

# Smart automated agriculture environment.

Distributed Systems  
April 2023

**MAETZI RAMIREZ MARTINEZ 22103848**

## Contents

1	Introduction .....	4
2	Service 1: Climate Service .....	5
2.1	Methods.....	5
2.1.1	RPC Method 1: GetTemperature.....	5
2.1.2	RPC Method 2: GetHumidity .....	5
2.1.3	RPC Method 3: GetRain .....	5
2.2	Service Definition- uses of GRPC (Proto) .....	6
2.3	Service Implementation (Server / Client) / Naming Services / Remote error handling and advanced gRPC.....	6
2.4	Client – Graphical user interface (GUI) .....	8
3	Service 2: Irrigation Service.....	10
3.1	Methods.....	10
3.1.1	RPC Method 1: SetIrrigation.....	10
3.1.2	RPC Method 2: GetIrrigStatus .....	10
3.1.3	RPC Method 3: Cancellirrigation .....	10
3.2	Service Definition- uses of GRPC (Proto) .....	10
3.3	Service Implementation (Server / Client) / Naming Services / Remote error handling and advanced gRPC.....	11
3.4	Client – Graphical user interface (GUI) .....	13
4	Service 3: Crop Service .....	15
4.1	Methods.....	15
4.1.1	RPC Method 1: GetCropStatus .....	15
4.1.2	RPC Method 2: SetCropPlan.....	15
4.2	Service Definition- uses of GRPC (Proto) .....	16
4.3	Service Implementation (Server / Client) / Naming Services / Remote error handling and advanced gRPC.....	17
4.4	Client – Graphical user interface (GUI) .....	18
5	GitHub .....	19

# 1 Introduction

The project aims to develop a distributed system that simulates the operations of a smart automated agriculture environment. The system comprises three services that work together to provide a comprehensive solution for managing the agricultural environment. These services are the Climate Service, Irrigation Service, and Crop Service.

The Climate Service is responsible for monitoring the climate conditions in the agricultural environment, such as temperature, humidity, and rainfall. It provides methods for retrieving these parameters in real time. The Irrigation Service manages the irrigation system and provides methods for setting and canceling irrigation processes, as well as checking the current status of the system. The Crop Service is responsible for managing the crops in the environment and provides methods for retrieving the growth stage and health status of crops, as well as setting a care plan for each crop.

The services communicate with each other using gRPC, a modern high-performance remote procedure call (RPC) framework, which provides efficient communication between the services. Java is the programming language used to develop the solution, which is a popular and widely used language for building distributed systems.

To manage and monitor the smart automated agriculture environment, a GUI client is developed. The GUI client provides a user-friendly interface for managing the services, monitoring the environmental parameters, and receiving alerts when any parameter falls outside the normal range.

Overall, the project aims to provide a comprehensive solution for managing an agricultural environment through the use of modern distributed system technologies. The project utilizes gRPC and Java to build efficient and scalable services that communicate with each other, and a GUI client to provide a user-friendly interface for managing the system.

For the implementation of the program, a Maven POM file is used to configure and build this project. It describes the project's metadata, dependencies, and build process.

The `<modelVersion>` element specifies that the version that it is used is 4.0.0 of the POM file format, while `<groupId>`, `<artifactId>`, and `<version>` identify the project's unique coordinates. The `<name>` element specifies the project's name: SmartAutomatedAgricultureEnvironment.

The `<dependencies>` section lists the dependencies that the project requires to compile and run. In this case, the project depends on various libraries such as `protobuf-java`, `grpc-netty-shaded`, `guava`, and others.

The `<properties>` section defines properties that can be used throughout the POM file.

The `<build>` section contains information about the build process, including the default goal, which plugins to use, and their configuration. The `<plugins>` section specifies the plugins to use, such as the `protoc-jar-maven-plugin`, which is used to generate Java code from protobuf files, and the `maven-compiler-plugin`, which is used to compile the project's Java code.

## Pom running

```
Console ×
<terminated> SmartAutomatedAgricultureEnvironment (1) [Maven Build] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ SmartAutomatedAgricultureEnvironment ---
[INFO] Installing C:\Users\iztea\OneDrive\Documentos\semester 2\Distributed Systems\CA\AgricultureMaetziRamirezMartinez
[INFO] Installing C:\Users\iztea\OneDrive\Documentos\semester 2\Distributed Systems\CA\AgricultureMaetziRamirezMartinez
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 8.870 s
[INFO] Finished at: 2023-04-07T15:41:05+01:00
[INFO] -----
```

## 2 Service 1: Climate Service

The Climate Service is designed to provide climate monitoring functionalities for agricultural environments. The service uses remote procedure call (RPC) methods to offer real-time climate data to its users. Here are the different methods the service offers:

### 2.1 Methods

#### 2.1.1 RPC Method 1: GetTemperature

*rpc GetTemperature (TemperatureRequest) returns (TemperatureResponse){}*

This method is a vital function of the Climate Service. It enables users to monitor the current temperature of the agricultural environment in Celsius.

