Name:	Christoper Chan
Student Number:	20143087
Module:	Distributed Systems - Continuous Assessment Report
Domain:	Wearables
IDE:	IntelliJ IDEA
Project Type:	Gradle Build
Operating System:	MacOS
Github Repo:	*https://github.com/christobelc/dogcollar-grpc/tree/master/src/main*
Video Presentation:	*NCI Email* & *Gmail*

gRPC Dog Collar Wearable Report

Table of Contents:

Introduction:

Consistent exercise is key to maintaining a healthy lifestyle. Regular exercise is the key to staying fit and dogs are no exception to this(ref1). Exercise is one of your dog's basic needs. In general the amount of exercise that your dog needs varies depending on age, breed and tolerance. An older Shih Tzu might want to lounge around the house all day while a young Labrador might play outside for 5 hours and still want more exercise. No two dogs are the same so discovering a dogs exercise needs by logging a dog's data is an interesting domain that I will explore in this project.

In the fitness wearables space, people use smartwatches and fitness trackers to log the data of their exercise routines. In my project I will assume that there is a collar/chest harness device that a dog can wear and it has a heart-rate sensor (for recording how fast the heart is beating), pedometer (for measuring step count), ambient light sensor (for measuring visible brightness), thermometer (for measuring body temperature), LED light, small speaker (for outputting audio), GPS (for retrieving location coordinates) and has cellular connectivity(for sending and receiving data). The collar is going to log the data of the dog and provide the basis for 3 services to be implemented using gRPC.

In our system (*Fig1*), the client is represented by the Owner of the dog and the server by the Dog Collar. Both the Dog Collar and the Owner's interface have cellular connectivity so they can communicate with each other.

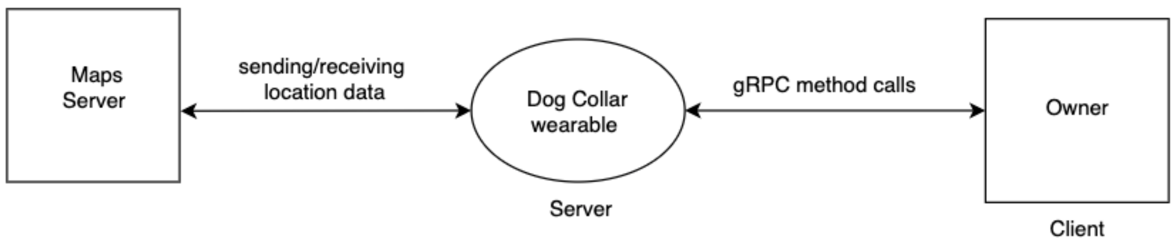


Fig1: System Overview

Service Definitions:

The methods within the services developed are in keeping for what the project specifications require but they can vary greatly from how real world implementations would be. For the purpose of sending and receiving different data via gRPC methods calls, I focused more on this than real world execution. The service definitions are defined in the proto files. They specify the data types and how messages should be encoded, decoded, sent and received facilitated by gRPC methods. The three services I have defined are Dog Tracking, Health Service and Collar Service. Within each of these services a number of gRPC methods can be invoked.

1 Dog Tracking:

Filepath: src/main/proto/dogtracking/dogtracking.proto

1.1 Is the dog wearing the collar - (collar status). (*Unary gRPC)*

Method overview

This method retrieves the status of the dog collar. Is it being worn by the dog? The current body temperature of the dog. The current heartbeat of the dog.

Protocol buffer defined methods, messages and types.

Request: WearingCollarRequest = currentStatus

Response: WearingCollarResponse = result

Unary gRPC Method: WearingCollar(WearingCollarRequest) returns (WearingCollarResponse)

Variable Name	Data Type	Purpose
wearing	bool	To notify whether the collar is being worn by the dog or not
heartBeatSensorBPM	int32	Hold value for beats per minute of the dog. e.g. 60bpm
thermometerBody-Temp	double	Hold value for current body temperature in celsius. e.g 38.5c
currentStatus	CurrentStatus	Hold the values of wearing, heartBeatSensorBPM and thermometerBodyTemp
result	string	Hold the value of the response message

1.2 Alerts if the dog strays out of pre-defined safe zone. (*Server Streaming gRPC*)

Method overview

This method lets the client outline a safety zone defining four coordinates(*Fig2*). If the dog goes outside of the defined safety zone, the method call will stream the dogs/collar's location.

