLocationLabel);

        JTextField historyLocationField = new JTextField(20);
        panel.add(historyLocationField);

        JLabel startTimeLabel = new JLabel("Start Time:");
        panel.add(startTimeLabel);

        SpinnerDateModel startTimeModel = new SpinnerDateModel();
        JSpinner startTimeSpinner = new JSpinner(startTimeModel);
        JSpinner.DateEditor startTimeEditor = new JSpinner.DateEditor(startTimeSpinner,
"yyyy-MM-dd HH:mm:ss");
        startTimeSpinner.setEditor(startTimeEditor);
        panel.add(startTimeSpinner);

        JLabel endTimeLabel = new JLabel("End Time:");
        panel.add(endTimeLabel);
```

```
        SpinnerDateModel endTimeModel = new SpinnerDateModel();
        JSpinner endTimeSpinner = new JSpinner(endTimeModel);
        JSpinner.DateEditor endTimeEditor = new JSpinner.DateEditor(endTimeSpinner,
"yyyy-MM-dd HH:mm:ss");
        endTimeSpinner.setEditor(endTimeEditor);
        panel.add(endTimeSpinner);

        JButton getHistoryButton = new JButton("Get Water Pollution History");
        panel.add(getHistoryButton);

        JTextArea historyOutputArea = new JTextArea(5, 40);
        historyOutputArea.setEditable(false);
        JScrollPane historyScrollPane = new JScrollPane(historyOutputArea);
        panel.add(historyScrollPane);

        // Get Water Pollution Alerts UI elements
        JLabel titleLabelAlerts = new JLabel("Method 2: Get Water Pollution Alerts");
        panel.add(titleLabelAlerts);

        JLabel alertsLocationLabel = new JLabel("Location:");
        panel.add(alertsLocationLabel);

        JTextField alertsLocationField = new JTextField(20);
        panel.add(alertsLocationField);
```

```
JLabel thresholdLabel = new JLabel("Threshold:");
panel.add(thresholdLabel);

JTextField thresholdField = new JTextField(20);
panel.add(thresholdField);

JButton getAlertsButton = new JButton("Get Water Pollution Alerts");
panel.add(getAlertsButton);

JTextArea alertsOutputArea = new JTextArea(5, 40);
alertsOutputArea.setEditable(false);
JScrollPane alertsScrollPane = new JScrollPane(alertsOutputArea);
panel.add(alertsScrollPane);

// Monitor Water Pollution UI elements
JLabel titleLabelMonitor = new JLabel("Method 3: Monitor Water Pollution");
panel.add(titleLabelMonitor);

JLabel monitorLocationLabel1 = new JLabel("Location 1:");
panel.add(monitorLocationLabel1);

JTextField monitorLocationField1 = new JTextField(20);
panel.add(monitorLocationField1);

JLabel monitorLocationLabel2 = new JLabel("Location 2:");
panel.add(monitorLocationLabel2);
```

```
        JTextField monitorLocationField2 = new JTextField(20);
        panel.add(monitorLocationField2);

        JButton monitorButton = new JButton("Monitor Water Pollution");
        panel.add(monitorButton);

        JTextArea monitorOutputArea = new JTextArea(5, 40);
        monitorOutputArea.setEditable(false);
        JScrollPane monitorScrollPane = new JScrollPane(monitorOutputArea);
        panel.add(monitorScrollPane);

        // Get Water Pollution History ActionListener
        getHistoryButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String location = historyLocationField.getText();
                long startTime = startTimeModel.getDate().getTime();
                long endTime = endTimeModel.getDate().getTime();

                ManagedChannel channel =
ManagedChannelBuilder.forAddress(resolvedIP, port).usePlaintext().build();
                WaterPollutionTrackerBlockingStub blockingStub =
WaterPollutionTrackerGrpc.newBlockingStub(channel);

                WaterPollutionHistoryRequest historyRequest =
```

```
WaterPollutionHistoryRequest.newBuilder()
                        .setStartTime(startTime)
                        .setEndTime(endTime)
                        .setLocation(location).build();

                // Set the deadline to 5 seconds
                long deadlineInSeconds = 5;
                Deadline deadline = Deadline.after(deadlineInSeconds,
TimeUnit.SECONDS);

                try {
                    Iterator<WaterPollutionLevel> historyResponseIterator =
blockingStub
                            .withDeadline(deadline)
                            .getWaterPollutionHistory(historyRequest);

                    while (historyResponseIterator.hasNext()) {
                        WaterPollutionLevel historyResponse =
historyResponseIterator.next();
                        historyOutputArea.append("History message sent by the server:
" + historyResponse + "₩n");
                    }
                } catch (StatusRuntimeException ex) {
                    historyOutputArea.append("Error encountered in
WaterPollutionTrackerClient: " + ex.getMessage() + "₩n");
                } finally {
                    channel.shutdown();
```

```
                }
            }
        });

        // Get Water Pollution Alerts ActionListener
        getAlertsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String location = alertsLocationField.getText();
                int threshold = Integer.parseInt(thresholdField.getText());

                ManagedChannel channel = ManagedChannelBuilder.forAddress(resolvedIP, port).usePlaintext().build();
                WaterPollutionTrackerBlockingStub blockingStub = WaterPollutionTrackerGrpc.newBlockingStub(channel);

                WaterPollutionAlertsRequest alertRequest = WaterPollutionAlertsRequest.newBuilder()
                        .setLocation(location).setThreshold(threshold).build();

                try {
                    // Set a deadline of 5 seconds
                    WaterPollutionAlertsResponse alertsResponse = blockingStub
                            .withDeadlineAfter(5, TimeUnit.SECONDS)
                            .getWaterPollutionAlerts(alertRequest);
                    alertsOutputArea.append("Alerts sent by the server: " + alertsResponse.getAlertsList() + "₩n");
```

```
                } catch (StatusRuntimeException ex) {
                    if (ex.getStatus() == Status.DEADLINE_EXCEEDED) {
                        alertsOutputArea.append("Error: Deadline exceeded while waiting for server response.₩n");
                    } else {
                        alertsOutputArea.append("Error encountered in WaterPollutionTrackerClient: " + ex.getMessage() + "₩n");
                    }
                } finally {
                    channel.shutdown();
                }
            }
        });