This method is an RPC method that returns the current temperature of the environment. It takes a TemperatureRequest as input and returns a TemperatureResponse as output. The method does not require any input parameters, and the response is representing the current temperature of the environment.

#### 2.1.2 RPC Method 2: GetHumidity

*rpc GetHumidity (HumidityRequest) returns (HumidityResponse){}*

This method provides users with the current humidity level of the agricultural environment. Users can use this method to make informed decisions on how to manage the crops in the environment.

This method is an RPC method that returns the current humidity level of the environment. It takes a HumidityRequest as input and returns a HumidityResponse as output. The method does not require any input parameters, and the response is representing the current humidity level as a percentage.

#### 2.1.3 RPC Method 3: GetRain

*rpc GetRain (RainfallRequest) returns (RainfallResponse){}*

This method is an RPC method that returns the total rainfall over a specified time period. It takes a RainfallRequest as input, which requires two parameters: a start date and an end date for the period to query. The method returns a RainfallResponse as output, which is representing the total rainfall during the specified time period.

## 2.2 Service Definition- uses of GRPC (Proto)

The name of the proto file for Service 1 is called “**Service1.proto**”.

The code is written in Proto3 syntax and defines the ClimateService as an RPC service. It contains three methods: GetTemperature, GetHumidity, and GetRain, all of them are Unary RPC methods, each of which has a corresponding input and output message.

The TemperatureRequest message has a single string field "mytemperature," which is not required to be set by the user. The TemperatureResponse message also has a single string field "mytemperatureresponse," which contains the current temperature value as a string.

The HumidityRequest message has a single string field "myhumidity," which is not required to be set by the user. The HumidityResponse message has a single string field "myhumidityresponse," which contains the current humidity value as a string.

The RainfallRequest message has two string fields "mystartdate" and "myenddate" that must be set by the user to specify the time period to query. The RainfallResponse message has a single string field "myrainfallresponse," which contains the total rainfall during the specified time period.

```
1 syntax = "proto3";
2
3 // Options for the GRPC
4 option java_multiple_files = true;
5 option java_package = "grpc.ca.agriculture1";
6 option java_outer_classname = "ClimateService";
7
8 package smartAgriculture;
9
10 // Service Definition
11 service climateService{
12     rpc GetTemperature (TemperatureRequest) returns (TemperatureResponse){}
13     rpc GetHumidity (HumidityRequest) returns (HumidityResponse){}
14     rpc GetRain (RainfallRequest) returns (RainfallResponse){}
15 }
16
17 // Define the messages for GetTemperature method
18 message TemperatureRequest{
19     string mytemperature = 1; // Users can make a request for this method without any parameters,
20 }
21
22 message TemperatureResponse{
23     string mytemperatureresponse = 1; // The response is a string with floating-point value that represents the current temperature of the environment
24 }
25
26 // Define the messages for GetHumidity method
27 message HumidityRequest{
28     string myhumidity = 1;
29 }
30
```

## 2.3 Service Implementation (Server / Client) / Naming Services / Remote error handling and advanced gRPC.

The name of the server file for Service 1 is called “**Service1ClimateServer.java**”.

The server **registers the service** using the registerService method and loads properties from the properties file using the getProperties method. The registerService method creates a **JmDNS** instance and registers the service with the specified name, type, and port. The getProperties method loads the properties from the properties file and prints them to the console. In the same way, the Listener class implements the ServiceListener interface and creates a **JmDNS** for **service discovery**, and provides methods for handling service events. The server uses the gRPC framework and logs the service activity using the log4j library.

The class also contains a nested class `LoggingInterceptor` that implements the `Interceptor` interface for **remote error handling**. The `OkHttpClient` client is initialized with this interceptor.

#### Server:

```
Console x
<terminated> Service1ClimateServer [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64.jre\bin\java
Service 1 Climate properties:
  service_type: _service1_tcp.local.
  service_name: simple_ClimateService1
  service_description: service for basic information about the weather
  service_port: 50051
-----
registering service with type _service1_tcp.local. and name simple_ClimateService1
-----
Discovery _service1_tcp.local.
Service added: [ServiceInfoImpl@627597120 name: 'simple_ClimateService1_service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS state: probing 1 task: null']
Service resolved: [ServiceInfoImpl@627597120 name: 'simple_ClimateService1_service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS state: probing 1 task: null']
Server 1 for Climate Service is running...
```

The name of the client file for Service 1 is called “**Service1ClimateClient.java**”.