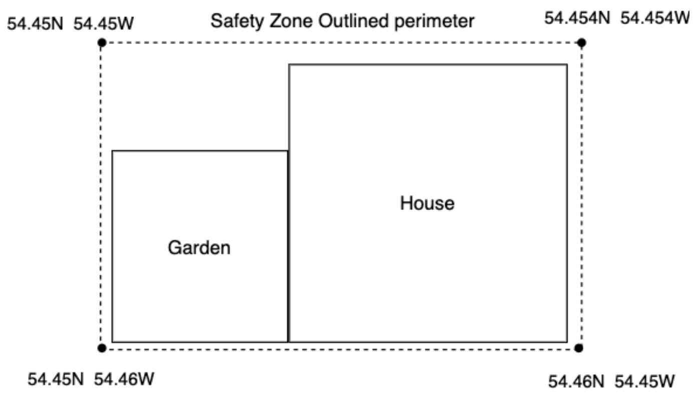


Fig2 Safety Zone example

Protocol buffer defined methods, messages and types

Request: SafetyZoneRequest = safeZoneCoordinates1, safeZoneCoordinates2, safeZoneCoordinates3, safeZoneCoordinates4,

Response: UpdateLocationResponse = result

Server Streaming gRPC: outOfBoundsLocation(WearingCollarRequest) returns (stream UpdateLocationResponse)

Variable Name	Data Type	Purpose
latitude	double	In an example location: 54.45N 59.67W. This variable represents the 54.45 value
longitude	double	In an example location: 54.45N 59.67W. This variable represents the 59.67 value
safetyZoneCoordinates1	SafetyZoneCoordinates	Holds value for the latitude and longitude of a specified location point 1
safetyZoneCoordinates2	SafetyZoneCoordinates	Holds value for the latitude and longitude of a specified location point 2

safetyZoneCoordinates3	SafetyZoneCoordinates	Holds value for the latitude and longitude of a specified location point 3
safetyZoneCoordinates4	SafetyZoneCoordinates	Holds value for the latitude and longitude of a specified location point 4
result	string	Hold the string representation for the response(dog's location)

1.3 Find the dog. (*Bi-directional Streaming*)

Method overview

This method streams the dogs location and the owners location simultaneously.

Protocol buffer defined methods, messages and types.

Request: OwnerLocationRequest = ownersCoordinates

Response: DogLocationResponse = dogCoordinates

Bi-directional Streaming gRPC: FindTheDog(stream OwnerLocationRequest) returns (stream DogLocationResponse)

Variable Name	Data Type	Purpose
dogCoordinates	string	String representation of the dog's location coordinates
ownersCoordinates	string	String representation of the owner's location coordinates

2 Health Service:

Filepath: src/main/proto/healthservice/healthservice.proto

2.1 Log the dogs heart-rate in bpm(beat per minute) (*Server Streaming*)

Method overview

This method checks if the heartbeat sensor is ready and then streams back the heart-rate in BPM.

Protocol buffer defined methods, messages and types.

Request: HeartBeatSensorRequest = heartBeatSensorStatus

Response: DogLocationResponse = result, bpm

Server Streaming gRPC: bpmCounter(HeartBeatSensorRequest) returns (stream HeartBeat-SensorResponse)

Variable Name	Data Type	Purpose
isActive	bool	To notify whether the heartbeat sensor is ready.
previousUsageTime	double	Hold value for previous usage of the heartbeat sensor. format: hh.mm e.g. “12.45” (12 hours and 45 minutes)
result	string	The result of the heartbeat sensor reading in string format.
bpm	int32	The beats per minute in integer format.
heartbeatSensorStatus	HeartbeatSensorStatus	Hold the values of isActive and previousUsage-Time

2.2 Dog step counter (*Server Streaming*)

Method overview

This method checks if the pedometer is ready to be used on the collar and when on records the amount of steps the dog takes.

Protocol buffer defined methods, messages and types.

Request: PedometerRequest = pedometerStatus

Response: PedometerResponse = result, avgSpeed, currentCount

Server Streaming gRPC: stepCounter(PedometerRequest) returns (stream PedometerResponse)

Variable Name	Data Type	Purpose
isActive	bool	To notify whether the pedometer sensor is ready.
previousCount	int32	Hold value for previous amount of steps recorded. e.g. 4500 steps

result	string	The result of the pedometer.
bpm	int32	The beats per minute in integer format.
avgSpeed	double	Hold the value of the average speed. e.g 10 km/ph
currentCount	int32	Hold the value of the number of steps taken in this session of tracking: e.g 1000 steps
pedometerStatus	PedometerStatus	Holds value for isActive and previousCount variables

2.3 Dog's temperature (*Unary gRPC*)

Method overview

This method checks the collars thermometer and returns the body temperature of the dog in Celsius and in Fahrenheit.