        // Monitor Water Pollution ActionListener
        monitorButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String location1 = monitorLocationField1.getText();
                String location2 = monitorLocationField2.getText();

                ManagedChannel channel = ManagedChannelBuilder.forAddress(resolvedIP, port)
                        .usePlaintext().build();
                WaterPollutionTrackerStub asyncStub = WaterPollutionTrackerGrpc.newStub(channel);
```

```
                    CountDownLatch finishedLatch = new CountDownLatch(1);
                    StreamObserver<WaterPollutionMonitorRequest>
monitorRequestObserver1 = asyncStub
                            .withDeadlineAfter(10, TimeUnit.SECONDS) // set deadline to 10
seconds
                            .monitorWaterPollution(new
StreamObserver<WaterPollutionLevel>() {
                                @Override
                                public void onNext(WaterPollutionLevel value) {
                                    monitorOutputArea.append("Received real-time data: "
+ value + "₩n");
                                }

                                @Override
                                public void onError(Throwable t) {
                                    monitorOutputArea.append("Error encountered in
WaterPollutionTrackerClient: " + t.getMessage() + "₩n");
                                    finishedLatch.countDown();
                                }

                                @Override
                                public void onCompleted() {
                                    monitorOutputArea.append("Real-time data receiving
completed.₩n");
                                    finishedLatch.countDown();
                                }
                            });
```

```
                    StreamObserver<WaterPollutionMonitorRequest>
monitorRequestObserver2 = asyncStub
                            .withDeadlineAfter(10, TimeUnit.SECONDS) // set deadline to 10
seconds
                            .monitorWaterPollution(new
StreamObserver<WaterPollutionLevel>() {
                                @Override
                                public void onNext(WaterPollutionLevel value) {
                                    monitorOutputArea.append("Received real-time data: "
+ value + "₩n");
                                }

                                @Override
                                public void onError(Throwable t) {
                                    monitorOutputArea.append("Error encountered in
WaterPollutionTrackerClient: " + t.getMessage() + "₩n");
                                    finishedLatch.countDown();
                                }

                                @Override
                                public void onCompleted() {
                                    monitorOutputArea.append("Real-time data receiving
completed.₩n");
                                    finishedLatch.countDown();
                                }
                            });
```

```
                try {
                    if (!location1.isEmpty()) {
                        WaterPollutionMonitorRequest monitorRequest1 =
WaterPollutionMonitorRequest.newBuilder()
                                .setLocation(location1).build();
                        monitorRequestObserver1.onNext(monitorRequest1);
                    }

                    if (!location2.isEmpty()) {
                        WaterPollutionMonitorRequest monitorRequest2 =
WaterPollutionMonitorRequest.newBuilder()
                                .setLocation(location2)
                                .build();
                        monitorRequestObserver2.onNext(monitorRequest2);
                    }

                    // Sleep for demonstration purposes, replace with appropriate logic
                    Thread.sleep(5000);

                    monitorRequestObserver1.onCompleted();
                    monitorRequestObserver2.onCompleted();

                    // Wait for the server to complete sending data
                    if (!finishedLatch.await(1, TimeUnit.MINUTES)) {
                        monitorOutputArea.append("monitorWaterPollution can not
```

```
finish within 1 minute₩n");
                        }
                    } catch (RuntimeException e1) {
                        monitorRequestObserver1.onError(e1);
                        monitorRequestObserver2.onError(e1);
                        throw e1;
                    } catch (InterruptedException e2) {
                        e2.printStackTrace();
                    } finally {
                        channel.shutdown();
                    }
                }
            });

            frame.setVisible(true);
        }

        // JmDNS service discovery method
            public static void discoverServiceWithJmDNS() {
                try {
                        JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

                        jmdns.addServiceListener(serviceType, new ServiceDiscoverer());

                        Thread.sleep(20000);
                } catch (Exception e) {
```

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

        private static class ServiceDiscoverer 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());
            }

            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);

            // Signal that the service has been resolved
            serviceResolvedLatch.countDown();
          }
        }
    }
```