This is the gRPC client that connects to a gRPC server (`Service1ClimateServer`) and sends requests for three different methods: `getTemperature`, `getHumidity`, and `getRain`. It also includes a service discovery functionality using `JmDNS` to discover available services with the name “`_service1_tcp.local.`”.

First creates a channel to connect to the server using the `ManagedChannelBuilder` with the host and port information. Then, it creates a blocking stub to call the gRPC methods on the server. An `OkHttpClient` is also created with a `LoggingInterceptor` to log the request and response messages.

The three gRPC methods are called with the corresponding request message types (`TemperatureRequest`, `HumidityRequest`, and `RainfallRequest`) to get the response messages (`TemperatureResponse`, `HumidityResponse`, and `RainfallResponse`), respectively, and then the response messages are printed to the console.

Finally, `JmDNS` is used to **discover** available services with the name “`_service1_tcp.local.`” by creating a `JmDNS` instance and adding a listener for service events. The Listener class implements the `ServiceListener` interface to handle the service events: `serviceAdded`, `serviceRemoved`, and `serviceResolved`.

The channel is shut down after all requests have been sent and the service discovery functionality is finished.

#### Client:

```
Console x
<terminated> Service1ClimateClient [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64.jre\bin\java
Message send by the server
The temperature is 14 degrees
-----
Message send by the server
The humidity is: 60%
-----
Message send by the server
The rainfall: The start date is 12th April 2023 The end date is 14th April 2023
-----
Discovery _service1_tcp.local.
Service added: [ServiceInfoImpl@149997312 name: 'simple_ClimateService1_service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS state: probing 1 task: null']
Service resolved: [ServiceInfoImpl@149997312 name: 'simple_ClimateService1_service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS state: probing 1 task: null']
```

Server:

```
Service1ClimateServer [Java Application] C:\Users\iztea\OneDrive\Escritorio\ eclipse-jee-2022-12-R-win32-x86_64\ eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32_x86_64_17.0.6.v20230204-1729\jre\bin\java.exe
Service 1 Climate properties:
  service_type: _service1_tcp.local.
  service_name: simple_ClimateService1
  service_description: service for basic information about the weather
  service_port: 50051
-----
registering service with type _service1_tcp.local. and name simple_ClimateService1
-----
Discovery _service1_tcp.local.

Service added: [ServiceInfoImpl@627597120 name: 'simple_ClimateService1._service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS state: probi
Service resolved: [ServiceInfoImpl@627597120 name: 'simple_ClimateService1._service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS state: pr

Server 1 for Climate Service is running...
--Receiving Temperature Request from Client--
--Receiving Humidity Request from Client--
--Receiving Rainfall Request from Client--
```

## 2.4 Client – Graphical user interface (GUI)

The name of the GUI file for Service 1 is called “**Service1GUIApplication.java**”.

This GUI application connects to a gRPC server (Service1ClimateServer) and sends requests to get temperature, humidity, and rainfall data. The program also uses JmDNS to discover the service and obtain information about its properties, such as the port, service type, service name, service description, and host address.

The main method creates a JFrame and sets up the GUI layout with a panel that contains three buttons for getting temperature, humidity, and rainfall data, respectively. Another panel contains a JTextPane that displays the server response messages. The program then creates a gRPC channel using ManagedChannelBuilder and generates blocking and asynchronous stubs for the climateService service defined in the Service1.proto file. The program also calls the discoveryService1Climate method to discover the climate service using JmDNS.

Each button has an action listener that sends a request to the server and updates the JTextPane with the response message. The temperatureButton sends a TemperatureRequest to the server and receives a TemperatureResponse with the temperature data. The humidityButton sends a HumidityRequest to the server and receives a HumidityResponse with the humidity data. The rainfallButton sends a RainfallRequest to the server and receives a RainfallResponse with the rainfall data.

GUI:

```
Service1GUIApplication [Java Application] C:\Users\iztea\OneDrive\Escritorio\ eclipse-jee-2022-12-R-win32-x86_64\ eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32_x86_64_17.0.6.v20230204-1729\jre\bin\java.exe
Climate Service added: [ServiceInfoImpl@110021424 name: 'simple_ClimateService1._service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS stat
Climate Service resolved: [ServiceInfoImpl@110021424 name: 'simple_ClimateService1._service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS s

resolving _service1_tcp.local. with properties ...
  service_port: 50051
  service_type: _service1_tcp.local.
  service_name: simple_ClimateService1
  service_description: service for basic information about the weather
  host: 192.168.56.1
```