Protocol buffer defined methods, messages and types.

Request: TemperatureRequest = currentTempRequest

Response: TemperatureResponse = result

Unary gRPC Method: checkTemperature(TemperatureRequest) returns (TemperatureResponse)

Variable Name	**Data Type**	**Purpose**
currentTempRequest	int32	Holds the value for the current Temperature
result	string	String representation of the body temperature of the dog in Celsius and Fahrenheit.

3 Dog Collar Service:

Filepath: src/main/proto/healthservice/healthservice.proto

3.1 Dog come home (*Client Streaming*)

Method overview

This method takes multiple messages from the owner and sends them to the dog collar. In theory voice messages will be sent from the owner to the dog. I will sent text instead.

Protocol buffer defined methods, messages and types.

Request: StreamVoiceRequest = voiceMessages

Response: StreamVoiceResponse = result

Client Streaming gRPC: streamVoice(stream StreamVoiceRequest) returns

(StreamVoiceResponse)

Variable Name	Data Type	Purpose
message1	string	A specific message that the owner wants to send to the dog.
message2	string	A specific message that the owner wants to send to the dog.
message3	string	A specific message that the owner wants to send to the dog.
result	string	The result of the sent messages to the dog.
voiceMessages	VoiceMessages	Hold the values of message1, message2 and mes-sage3.

3.2 Turn on LED light and alert owner its dark outside(*Server Streaming*)

Method overview

This method checks the brightness outside from the light sensor on the collar. It turns on the light if it is dark outside. Also it send the owner this information.

Protocol buffer defined methods, messages and types.

Request: LedLightRequest = ledLightStatus

Response: LedLightResponse = result

Client Streaming gRPC: collarLight(LedLightRequest) returns

(stream LedLightResponse)

Variable Name	Data Type	Purpose
turnOn	bool	To notify whether the LED Light is on or off.

battery	int32	Hold value for battery percentage of the LED light. e.g 98%
ledLightStatus	LedLightStatus	Hold values of the variables turnOn and battery
result	string	Hold the value of the response message

Service Implementation:

All the gRPC service implementations have a server and client component. They are based on the remote procedure call concept of a client sending request and the server sending back response. The client implementations reside in the GUIs(the entry points for the services). The server implementations are located in files that end with “Impl.java”.

Server:

The service implementations on the server side are method calls that are defined by the service and execute a gRPC server to deal with incoming client requests. gRPC handles the incoming client method calls, executes the method and encodes and sends it own response.

Client:

All client implementations require instantiating of a stub to communicate with the server. In the case of Unary and Server streaming method calls a non-blocking stub is required to make calls to the server whereas for client streaming and bi-directional streaming an asynchronous stub is needed to return the response asynchronously. Each stub wraps a predefined user specified channel and the stub uses the channel to send gRPC methods to the service.

1 Dog Tracking:

Client gui Filepath: src/main/java/com.github/clientgui/DogTrackerGUI.java

Server Impl Filepath: src/main/java/com.github/dogtracking.grpc/server/DogTrackingImpl.java

1.1 Is the dog wearing the collar - (collar status).

Unary gRPC Method: WearingCollar(WearingCollarRequest) returns (WearingCollarResponse)

The client sends one request and receives one response back.

1.1 Client:

- Synchronous Blocking stub dogtracker is created
- Inputs from gui for bpm, bodyTemp and isWearing (checks if dog is wearing collar)
- CurrentStatus obj is created binding the values for bpm, bodyTemp and isWearing
- WearingCollarRequest is created and the currentStatus is passed to the Obj.
- WearingCollarResponse is built, the gRPC method is called passing WearingCollarRequest.

1.1 Server gRPC Method:

- CurrentStatus request is retrieved, all values are read assigned to local variables.
- String result concatenates the values of wearing, heartBeatBPM and thermometerBodyTemp
- WearingCollarResponse is created, the result is passed to the Obj.
- responseObserver sends back the response .onNext()
- responseObserver closes communicated with onCompleted()

1.2 Alerts if the dog strays out of pre-defined safe zone.

Server Streaming gRPC: outOfBoundsLocation(WearingCollarRequest) returns (stream UpdateLocationResponse)

1.2 Client:

- Synchronous Blocking stub dogtracker is created
- Inputs from gui for coordinates1, coordinates2, coordinates3 and coordinates4 are parsed
- SafetyZoneRequest is created and the values for coordinates1, coordinates2, coordinates3 and coordinates4 are passed into the Obj.
- gRPC method outOfBoundsLocation is called and the safetyZoneRequest is passed. For each remaining response from the server, output values to our gui.