viii. AirPollutionTrackerServer.java

```
package ds.airpollutiontracker;

import java.io.IOException;
import java.net.InetAddress;
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

import javax.jmdns.JmDNS;
import javax.jmdns.ServiceInfo;

import ds.airpollutiontracker.AirPollutionTrackerGrpc.AirPollutionTrackerImplBase;
import io.grpc.Server;
import io.grpc.ServerBuilder;
import io.grpc.stub.StreamObserver;

public class AirPollutionTrackerServer extends AirPollutionTrackerImplBase {
        static int port = 50085;

    public static void main(String[] args) throws InterruptedException, IOException {
        AirPollutionTrackerServer aTracker = new AirPollutionTrackerServer();
```

```
        Server server;
        try {
            server = ServerBuilder.forPort(port)
                    .addService(aTracker)
                    .build()
                    .start();

            System.out.println("AirPollutionTracker started, listening on " + port);

            registerWithJmDNS();
            server.awaitTermination();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void getAirPollutionHistory(AirPollutionHistoryRequest request,
                                       StreamObserver<AirPollutionLevel>
responseObserver) {
        Instant startTime = Instant.ofEpochMilli(request.getStartTime());
        Instant endTime = Instant.ofEpochMilli(request.getEndTime());
        String location = request.getLocation();
```

```
        // Call the generateSampleData() method with startTime and endTime
        List<AirPollutionLevel> pollutionLevels = generateSampleData(startTime,
endTime);

        List<AirPollutionLevel> filteredData = pollutionLevels.stream()
                .filter(level -> level.getLocation().equals(location))
                .filter(level -> Instant.parse(level.getTimestamp()).isAfter(startTime))
                .filter(level -> Instant.parse(level.getTimestamp()).isBefore(endTime))
                .collect(Collectors.toList());

        for (AirPollutionLevel level : filteredData) {
            responseObserver.onNext(level);
        }
        responseObserver.onCompleted();
    }

    @Override
    public void getAirPollutionAlerts(AirPollutionAlertsRequest request,
                                      StreamObserver<AirPollutionAlertsResponse>
responseObserver) {
        // TODO: Replace this with actual data retrieval from the database
        List<AirPollutionAlert> alerts = generateSampleAlerts();

        int threshold = request.getThreshold();
        String location = request.getLocation();
```

```
        List<AirPollutionAlert> filteredAlerts = alerts.stream()
                .filter(alert -> alert.getLocation().equals(location))
                .filter(alert -> alert.getPollutionLevel() > threshold)
                .collect(Collectors.toList());

        AirPollutionAlertsResponse response = AirPollutionAlertsResponse.newBuilder()
                .addAllAlerts(filteredAlerts)
                .build();

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

    @Override
    public StreamObserver<AirPollutionMonitorRequest> monitorAirPollution(
            StreamObserver<AirPollutionLevel> responseObserver) {
        // TODO: Replace this with actual real-time data monitoring implementation
        return new StreamObserver<AirPollutionMonitorRequest>() {
            @Override
            public void onNext(AirPollutionMonitorRequest request) {
                String location = request.getLocation();

                // Generate sample data for demonstration purposes
                List<AirPollutionLevel> pollutionLevels = generateSampleDataMonitor();

                for (AirPollutionLevel level : pollutionLevels) {
```

```
                    if (level.getLocation().equals(location)) {
                        responseObserver.onNext(level);
                    }
                }
            }

            @Override
            public void onError(Throwable t) {
                System.out.println("Error encountered in AirPollutionTrackerServer: " +
t.getMessage());
            }

            @Override
            public void onCompleted() {
                responseObserver.onCompleted();
            }
        };
    }

    private List<AirPollutionLevel> generateSampleData(Instant startTime, Instant endTime)
{
        // This method generates sample data for demonstration purposes
        List<AirPollutionLevel> pollutionLevels = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            Instant timestamp = startTime.plusSeconds(random.nextInt((int)
```

```
(endTime.getEpochSecond() - startTime.getEpochSecond())));
                AirPollutionLevel level = AirPollutionLevel.newBuilder()
                        .setLocation("Dublin " + i)
                        .setPollutionType("Polluion type " + i)
                        .setPollutionLevel(random.nextFloat() * 200)
                        .setTimestamp(timestamp.toString())
                        .build();
                pollutionLevels.add(level);
            }
            for (int i = 0; i < 10; i++) {
                Instant timestamp = startTime.plusSeconds(random.nextInt((int)
(endTime.getEpochSecond() - startTime.getEpochSecond())));
                AirPollutionLevel level = AirPollutionLevel.newBuilder()
                        .setLocation("Dublin " + i)
                        .setPollutionType("Polluion type " + i)
                        .setPollutionLevel(random.nextFloat() * 200)
                        .setTimestamp(timestamp.toString())
                        .build();
                pollutionLevels.add(level);
            }
            return pollutionLevels;
        }

        private List<AirPollutionLevel> generateSampleDataMonitor() {
            // This method generates sample data for demonstration purposes
            List<AirPollutionLevel> pollutionLevels = new ArrayList<>();
```

```
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            AirPollutionLevel level = AirPollutionLevel.newBuilder()
                    .setLocation("Dublin " + i)
                    .setPollutionType("Polluion type " + i)
                    .setPollutionLevel(random.nextFloat() * 200)
                    .setTimestamp(Instant.now().minusSeconds(random.nextInt(3600)).to
String())
                    .build();
            pollutionLevels.add(level);
        }
        for (int i = 0; i < 10; i++) {
            AirPollutionLevel level = AirPollutionLevel.newBuilder()
                    .setLocation("Dublin " + i)
                    .setPollutionType("Polluion type " + i)
                    .setPollutionLevel(random.nextFloat() * 200)
                    .setTimestamp(Instant.now().minusSeconds(random.nextInt(3600)).to
String())
                    .build();
            pollutionLevels.add(level);
        }
        return pollutionLevels;
    }

    private List<AirPollutionAlert> generateSampleAlerts() {
        // This method generates sample alerts for demonstration purposes
```

```
        List<AirPollutionAlert> alerts = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < 5; i++) {
            AirPollutionAlert alert = AirPollutionAlert.newBuilder()
                    .setLocation("Dunlin " + i)
                    .setPollutionType("Pollution type " + i)
                    .setPollutionLevel(random.nextFloat() * 300)
                    .setTimestamp(Instant.now().minusSeconds(random.nextInt(3600)).to
String())
                    .build();
            alerts.add(alert);
        }
        return alerts;
    }

    // JmDNS registration method
    public static void registerWithJmDNS() {
        try {
            // Create a JmDNS instance
            JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

            // Register a service
            ServiceInfo serviceInfo = ServiceInfo.create("_http._tcp.local.", "air-pollution-
tracker", port, "AirPollutionTracker service");
            jmdns.registerService(serviceInfo);
```

```
                // Wait a bit
                Thread.sleep(20000);

                // Unregister all services
                // jmdns.unregisterAllServices();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
```