## GetTemperature



## GetHumidity



## GetRain



## Server:

```
Console x
Service1ClimateServer [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin
Service 1 Climate properties:
  service_type: _service1_tcp.local.
  service_name: simple_ClimateService1
  service_description: service for basic information about the weather
  service_port: 50051
-----
registrering service with type _service1_tcp.local. and name simple_ClimateService1
-----
Discovery _service1_tcp.local.
Service added: [ServiceInfoImpl@627597120 name: 'simple_ClimateService1._service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS state: probi
Service resolved: [ServiceInfoImpl@627597120 name: 'simple_ClimateService1._service1_tcp.local.' address: '/192.168.56.1:50051' status: 'NO DNS state: pr
-----
Server 1 for Climate Service is running...
--Receiving Temperature Request from Client--
--Receiving Humidity Request from Client--
--Receiving Rainfall Request from Client--
--Receiving Temperature Request from Client--
--Receiving Humidity Request from Client--
--Receiving Rainfall Request from Client--
--Receiving Humidity Request from Client--
--Receiving Temperature Request from Client--
```



### 3 Service 2: Irrigation Service

The Irrigation Service is a vital component of a smart automated agriculture environment. The service manages the irrigation system and provides users with real-time data on the system's status. Here are the different methods that the Irrigation Service offers:

#### 3.1 Methods

##### 3.1.1 RPC Method 1: SetIrrigation

*rpc SetIrrigation (IrrigationParametersRequest) returns (IrrigationStatusResponse){}*

This method is responsible for setting the irrigation system according to the user's request. The method takes in a request that includes irrigation parameters, which are floating-point values that represent the irrigation requirements for the crops. The response to this method represents the irrigation system's current status.

##### 3.1.2 RPC Method 2: GetIrrigStatus

*rpc GetIrrigStatus (LocationParametersRequest) returns (stream CurrentStatusResponse){}*

This method enables users to get real-time data on the irrigation system's status. The method requires location parameters to specify the area of interest. Users can make a request for this method, and the response is a string that represents the current status of the irrigation system.

##### 3.1.3 RPC Method 3: CancellIrrigation

*rpc CancellIrrigation (stream CancelationRequest) returns (stream CancelationResponse){}*

This method enables users to cancel the current irrigation process. The method takes in a request that includes a string to specify the cancellation. This functionality is useful when the user realizes that the irrigation process was started in error or if the crop's irrigation requirements have changed.

#### 3.2 Service Definition- uses of GRPC (Proto)

The name of the proto file for Service 2 is called "**Service2.proto**".

This is a Protocol Buffer file written in Proto3 syntax. It defines a gRPC service for smart agriculture and it is named irrigationService. The irrigationService contains three methods:

SetIrrigation is a Unary RPC method that takes an IrrigationParametersRequest message and returns an IrrigationStatusResponse message. IrrigationParametersRequest has one field "myirrigationparameters", and the IrrigationStatusResponse has one field "myirrigationstatusResponse" of type string.

GetIrrigStatus is a Streaming RPC method that takes a LocationParametersRequest message and returns a stream of CurrentStatusResponse messages. The LocationParametersRequest has two fields "latitude" and "longitude", both of type float, and CurrentStatusResponse has one field "mycurrentstatusResponse" of type string.

CancellIrrigation is a Bidirectional method that takes a stream of CancelationRequest messages and returns a stream of CancelationResponse messages. CancelationRequest has one field

"mycancelationrequest", and CancelationResponse has one field "mycancelationresponse" of type string both of them.

```
Service2IrrigationServer.java Service2.proto X
1 syntax = "proto3";
2
3 // Options for the GRPC
4 option java_multiple_files = true;
5 option java_package = "grpc.ca.agriculture2";
6 option java_outer_classname = "IrrigationService";
7
8 package smartAgriculture;
9
10 // Service Definition
11 service IrrigationService{
12     rpc SetIrrigation (IrrigationParametersRequest) returns (IrrigationStatusResponse){}
13     rpc GetIrrigStatus (LocationParametersRequest) returns (stream CurrentStatusResponse){}
14     rpc CancelIrrigation (stream CancelationRequest) returns (stream CancelationResponse){}
15 }
16
17 // Define the messages for SetIrrigation method
18 message IrrigationParametersRequest{
19     string myirrigationparameters = 1;
20 }
21
22 message IrrigationStatusResponse{
23     string myirrigationstatusResponse = 1;
24 }
25
26 // Define the messages for GetIrrigStatus method
27 message LocationParametersRequest{
28     float latitude = 1;
29     float longitude = 2;
30 }
31
32 message CurrentStatusResponse{
33     string mycurrentstatusResponse = 1;
34 }
35
```

### 3.3 Service Implementation (Server / Client) / Naming Services / Remote error handling and advanced gRPC.

The name of the server file for Service 2 is called "Service2IrrigationServer.java".

Which is the implementation of a gRPC server for irrigation service. The server listens on a given port and provides the three RPC methods.