1.2 Server gRPC Method:

- 2d Array of safetyZone values are created from the request Obj containing coordinates1, coordinates2, coordinates3 and coordinates4
- Since this is a simulation, hard code a value for dog's Location: 54.7287678, 52.4566476
- Check if dog location outside the first value of safetyZone
- If dog location outside the first value of safetyZone then create some fake movement data for the dog's location
- Create a string result of the dog's movement and pass it to the response, UpdateLocationResponse.
- responseObserver passes the response, onNext()
- Complete the gRPC call with .onComplete()

1.3 Find the dog.

Bi-directional Streaming gRPC: FindTheDog(stream OwnerLocationRequest) returns (stream DogLocationResponse)

1.3 Client:

- Asynchronous stub asyncClient is created
- StreamObserver for the request, OwnerLocationRequest, is created and the findTheDog() method is called passing a new StreamObserver for the response, DogLocationResponse.
- Implement the abstract methods for onNext and onCompleted. The onNext method retrieves the values from the DogLocationResponse from the server. onComplete method concludes the messages received from server.
- Client Streaming part: get and parse the dogsCoordinates from the gui. Copy to an array of fake dog's coordinates data. For each of the fake dog's coordinates, onNext, print to the gui and also pass to the request, OwnerLocationRequest, the coordinates.

1.3 Server gRPC Method:

- Create a new StreamObserver of OwnerLocationRequest, requestObserver. Implement the abstract methods for the new StreamObserver.
- Retrieving the value for coordinate passed my the OwnerLocationRequest. We make a new string of OwnersCoordinates and pass int back to the DogLocationResponse Obj, response.
- Complete the gRPC call with onCompleted()

2 Health Service:

Client gui Filepath: src/main/java/com.github/clientgui/HealthServiceGUI.java

Server Impl Filepath: src/main/java/com.github/healthservice.grpc/server/HealthServiceImpl.java

2.1 Log the dogs heart-rate in bpm(beat per minute)

Server Streaming gRPC: bpmCounter(HeartBeatSensorRequest) returns (stream HeartBeat-SensorResponse)

2.1 Client:

- Synchronous Blocking stub healthService is created
- Inputs from gui for isActive(the heartbeat sensor ready), previousUsageTime(the time until now using the heartbeat sensor)
- Create our request, heartBeatSensorRequest, passing the values for isActive and previous-UsageTime and then build.
- For each remaining response of heartbeatSensorResponse print to the gui the results.

2.1 Server gRPC Method:

- Get the request values for IsActive and previousUsageTime.
- Creating fake heartbeat data using a for loop for 10 seconds.
- Parse the values for usageTime in hours and minutes. Then build a string result showing the current usage details.
- Create the response passing in the result.
- responseObserver closes communicated with onCompleted()

2.2 Dog step counter

Server Streaming gRPC: stepCounter(PedometerRequest) returns (stream PedometerResponse)

2.2 Client:

- Synchronous Blocking stub healthService is created.
- Inputs from gui for isActive(the pedometer ready), previousCount(the log of the previous count of steps).
- Create our request, pedometerRequest, passing the values for isActive and previousCount and then build.
- For each remaining response of pedometerResponse print to the gui the results.

2.1 Server gRPC Method:

- Get the request values for IsActive and previousCount. Assigning previousCount int a current Count variable.
- Creating fake step counting data using a for loop to simulate the time in seconds.
- Build a string result with the current Steps and with the currentCount.
- Create the response passing in the result.
- responseObserver closes communicated with onCompleted().

2.3 Dog's temperature

Unary Streaming gRPC: checkTemperature(TemperatureRequest) returns (TemperatureResponse)

2.3 Client:

- Synchronous Blocking stub healthService is created.
- Input from gui for bodyTemp.
- Create the temperatureRequest and pass the value for bodyTemp and build.
- Create the temperatureResponse and call the gRPC method passing the temperatureRequest.
- Output the result to gui.

2.3 Server gRPC Method:

- CurrentStatus request is retrieved, all values are read assigned to local variables.
- String result concatenates the values of wearing, heartBeatBPM and thermometerBodyTemp
- WearingCollarResponse is created, the result is passed to the Obj.
- responseObserver sends back the response .onNext()
- responseObserver closes communicated with onCompleted()

3 Dog Collar Service:

Client gui Filepath: src/main/java/com.github/clientgui/CollarServiceGUI.java

Server Impl Filepath: src/main/java/com.github/collarservice.grpc/server/CollarServiceImpl.java

3.1 Dog come home

Client Streaming gRPC: streamVoice(stream StreamVoiceRequest) returns (StreamVoiceResponse)

3.1 Client:

- Asynchronous stub asyncClient is created.
- StreamObserver of StreamVoiceRequest uses the stub to invoke the streamVoice gRPC. With a new StreamObserver we can get the response back.
- So have message1 message2 and message3 inputted from the gui to stream to the server, using the onNext implemented abstract method we can achieve this.
- After all the messages are streamed onComplete() closes the gRPC connection.