ix. AirPollutionTrackerClient.java

```
package ds.airpollutiontracker;

import
ds.airpollutiontracker.AirPollutionTrackerGrpc.AirPollutionTrackerBlockingS
tub;
import
ds.airpollutiontracker.AirPollutionTrackerGrpc.AirPollutionTrackerStub;
import io.grpc.Deadline;
import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.StatusRuntimeException;
import io.grpc.stub.StreamObserver;
import io.grpc.Status;

import javax.jmdns.JmDNS;
import javax.jmdns.ServiceEvent;
import javax.jmdns.ServiceInfo;
import javax.jmdns.ServiceListener;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.InetAddress;
import java.util.Iterator;
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

public class AirPollutionTrackerClient {
       private static CountDownLatch serviceResolvedLatch = new
CountDownLatch(1);
       static String serviceType = "_http._tcp.local.";
       static String resolvedIP;
       static int port;
```

```
	public static void main(String[] args) {
		discoverServiceWithJmDNS();

		try {
			serviceResolvedLatch.await();
		} catch (InterruptedException e) {
			System.err.println("Service resolution interrupted: " +
e.getMessage());
			return;
		}

		// Check if the service has been resolved
		if (resolvedIP != null && port > 0) {
			// Run the GUI
			runClientGui(resolvedIP, port);
		} else {
			System.out.println("Could not resolve the service.");
		}
	}

	public static void runClientGui(String ip, int port) {
		JFrame frame = new JFrame("Air Pollution Tracker");
		frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		frame.setSize(800, 600);

		JPanel panel = new JPanel();
		panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
		frame.add(panel);

		// Method 2: Get Air Pollution History
		JLabel titleLabelHistory = new JLabel("Method 1: Get Air
Pollution History");
		panel.add(titleLabelHistory);

		JLabel historyLocationLabel = new JLabel("Location:");
		panel.add(historyLocationLabel);

		JTextField historyLocationField = new JTextField(20);
		panel.add(historyLocationField);

		JLabel startTimeLabel = new JLabel("Start Time:");
		panel.add(startTimeLabel);

		SpinnerDateModel startTimeModel = new SpinnerDateModel();
		JSpinner startTimeSpinner = new JSpinner(startTimeModel);
		JSpinner.DateEditor startTimeEditor = new
JSpinner.DateEditor(startTimeSpinner, "yyyy-MM-dd HH:mm:ss");
		startTimeSpinner.setEditor(startTimeEditor);
		panel.add(startTimeSpinner);

		JLabel endTimeLabel = new JLabel("End Time:");
		panel.add(endTimeLabel);

		SpinnerDateModel endTimeModel = new SpinnerDateModel();
		JSpinner endTimeSpinner = new JSpinner(endTimeModel);
		JSpinner.DateEditor endTimeEditor = new
JSpinner.DateEditor(endTimeSpinner, "yyyy-MM-dd HH:mm:ss");
		endTimeSpinner.setEditor(endTimeEditor);
		panel.add(endTimeSpinner);
```

```
                JButton getHistoryButton = new JButton("Get Air Pollution
History");
                panel.add(getHistoryButton);

                JTextArea historyOutputArea = new JTextArea(5, 40);
                historyOutputArea.setEditable(false);
                JScrollPane historyScrollPane = new
JScrollPane(historyOutputArea);
                panel.add(historyScrollPane);

                // Get Air Pollution Alerts UI elements
                JLabel titleLabelAlerts = new JLabel("Method 2: Get Air
Pollution Alerts");
                panel.add(titleLabelAlerts);

                JLabel alertsLocationLabel = new JLabel("Location:");
                panel.add(alertsLocationLabel);

                JTextField alertsLocationField = new JTextField(20);
                panel.add(alertsLocationField);

                JLabel thresholdLabel = new JLabel("Threshold:");
                panel.add(thresholdLabel);

                JTextField thresholdField = new JTextField(20);
                panel.add(thresholdField);

                JButton getAlertsButton = new JButton("Get Air Pollution
Alerts");
                panel.add(getAlertsButton);

                JTextArea alertsOutputArea = new JTextArea(5, 40);
                alertsOutputArea.setEditable(false);
                JScrollPane alertsScrollPane = new
JScrollPane(alertsOutputArea);
                panel.add(alertsScrollPane);

                // Monitor Air Pollution UI elements
                JLabel titleLabelMonitor = new JLabel("Method 3: Monitor Air
Pollution");
                panel.add(titleLabelMonitor);

                JLabel monitorLocationLabel1 = new JLabel("Location 1 :");
                panel.add(monitorLocationLabel1);

                JTextField monitorLocationField1 = new JTextField(20);
                panel.add(monitorLocationField1);

                JLabel monitorLocationLabel2 = new JLabel("Location 2 :");
                panel.add(monitorLocationLabel2);

                JTextField monitorLocationField2 = new JTextField(20);
                panel.add(monitorLocationField2);

                JButton monitorButton = new JButton("Monitor Air Pollution");
                panel.add(monitorButton);

                JTextArea monitorOutputArea = new JTextArea(5, 40);
                monitorOutputArea.setEditable(false);
```

```
            JScrollPane monitorScrollPane = new
JScrollPane(monitorOutputArea);
            panel.add(monitorScrollPane);

            // Get Air Pollution History ActionListener
            getHistoryButton.addActionListener(new ActionListener() {
                  @Override
                  public void actionPerformed(ActionEvent e) {
                        String location =
historyLocationField.getText();
                        long startTime =
startTimeModel.getDate().getTime();
                        long endTime = endTimeModel.getDate().getTime();
                        ManagedChannel channel =
ManagedChannelBuilder.forAddress(resolvedIP, port).usePlaintext().build();
                        AirPollutionTrackerBlockingStub blockingStub =
AirPollutionTrackerGrpc.newBlockingStub(channel);

                        AirPollutionHistoryRequest historyRequest =
AirPollutionHistoryRequest.newBuilder()

      .setStartTime(startTime).setEndTime(endTime).setLocation(location).b
uild();

                        // Set the deadline to 5 seconds
                        long deadlineInSeconds = 5;
                        Deadline deadline =
Deadline.after(deadlineInSeconds, TimeUnit.SECONDS);

                        try {
                              Iterator<AirPollutionLevel>
historyResponseIterator = blockingStub.withDeadline(deadline)

      .getAirPollutionHistory(historyRequest);
                              while (historyResponseIterator.hasNext())
{
                                    AirPollutionLevel historyResponse =
historyResponseIterator.next();
                                    historyOutputArea.append("History
message sent by the server: " + historyResponse + "\n");
                              }
                        } catch (StatusRuntimeException ex) {
                              if (ex.getStatus().getCode() ==
Status.Code.DEADLINE_EXCEEDED) {
                                    historyOutputArea.append("Deadline
exceeded. Could not get air pollution history.\n");
                              } else {
                                    historyOutputArea
                                                .append("Error
encountered in AirPollutionTrackerClient: " + ex.getMessage() + "\n");
                              }
                        } finally {
                              channel.shutdown();