The code also includes **JmDNS** service **registration** and **discovery**. When the server starts, it registers itself as a service with **JmDNS**, and when it receives a discovery event for the service, it prints out the event information. The code uses OkHttp for logging network requests and responses.

**main()**: The entry point of the server application. It creates an instance of the Service2IrrigationServer class and starts the gRPC server. It also registers the server with JmDNS for service discovery.

**getProperties()**: Loads the configuration properties from the Service2.properties file and returns them as a Properties object.


**registerService()**: Registers the service with JmDNS using the properties loaded from the Service2.properties file.

**Listener:** A nested class that implements the ServiceListener interface to handle JmDNS service discovery events.

**LoggingInterceptor:** A nested class that implements the Interceptor interface for **remote error handling** from OkHttp to log network requests and responses.

The implementation of the three methods setIrrigation(), getIrrigStatus() and CancellIrrigation(), receive their respective inputs and get their respective outputs, which are printing in the consol.

## Server



```
Service2IrrigationServer [Java Application] C:\Users\jztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\
Service 2 Irrigation properties:
  service_type: _service2._tcp.local.
  service_name: simple_IrrigationService2
  service_description: The Irrigation Service is a vital component of a smart automated agriculture environment. The service manages the irrigation
  service_port: 50052

-----
registrering service with type _service2._tcp.local. and name simple_IrrigationService2
-----

Discovered _service2._tcp.local.

Service added: [ServiceInfoImpl@1014634006 name: 'simple_IrrigationService2._service2._tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: prob
Service resolved: [ServiceInfoImpl@2096231274 name: 'simple_IrrigationService2._service2._tcp.local.' address: '/192.168.56.1:50052 ' status: 'DNS: MSI.loc

Server 2 for Irrigation Service is running...
```

The name of the client file for Service 2 is called “**Service2IrrigationClient.java**”.

This is the gRPC client that connects to a gRPC server (Service2IrrigationServer). The client is using both the blocking and asynchronous stubs to interact with the service.

The main method starts by creating a managed channel with a plain-text connection to the server. It then creates an OkHttpClient with a LoggingInterceptor to log the requests and responses of the client.

After setting up the stubs, the main method invokes three different RPC methods of the service: SetIrrigation(), GetIrrigStatus(), and CancellIrrigation(). Each method makes use of either the blocking or asynchronous stubs to send requests to the server and receive responses.

The client also includes a JmDNS listener to discover any services with the name "\_service2.\_tcp.local.". When a service event occurs, the client will print a message indicating whether the service was added, removed, or resolved.

## Client

```
Console x
<terminated> Service2IrrigationClient [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230202\jre\bin\java.exe
Message send by the server:
The irrigation parameters are between 10 to 15
-----
Message send by the server:
Current Latitude and Longitude 51.031.0
Message send by the server:
Current Latitude and Longitude 51.031.0
Message send by the server:
Current Latitude and Longitude 51.031.0
Message send by the server:
Current Latitude and Longitude 51.031.0
Message send by the server:
Current Latitude and Longitude 51.031.0
Message send by the server:
Current Latitude and Longitude 51.031.0
Current Status
-----
Receiving cancellation response Cancel irrigation
Cancellation is completed
-----
Discovery _service2._tcp.local.

Service added: [ServiceInfoImpl@1491496599 name: 'simple_IrrigationService2._service2._tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: prob
Service resolved: [ServiceInfoImpl@1445043195 name: 'simple_IrrigationService2._service2._tcp.local.' address: '/192.168.56.1:50052 ' status: 'DNS: MSI.loc
```

## Server

```
Console x
Service2IrrigationServer [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\java.exe
Service 2 Irrigation properties:
  service_type: _service2._tcp.local.
  service_name: simple_IrrigationService2
  service_description: The Irrigation Service is a vital component of a smart automated agriculture environment. The service manages the irrigation
  service_port: 50052
-----
registering service with type _service2._tcp.local. and name simple_IrrigationService2
-----
Discovered _service2._tcp.local.

Service added: [ServiceInfoImpl@1014634006 name: 'simple_IrrigationService2._service2._tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: prob
Service resolved: [ServiceInfoImpl@2096231274 name: 'simple_IrrigationService2._service2._tcp.local.' address: '/192.168.56.1:50052 ' status: 'DNS: MSI.loc
-----
Server 2 for Irrigation Service is running...
--Receiving Irrigation Parameters from Client--
--Receiving Location Parameters Request from Client--
Receiving Cancellation Request:
Client has completed the request.
```

## 3.4 Client – Graphical user interface (GUI)

The name of the GUI file for Service 2 is called “**Service2IrrigationServer.java**”.

This is the graphical user interface (GUI) for irrigation services. The program connects to the service via gRPC and allows the user to set an irrigation schedule, get the current irrigation status, and cancel an irrigation schedule.

The GUI has three buttons: "Set Irrigation," "Get Irrigation Status," and "Cancel Irrigation." When the "Set Irrigation" button is clicked, the program sends a message to the irrigation service to set the irrigation schedule. When the "Get Irrigation Status" button is clicked, the program sends a message to the irrigation service to get the current irrigation status, and displays the response in a text pane. When the "Cancel Irrigation" button is clicked, the program sends a message to the irrigation service to cancel the irrigation schedule.

The program uses JmDNS to discover the irrigation service on the local network. Once the service is discovered, is created a gRPC channel to connect to the service. Then, are created two stubs from the irrigationServiceGrpc class: a blocking stub and an asynchronous stub. The blocking stub is used for the "Set Irrigation" and "Get Irrigation Status" functions, while the asynchronous stub is used for the "Cancel Irrigation" function.

## GUI

```
Service2GUIApplication [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64.17.0.6.v20230204-1729\jre\
Irrigation Service added: [ServiceInfoImpl@1359303467 name: 'simple_IrrigationService2_service2_tcp.local.' address: '(null):0' status: 'DNS: MSI.local.'
Irrigation Service resolved: [ServiceInfoImpl@1606596475 name: 'simple_IrrigationService2_service2_tcp.local.' address: '/192.168.56.1:50052' status: 'D
resolving _service2_tcp.local. with the following properties:
service_type: _service2_tcp.local.
service_name: simple_IrrigationService2
service_description: The Irrigation Service is a vital component of a smart automated agriculture environment. The service manages the irrigation system a
service_port: 50052
host: 192.168.56.1
```

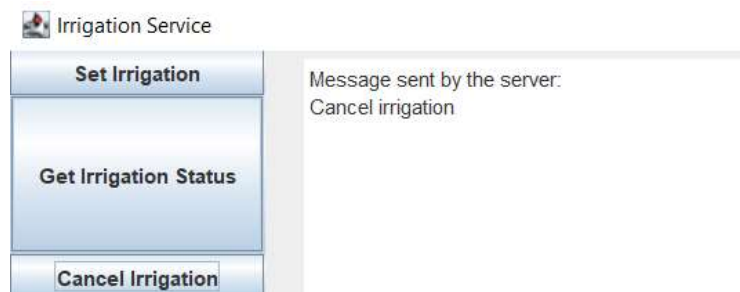
## Set Irrigation



## Get Irrigation Status



## Cancel Irrigation



```

Console ×
Service2GUIApplication [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.6.v20230204-1729\jre\bin
Irrigation Service added: [ServiceInfoImpl@1359303467 name: 'simple_IrrigationService2_service2_tcp.local.' address: '(null):0' status: 'DNS: MSI.local.'
Irrigation Service resolved: [ServiceInfoImpl@1606596475 name: 'simple_IrrigationService2_service2_tcp.local.' address: '/192.168.56.1:50052' status: 'DNS: MSI.local.']
resolving _service2_tcp.local. with the following properties:
service_type: _service2_tcp.local.
service_name: simple_IrrigationService2
service_description: The Irrigation Service is a vital component of a smart automated agriculture environment. The service manages the irrigation system and
service_port: 50052
host: 192.168.56.1
Current Status
Cancellation is completed

```

## Server

```

Console ×
Service2IrrigationServer [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.17.0.6.v20230204-1729\jre\bin
Service 2 Irrigation properties:
service_type: _service2_tcp.local.
service_name: simple_IrrigationService2
service_description: The Irrigation Service is a vital component of a smart automated agriculture environment. The service manages the irrigation
service_port: 50052
-----
registrering service with type _service2_tcp.local. and name simple_IrrigationService2 |
-----
Discovered _service2_tcp.local.
Service added: [ServiceInfoImpl@1014634006 name: 'simple_IrrigationService2_service2_tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: prob
Service resolved: [ServiceInfoImpl@2096231274 name: 'simple_IrrigationService2_service2_tcp.local.' address: '/192.168.56.1:50052' status: 'DNS: MSI.loc
-----
Server 2 for Irrigation Service is running...
--Receiving Irrigation Parameters from Client--
--Receiving Location Parameters Request from Client--
Receiving Cancellation Request:
Client has completed the request.

```

## 4 Service 3: Crop Service

The Crop Service is responsible for providing information about crop growth and health. It enables users to monitor the status of their crops, identify issues and optimize crop care practices. The following are the methods offered by the Crop Service:

### 4.1 Methods

#### 4.1.1 RPC Method 1: GetCropStatus

*rpc GetCropStatus (stream CropTypeRequest) returns (CropStatusResponse){}*

This method is responsible for providing real-time data on the crop's growth stage. The method requires a request message containing the crop type and location as input parameters. The response to this method is a message containing the crop's growth stage and health status. This information is critical for farmers to monitor crop development and identify any issues that need to be addressed.