                        }
                  }
            });

            // Get Air Pollution Alerts ActionListener
            getAlertsButton.addActionListener(new ActionListener() {
                  @Override
```

```
                    public void actionPerformed(ActionEvent e) {
                        String location = alertsLocationField.getText();
                        int threshold =
Integer.parseInt(thresholdField.getText());

                        ManagedChannel channel =
ManagedChannelBuilder.forAddress(resolvedIP, port).usePlaintext().build();
                        AirPollutionTrackerBlockingStub blockingStub =
AirPollutionTrackerGrpc.newBlockingStub(channel);

                        AirPollutionAlertsRequest alertRequest =
AirPollutionAlertsRequest.newBuilder().setLocation(location)
                                .setThreshold(threshold).build();

                        // Set the deadline to 5 seconds
                        long deadlineInSeconds = 5;
                        Deadline deadline =
Deadline.after(deadlineInSeconds, TimeUnit.SECONDS);

                        try {
                            AirPollutionAlertsResponse alertsResponse
= blockingStub.withDeadline(deadline)

       .getAirPollutionAlerts(alertRequest);
                            alertsOutputArea.append("Alerts sent by
the server: " + alertsResponse.getAlertsList() + "\n");
                        } catch (StatusRuntimeException ex) {
                            if (ex.getStatus().getCode() ==
Status.Code.DEADLINE_EXCEEDED) {
                                alertsOutputArea.append("Deadline
exceeded. Could not get air pollution alerts.\n");
                            } else {
                                alertsOutputArea
                                        .append("Error
encountered in AirPollutionTrackerClient: " + ex.getMessage() + "\n");
                            }
                        } finally {
                            channel.shutdown();
                        }
                    }
                });

                // Monitor Air Pollution ActionListener
                monitorButton.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent e) {
                        String location1 = monitorLocationField1.getText();
                        String location2 = monitorLocationField2.getText();

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

                        AirPollutionTrackerStub asyncStub =
AirPollutionTrackerGrpc.newStub(channel);

                        CountDownLatch finishedLatch = new CountDownLatch(2);
```

```
                    StreamObserver<AirPollutionMonitorRequest>
monitorRequestObserver1 = asyncStub.monitorAirPollution(
                            new StreamObserver<AirPollutionLevel>() {
                                @Override
                                public void onNext(AirPollutionLevel value) {
                                    monitorOutputArea.append("Received real-
time data from location 1: " + value + "\n");
                                }

                                @Override
                                public void onError(Throwable t) {
                                    monitorOutputArea.append("Error
encountered in AirPollutionTrackerClient: " + t.getMessage() + "\n");
                                    finishedLatch.countDown();
                                }

                                @Override
                                public void onCompleted() {
                                    monitorOutputArea.append("Real-time data
receiving completed for location 1.\n");
                                    finishedLatch.countDown();
                                }
                            });

                    StreamObserver<AirPollutionMonitorRequest>
monitorRequestObserver2 = asyncStub.monitorAirPollution(
                            new StreamObserver<AirPollutionLevel>() {
                                @Override
                                public void onNext(AirPollutionLevel value) {
                                    monitorOutputArea.append("Received real-
time data from location 2: " + value + "\n");
                                }

                                @Override
                                public void onError(Throwable t) {
                                    monitorOutputArea.append("Error
encountered in AirPollutionTrackerClient: " + t.getMessage() + "\n");
                                    finishedLatch.countDown();
                                }

                                @Override
                                public void onCompleted() {
                                    monitorOutputArea.append("Real-time data
receiving completed for location 2.\n");
                                    finishedLatch.countDown();
                                }
                            });

                    try {
                        if (!location1.isEmpty()) {
                            AirPollutionMonitorRequest monitorRequest1 =
AirPollutionMonitorRequest.newBuilder()
                                    .setLocation(location1)
                                    .build();
                            monitorRequestObserver1.onNext(monitorRequest1);
                        }

                        if (!location2.isEmpty()) {
                            AirPollutionMonitorRequest monitorRequest2 =
AirPollutionMonitorRequest.newBuilder()
```

```
                                .setLocation(location2)
                                .build();
                        monitorRequestObserver2.onNext(monitorRequest2);
                    }

                    // Add a deadline of 10 seconds to the
monitorAirPollution calls
                    asyncStub = asyncStub.withDeadlineAfter(10,
TimeUnit.SECONDS);

                    // Sleep for demonstration purposes, replace with
appropriate logic
                    Thread.sleep(5000);

                    monitorRequestObserver1.onCompleted();
                    monitorRequestObserver2.onCompleted();

                    // Wait for the server to complete sending data
                    if (!finishedLatch.await(1, TimeUnit.MINUTES)) {
                        monitorOutputArea.append("monitorAirPollution
can not finish within 1 minute\n");
                    }
                } catch (RuntimeException e1) {
                    monitorRequestObserver1.onError(e1);
                    monitorRequestObserver2.onError(e1);
                    throw e1;
                } catch (InterruptedException e2) {
                    e2.printStackTrace();
                } finally {
                    channel.shutdown();
                }
            }
        });

        frame.setVisible(true);
    }

    // JmDNS service discovery method
    public static void discoverServiceWithJmDNS() {
        try {
            JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

            jmdns.addServiceListener(serviceType, new
ServiceDiscoverer());

            Thread.sleep(20000);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    private static class ServiceDiscoverer 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());
```

```
            }

            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);

                // Signal that the service has been resolved
                serviceResolvedLatch.countDown();
            }
        }
}
```