#### 4.1.2 RPC Method 2: SetCropPlan

*rpc SetCropPlan (CropPlanRequest) returns (stream CropPlanResponse){}*

This method enables users to set a care plan for their crops. The method requires a request message containing the crop type. The care plan includes the frequency of watering, fertilization, and pest control. The response to this method is a string that represents the corresponding plan for the crop. This functionality enables farmers to optimize their crop care practices and maximize yield.



## 4.2 Service Definition- uses of GRPC (Proto)

The name of the proto file for Service 3 is called “**Service3**.proto”.

This is a protocol buffer file with the syntax version set to "proto3". It defines a gRPC service called "cropService" in the "smartAgriculture" package with two methods: "GetCropStatus" and "GetCropPlan".

The "GetCropStatus" method takes a stream of "CropTypeRequest" messages and returns a single "CropStatusResponse" message. The "CropTypeRequest" message has a single field called "mycroptype" of type string, and the "CropStatusResponse" message has a single field called "mycropstatus" of type string.

The "GetCropPlan" method takes a single "CropPlanRequest" message and returns a stream of "CropPlanResponse" messages. The "CropPlanRequest" message has a single field called "mycroptype" of type string, and the "CropPlanResponse" message has a single field called "cropPlan" of type string.

```
Service3.proto x Service3CropServer.java
1  syntax = "proto3";
2
3  // Options for the GRPC
4  option java_multiple_files = true;
5  option java_package = "grpc.ca.agriculture3";
6  option java_outer_classname = "CropService";
7
8  package smartAgriculture;
9
10 // Service Definition
11 service cropService{
12     rpc GetCropStatus (stream CropTypeRequest) returns (CropStatusResponse){}
13     rpc GetCropPlan (CropPlanRequest) returns (stream CropPlanResponse){}
14 }
15
16 // Define the messages for GetCropStatus method
17 message CropTypeRequest{
18     string mycroptype = 1;
19 }
20
21 message CropStatusResponse{
22     string mycropstatus = 1;
23 }
24
25 // Define the messages for GetCropPlan method
26 message CropPlanRequest{
27     string mycroptype = 1;
28 }
29
30 message CropPlanResponse{
31     string cropPlan = 1;
32 }
33
```

### 4.3 Service Implementation (Server / Client) / Naming Services / Remote error handling and advanced gRPC.

The name of the server file for Service 2 is called “**Service3CropServer.java**”.

This is the implementation of a gRPC server for a crop service. It registers itself on the network using JmDNS and waits for incoming requests.

When the main method is executed, it first creates an instance of the Service3CropServer class and then gets the properties for the server from the Service3.properties file located in the src/main/resources directory. After that, it registers the service on the network by creating an instance of the JmDNS class and registering the serviceInfo object with it.

Then, it initializes the Log4j library by reading the configuration file Service3.properties. After that, it sets the server's port number to the value specified in the Service3.properties file and starts the server.

The code is defined as a nested class Listener which implements the ServiceListener interface. This class is used to listen and discover to service events and print messages to the console when a service is added, removed, or resolved.

Finally, the code defines an inner class LoggingInterceptor which implements the Interceptor interface. This class is used to log messages related to network communication between the server and the client using the OkHttp library and for remote error handling.

#### Server



```
Service3CropServer [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk hotspot.jre.full.win32.x86_64.17.0.6.v20230204-1729\jre\bin\java
Service 3 Crop properties:
  service_type: _service3._tcp.local.
  service_name: simple_CropService3
  service_description: The Crop Service is responsible for providing information about crop growth and health. It enables users to monitor the statu
  service_port: 50054
-----
  registering service with type _service3._tcp.local. and name simple_CropService3
-----
Discovered _service3._tcp.local.
Service added: [ServiceInfoImpl@910191298 name: 'simple_CropService3._service3._tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: probing 1 t
-----
Server 3 for Crop Service is running...
```

The name of the client file for Service 2 is called “**Service3CropClient.java**”.

This code is an implementation of a gRPC client for a crop service. The client connects to a gRPC server (Service3CropServer) using a plaintext channel and uses both blocking and asynchronous stubs to communicate with the server.

The main() method creates a plaintext channel with the server using the hostname and port number. Then it creates both blocking and asynchronous stubs to communicate with the server. The GetCropStatus() and GetCropPlan() methods are used to call the respective gRPC methods on the server.

The code includes a JmDNS instance for service discovery. It listens for service events for the \_service3.\_tcp.local. service type and prints out the details of any discovered services.



In the `GetCropStatus()` method, the client sends a stream of `CropTypeRequest` messages to the server using the asynchronous stub and receives a stream of `CropStatusResponse` messages as a response using a stream observer. The stream observer is responsible for processing the server responses asynchronously. The client sleeps for a short period between sending each request to simulate a delay.

In the `GetCropPlan()` method, the client sends a single `CropPlanRequest` message to the server using the asynchronous stub and receives a single `CropPlanResponse` message as a response using a stream observer. The `CropPlanRequest` message contains information about the types of crops being grown. The client then sleeps for a longer period to simulate a delay.

### Client

```
Console X
<terminated> Service3CropClient [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-
Message send by the server
The crop status is mixed
-----
Crop Status finished
-----
Message send by the server:
This is the crop plan: Unhealthy
-----
Message send by the server:
This is the crop plan: Unhealthy
-----
Message send by the server:
This is the crop plan: Unhealthy
-----
Message send by the server:
This is the crop plan: Unhealthy
-----
Message send by the server:
This is the crop plan: Unhealthy
-----
Crop Plan
-----
Discovered _service3._tcp.local.
Service added: [ServiceInfoImpl@631995763 name: 'simple_CropService3._service3._tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: probing 1 t
```

### Server

```
Console X
Service3CropServer [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\j
Service 3 Crop properties:
  service_type: _service3._tcp.local.
  service_name: simple_CropService3
  service_description: The Crop Service is responsible for providing information about crop growth and health. It enables users to monitor the statu
  service_port: 50054
-----
  registering service with type _service3._tcp.local. and name simple_CropService3
Discovered _service3._tcp.local.
Service added: [ServiceInfoImpl@910191298 name: 'simple_CropService3._service3._tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: probing 1 t
Server 3 for Crop Service is running...
Receiving Crop Type Request
--Receiving Crop Plan Request from Client--
```

## 4.4 Client – Graphical user interface (GUI)

The name of the GUI file for Service 3 is called “**Service3GUIApplication.java**”.

This is a GUI application for a gRPC-based crop service. It uses the `cService3.proto` file to generate the gRPC client and server stubs. The application includes two buttons: `Get CropStatus` and `Get CropPlan`. When the user clicks the `Get CropStatus` button, the client sends a stream of `CropTypeRequest` messages to the server and receives a stream of `CropStatusResponse` messages from the server. Each

CropTypeRequest message specifies a crop type for which the server returns a corresponding crop status message. When the user clicks the Get CropPlan button, the client sends a CropPlanRequest message to the server and receives a CropPlanResponse message from the server. The CropPlanRequest message specifies a crop plan for which the server returns a corresponding crop plan message. The program also uses JmDNS to discover the crop service running on the local network.

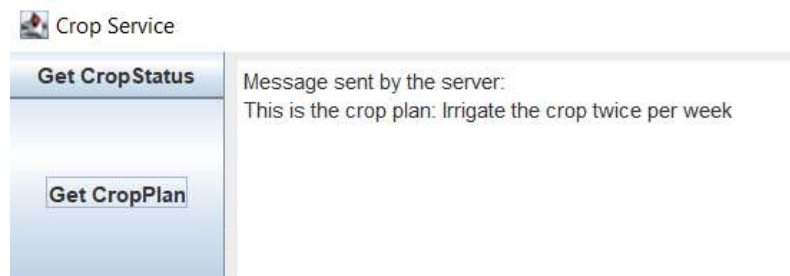
## GUI

```
Console x
Service3GUIApplication [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\
Crop Service added: [ServiceInfoImpl@360196053 name: 'simple_CropService3._service3._tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: probing'
Crop Status finished
Crop Plan
Crop Status finished
```

### Get Crop Status



### Get Crop Plan



## Server

```
Console x
Service3CropServer [Java Application] C:\Users\iztea\OneDrive\Escritorio\eclipse-jee-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_17.0.6.v20230204-1729\jre\bin\jav
Service 3 Crop properties:
  service_type: _service3._tcp.local.
  service_name: simple_CropService3
  service_description: The Crop Service is responsible for providing information about crop growth and health. It enables users to monitor the statu
  service_port: 50054
-----
  registering service with type _service3._tcp.local. and name simple_CropService3
Discovered _service3._tcp.local.
Service added: [ServiceInfoImpl@910191298 name: 'simple_CropService3._service3._tcp.local.' address: '(null):0' status: 'DNS: MSI.local. state: probing 1 t
-----
Server 3 for Crop Service is running...

Receiving Crop Type Request
--Receiving Crop Plan Request from Client--
Receiving Crop Type Request
```

## 5 GitHub

<https://github.com/Izte/AgricultureMaetziRamirezMartinez>