x. DataVizualizerServer.java

```
package ds.datavisualizer;


import java.io.IOException;

import java.net.InetAddress;

import java.time.Instant;

import java.util.ArrayList;

import java.util.List;

import java.util.Random;


import javax.jmdns.JmDNS;

import javax.jmdns.ServiceInfo;


import ds.datavisualizer.DataVisualizationGrpc.DataVisualizationImplBase;

import io.grpc.Server;

import io.grpc.ServerBuilder;
```

```
import io.grpc.stub.StreamObserver;

public class DataVisualizerServer extends DataVisualizationImplBase {
	static int port = 50088;

	public static void main(String[] args) throws InterruptedException, IOException {
		DataVisualizerServer dataVisualizer = new DataVisualizerServer();

		Server server;
		try {
			server = ServerBuilder.forPort(port).addService(dataVisualizer).build().start();

			System.out.println("DataVisualizer started, listening on " + port);

			registerWithJmDNS();
			server.awaitTermination();
		} catch (IOException e) {
			e.printStackTrace();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
	}

	@Override
	public void getPollutionStatistics(PollutionStatisticsRequest request,
```

```
                StreamObserver<PollutionStatistics> responseObserver) {
        List<PollutionLevel> sampleData = generateSamplePollutionData();
        String location = request.getLocation();
        float totalPollutionLevel = 0;
        int count = 0;
        float highestPollutionLevel = Float.MIN_VALUE;

        // Calculate the average and highest pollution level for the requested location
        for (PollutionLevel pollutionLevel : sampleData) {
            if (pollutionLevel.getLocation().equals(location)) {
                float currentPollutionLevel = pollutionLevel.getPollutionLevel();
                totalPollutionLevel += currentPollutionLevel;
                count++;

                if (currentPollutionLevel > highestPollutionLevel) {
                    highestPollutionLevel = currentPollutionLevel;
                }
            }
        }

        PollutionStatistics.Builder statisticsBuilder = PollutionStatistics.newBuilder();

        if (count > 0) {
```

```
                    float averagePollutionLevel = totalPollutionLevel / count;
                    statisticsBuilder.setAveragePollutionLevel(averagePollutionLevel)
        .setHighestPollutionLevel(highestPollutionLevel);
                }

                responseObserver.onNext(statisticsBuilder.build());
                responseObserver.onCompleted();
        }

        @Override
        public StreamObserver<LocationFilterRequest>
filterDataByLocation(StreamObserver<PollutionLevel> responseObserver) {
                return new StreamObserver<LocationFilterRequest>() {
                        @Override
                        public void onNext(LocationFilterRequest request) {
                                List<PollutionLevel> sampleData =
generateSamplePollutionData();

                                // Filter the data based on the location in the request
                                for (PollutionLevel pollutionLevel : sampleData) {
                                        if
(pollutionLevel.getLocation().equals(request.getLocation())) {
        responseObserver.onNext(pollutionLevel);
                                        }
                                }
```

```
                }

                @Override
                public void onError(Throwable t) {
                        t.printStackTrace();
                }

                @Override
                public void onCompleted() {
                        responseObserver.onCompleted();
                }
        };
    }

    @Override
    public StreamObserver<FavoriteLocationRequest> setFavoriteLocation(
                StreamObserver<FavoriteLocationResponse> responseObserver)
{
        return new StreamObserver<FavoriteLocationRequest>() {
                @Override
                public void onNext(FavoriteLocationRequest request) {
                        // Save the favorite location
                        // For simplicity, we'll just print the location and assume
it's saved
                        System.out.println("Favorite location saved: " +
request.getLocation());
```

```
                FavoriteLocationResponse response = FavoriteLocationResponse.newBuilder()
                        .setStatus("Location saved successfully").build();
                responseObserver.onNext(response);
            }

            @Override
            public void onError(Throwable t) {
                t.printStackTrace();
            }

            @Override
            public void onCompleted() {
                responseObserver.onCompleted();
            }
        };
    }

    private List<PollutionLevel> generateSamplePollutionData() {
        // This method generates sample data for demonstration purposes
        List<PollutionLevel> pollutionData = new ArrayList<>();
        Random random = new Random();
        for (int i = 1; i <= 10; i++) {
            PollutionLevel pollutionLevel = PollutionLevel.newBuilder()
```

```
                                        .setLocation("Dublin " + i)
                                        .setPollutionType("Pollution type" +
i).setPollutionLevel(random.nextFloat() * 100)

            .setTimestamp(Instant.now().getEpochSecond()).build();
                        pollutionData.add(pollutionLevel);
                }
                return pollutionData;
            }

            // JmDNS registration method
            public static void registerWithJmDNS() {
                try {
                        // Create a JmDNS instance
                        JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

                        // Register a service
                        ServiceInfo serviceInfo = ServiceInfo.create("_http._tcp.local.",
"data-visualizer", port,
                                        "DataVisualizer service");
                        jmdns.registerService(serviceInfo);

                        // Wait a bit
                        Thread.sleep(20000);

                        // Unregister all services
```

```
                    // jmdns.unregisterAllServices();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }
```

xi. DataVizualizerClient.java

```
package ds.datavisualizer;

import io.grpc.ManagedChannel;
import io.grpc.ManagedChannelBuilder;
import io.grpc.Status;
import io.grpc.StatusRuntimeException;
import io.grpc.stub.StreamObserver;

import javax.jmdns.JmDNS;
import javax.jmdns.ServiceEvent;
import javax.jmdns.ServiceInfo;
import javax.jmdns.ServiceListener;
import javax.swing.*;

import ds.datavisualizer.DataVisualizationGrpc.DataVisualizationStub;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.net.InetAddress;
```

```
import java.util.concurrent.CountDownLatch;
import java.util.concurrent.TimeUnit;

public class DataVisualizerClient {
    private static CountDownLatch serviceResolvedLatch = new CountDownLatch(1);
    static String serviceType = "_http._tcp.local.";
    static String resolvedIP;
    static int port;

    public static void main(String[] args) {
        discoverServiceWithJmDNS();

        try {
            serviceResolvedLatch.await();
        } catch (InterruptedException e) {
            System.err.println("Service resolution interrupted: " + e.getMessage());
            return;
        }

        // Check if the service has been resolved
        if (resolvedIP != null && port > 0) {
            // Run the GUI
            runClientGui();
        } else {
            System.out.println("Could not resolve the service.");
        }
```

```
}

public static void runClientGui() {
    JFrame frame = new JFrame("Data Visualizer");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(600, 400);

    JPanel panel = new JPanel();
    panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
    frame.add(panel);

    // Get Pollution Statistics UI elements
    JLabel titleLabelStatistics = new JLabel("Method 1: Get Pollution Statistics");
    panel.add(titleLabelStatistics);

    JLabel locationLabel = new JLabel("Location:");
    panel.add(locationLabel);

    JTextField locationField = new JTextField(20);
    panel.add(locationField);

    JButton getStatisticsButton = new JButton("Get Pollution Statistics");
    panel.add(getStatisticsButton);

    JTextArea statisticsOutputArea = new JTextArea(5, 40);
    statisticsOutputArea.setEditable(false);
```

```
JScrollPane statisticsScrollPane = new JScrollPane(statisticsOutputArea);
panel.add(statisticsScrollPane);

// Filter Data by Location UI elements
JLabel titleLabelFilter = new JLabel("Method 2: Filter Data by Location");
panel.add(titleLabelFilter);

JLabel filterLocationLabel1 = new JLabel("Location1 1 :");
panel.add(filterLocationLabel1);

JTextField filterLocationField1 = new JTextField(20);
panel.add(filterLocationField1);

JLabel filterLocationLabel2 = new JLabel("Location1 2 :");
panel.add(filterLocationLabel2);

JTextField filterLocationField2 = new JTextField(20);
panel.add(filterLocationField2);

JLabel filterLocationLabel3 = new JLabel("Location1 3 :");
panel.add(filterLocationLabel3);

JTextField filterLocationField3 = new JTextField(20);
panel.add(filterLocationField3);

JButton filterDataButton = new JButton("Filter Data by Location");
```

```
panel.add(filterDataButton);

JTextArea filterOutputArea = new JTextArea(5, 40);
filterOutputArea.setEditable(false);
JScrollPane filterScrollPane = new JScrollPane(filterOutputArea);
panel.add(filterScrollPane);

// Set Favorite Location UI elements
JLabel titleLabelFavorite = new JLabel("Method 3: Set Favorite Location");
panel.add(titleLabelFavorite);

JLabel favoriteLocationLabel1 = new JLabel("Location1:");
panel.add(favoriteLocationLabel1);

JTextField favoriteLocationField1 = new JTextField(20);
panel.add(favoriteLocationField1);

JLabel favoriteLocationLabel2 = new JLabel("Location 2 :");
panel.add(favoriteLocationLabel2);

JTextField favoriteLocationField2 = new JTextField(20);
panel.add(favoriteLocationField2);

JLabel favoriteLocationLabel3 = new JLabel("Location3 :");
panel.add(favoriteLocationLabel3);
```

```
        JTextField favoriteLocationField3 = new JTextField(20);
        panel.add(favoriteLocationField3);

        JButton setFavoriteButton = new JButton("Set Favorite Location");
        panel.add(setFavoriteButton);

        JTextArea favoriteOutputArea = new JTextArea(5, 40);
        favoriteOutputArea.setEditable(false);
        JScrollPane favoriteScrollPane = new JScrollPane(favoriteOutputArea);
        panel.add(favoriteScrollPane);

        // Get Pollution Statistics ActionListener
        getStatisticsButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String location = locationField.getText();

                ManagedChannel channel = ManagedChannelBuilder.forAddress(resolvedIP, port).usePlaintext().build();
                DataVisualizationStub asyncStub = DataVisualizationGrpc.newStub(channel);

                CountDownLatch statisticsLatch = new CountDownLatch(1);

                try {
                    asyncStub.withDeadlineAfter(5, TimeUnit.SECONDS)
```

```
                .getPollutionStatistics(PollutionStatisticsRequest.newBuilder
().setLocation(location).build(), new StreamObserver<PollutionStatistics>() {
                    @Override
                    public void onNext(PollutionStatistics value) {
                        statisticsOutputArea.append("Pollution statistics response: "
+ value + "₩n");
                    }

                    @Override
                    public void onError(Throwable t) {
                        statisticsOutputArea.append("Error encountered in
DataVisualizerClient: " + t.getMessage() + "₩n");
                        statisticsLatch.countDown();
                    }

                    @Override
                    public void onCompleted() {
                        statisticsOutputArea.append("Pollution statistics request
completed.₩n");
                        statisticsLatch.countDown();
                    }
                });

                // Wait for the server to complete sending data
                if (!statisticsLatch.await(1, TimeUnit.MINUTES)) {
                    statisticsOutputArea.append("getPollutionStatistics can not finish
within 1 minute₩n");
```

```
                    }
                } catch (StatusRuntimeException ex) {
                    if (ex.getStatus().getCode() ==
Status.DEADLINE_EXCEEDED.getCode()) {
                        statisticsOutputArea.append("Deadline exceeded for
getPollutionStatistics request.₩n");
                    } else {
                        statisticsOutputArea.append("Error encountered in
DataVisualizerClient: " + ex.getMessage() + "₩n");
                    }
                } catch (InterruptedException ex) {
                    ex.printStackTrace();
                } finally {
                    channel.shutdown();
                }
            }
        });

        // Filter Data by Location ActionListener
        filterDataButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String location1 = filterLocationField1.getText();
                String location2 = filterLocationField2.getText();
                String location3 = filterLocationField3.getText();
```

```
                ManagedChannel channel = ManagedChannelBuilder.forAddress(resolvedIP, port).usePlaintext().build();
                DataVisualizationStub asyncStub = DataVisualizationGrpc.newStub(channel);

                CountDownLatch filterLatch = new CountDownLatch(1);
                StreamObserver<LocationFilterRequest> filterRequestObserver1 = asyncStub
                        .withDeadlineAfter(5, TimeUnit.SECONDS)
                        .filterDataByLocation(new StreamObserver<PollutionLevel>() {
                            @Override
                            public void onNext(PollutionLevel value) {
                                filterOutputArea.append("Filtered data response: " + value + "₩n");
                            }

                            @Override
                            public void onError(Throwable t) {
                                filterOutputArea.append("Error encountered in DataVisualizerClient: " + t.getMessage() + "₩n");
                                filterLatch.countDown();
                            }

                            @Override
                            public void onCompleted() {
                                filterOutputArea.append("Filtering completed.₩n");
                                filterLatch.countDown();
```

```
                    }
                });
        StreamObserver<LocationFilterRequest> filterRequestObserver2 =
asyncStub
                    .withDeadlineAfter(5, TimeUnit.SECONDS)
                    .filterDataByLocation(new StreamObserver<PollutionLevel>() {
                        @Override
                        public void onNext(PollutionLevel value) {
                            filterOutputArea.append("Filtered data response: " +
value + "₩n");
                        }

                        @Override
                        public void onError(Throwable t) {
                            filterOutputArea.append("Error encountered in
DataVisualizerClient: " + t.getMessage() + "₩n");
                            filterLatch.countDown();
                        }

                        @Override
                        public void onCompleted() {
                            filterOutputArea.append("Filtering completed.₩n");
                            filterLatch.countDown();
                        }
                    });
        StreamObserver<LocationFilterRequest> filterRequestObserver3 =
asyncStub
```

```
                .withDeadlineAfter(5, TimeUnit.SECONDS)
                .filterDataByLocation(new StreamObserver<PollutionLevel>() {
                    @Override
                    public void onNext(PollutionLevel value) {
                        filterOutputArea.append("Filtered data response: " +
value + "₩n");
                    }

                    @Override
                    public void onError(Throwable t) {
                        filterOutputArea.append("Error encountered in
DataVisualizerClient: " + t.getMessage() + "₩n");
                        filterLatch.countDown();
                    }

                    @Override
                    public void onCompleted() {
                        filterOutputArea.append("Filtering completed.₩n");
                        filterLatch.countDown();
                    }
                });

            try {
                // Send multiple location filter requests
                if (!location1.isEmpty()) {
                    LocationFilterRequest filterRequest1 =
```

```
LocationFilterRequest.newBuilder()
                        .setLocation(location1)
                        .build();
                filterRequestObserver1.onNext(filterRequest1);
            }

            if (!location2.isEmpty()) {
                LocationFilterRequest filterRequest2 = LocationFilterRequest.newBuilder()
                        .setLocation(location2)
                        .build();
                filterRequestObserver2.onNext(filterRequest2);
            }

            if (!location3.isEmpty()) {
                LocationFilterRequest filterRequest3 = LocationFilterRequest.newBuilder()
                        .setLocation(location3)
                        .build();
                filterRequestObserver3.onNext(filterRequest3);
            }

            filterRequestObserver1.onCompleted();

            filterRequestObserver2.onCompleted();
```

```
                    filterRequestObserver3.onCompleted();

                    // Wait for the server to complete sending data
                    if (!filterLatch.await(1, TimeUnit.MINUTES)) {
                        filterOutputArea.append("filterDataByLocation can not finish within 1 minutes₩n");
                    }
                } catch (RuntimeException e1) {
                    filterRequestObserver1.onError(e1);
                    filterRequestObserver2.onError(e1);
                    filterRequestObserver3.onError(e1);
                    throw e1;
                } catch (InterruptedException e2) {
                    e2.printStackTrace();
                } finally {
                    channel.shutdown();
                }
            }

        });

        // Set Favorite Location ActionListener
        setFavoriteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String location1 = favoriteLocationField1.getText();
```

```
                String location2 = favoriteLocationField2.getText();
                String location3 = favoriteLocationField3.getText();

                ManagedChannel channel = ManagedChannelBuilder.forAddress(resolvedIP, port).usePlaintext().build();
                DataVisualizationStub asyncStub = DataVisualizationGrpc.newStub(channel);

                CountDownLatch favoriteLatch = new CountDownLatch(1);
                StreamObserver<FavoriteLocationResponse> favoriteResponseObserver = new StreamObserver<FavoriteLocationResponse>() {
                    @Override
                    public void onNext(FavoriteLocationResponse value) {
                        favoriteOutputArea.append("Favorite location response: " + value + "₩n");
                    }

                    @Override
                    public void onError(Throwable t) {
                        favoriteOutputArea.append("Error encountered in DataVisualizerClient: " + t.getMessage() + "₩n");
                        favoriteLatch.countDown();
                    }

                    @Override
                    public void onCompleted() {
                        favoriteOutputArea.append("Favorite location setting
```

```
completed.₩n");
                    favoriteLatch.countDown();
                }
            };

            StreamObserver<FavoriteLocationRequest> favoriteRequestObserver =
asyncStub
                    .withDeadlineAfter(5, TimeUnit.SECONDS) // set a 5-second
deadline
                    .setFavoriteLocation(favoriteResponseObserver);

            try {
                if (!location1.isEmpty()) {
                    FavoriteLocationRequest favoriteRequest1 =
FavoriteLocationRequest.newBuilder()
                            .setLocation(location1)
                            .build();
                    favoriteRequestObserver.onNext(favoriteRequest1);
                }

                if (!location2.isEmpty()) {
                    FavoriteLocationRequest favoriteRequest2 =
FavoriteLocationRequest.newBuilder()
                            .setLocation(location2)
                            .build();
                    favoriteRequestObserver.onNext(favoriteRequest2);
                }
```

```
                    if (!location3.isEmpty()) {
                        FavoriteLocationRequest favoriteRequest3 = FavoriteLocationRequest.newBuilder()
                                .setLocation(location3)
                                .build();
                        favoriteRequestObserver.onNext(favoriteRequest3);
                    }

                    favoriteRequestObserver.onCompleted();

                    // Wait for the server to complete sending data
                    if (!favoriteLatch.await(1, TimeUnit.MINUTES)) {
                        favoriteOutputArea.append("setFavoriteLocation can not finish within 1 minutes₩n");
                    }
                } catch (RuntimeException e1) {
                    favoriteRequestObserver.onError(e1);
                    throw e1;
                } catch (InterruptedException e2) {
                    e2.printStackTrace();
                } finally {
                    channel.shutdown();
                }
            }
        });
```

```
        frame.setVisible(true);
    }

    // JmDNS service discovery method
    public static void discoverServiceWithJmDNS() {
        try {
            JmDNS jmdns = JmDNS.create(InetAddress.getLocalHost());

            jmdns.addServiceListener(serviceType, new ServiceDiscoverer());

            Thread.sleep(20000);
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    private static class ServiceDiscoverer 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());
        }
```

```
        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);

            // Signal that the service has been resolved
            serviceResolvedLatch.countDown();
        }
    }
}
```