3.1 Server gRPC Method:

- Creating a StreamObserver of StreamVoiceRequest the abstract methods onNext and on-Completed are implemented.
- onNext concatenates all the streamedVoiceRequest messages together ready to send back to the client one single response in a string result.
- onCompleted creates the StreamVoiceResponse and completes the sending of the return message.

3.2 Turn on LED light and alert owner its dark outside

Client Streaming gRPC: collarLight(LedLightRequest) returns

(stream LedLightResponse)

2.2 Client:

- Synchronous Blocking stub collarService is created.
- Inputs from gui for isActive(is the light ready and active), battery(the battery percentage of the light).
- Create our request, ledLightRequest, passing the values for isActive and battery and then build.
- For each remaining response of ledLightResponse print to the gui the results.

2.1 Server gRPC Method:

- Get the request values for isActive and battery assigning to variables.
- For 10 seconds lets create some fake lux values for the brightness of the light outside.
- Concatenate into a string result the lux value and the battery percentage
- Create the ledLightResponse passing in the result.
- responseObserver closes communicated with onCompleted().

Naming Services:

JmDNS is used for service registration and discovery in the project.

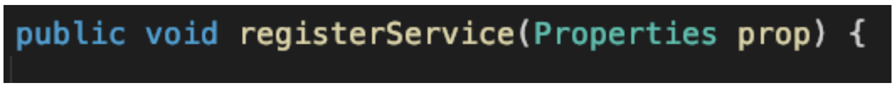
Service Registration

In my project each gRPC service has its own server, DogTrackingServerJmDNS.java, CollarServiceServerJmDNS.java and HealthServiceServerJmDNS.java. Each of these servers include methods to register the service using jmDNS. The method getProperties(Fig3) parses information found in separate properties files (*src/main/properties*)and returns a Properties Obj.

```
public Properties getProperties() {
```

Fig3: properties method found in every server class

The method registerService(*Fig4*) takes the values for service_type, service_name, service_port and service_desc from the properties Obj and passes them into a new object ServiceInfo.



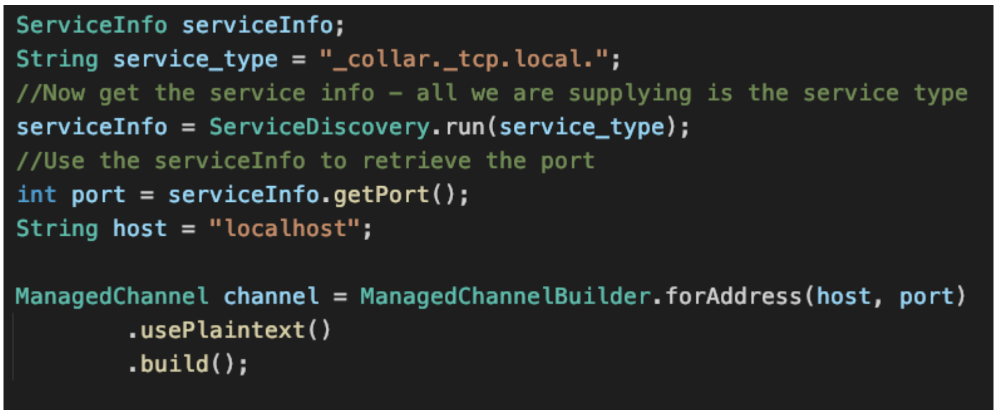
```java
public void registerService(Properties prop) {
```

Fig4: registerService method found in every server class

Once the ServiceInfo is passed then a jmDNS object can then register the service with the values specified. The service is now fully registered and accessible just by knowing the service_-type.

Service Discovery

Service discovery is achieved in a java file (*scr/main/java/com.github/jmdns/ServiceDiscovery.-java*) This class has methods to retrieve the information about the registered service, including the port, service type, service description and the service name. The ServiceListener allows implementation of the abstract methods, so we can add the service, remove the service and resolve the service. All that is left to do is add code to each of our client entry point implementations. By creating a new serviceInfo Obj and specifying the service type, we can return the serviceInfo of our desired service (*Fig5*).



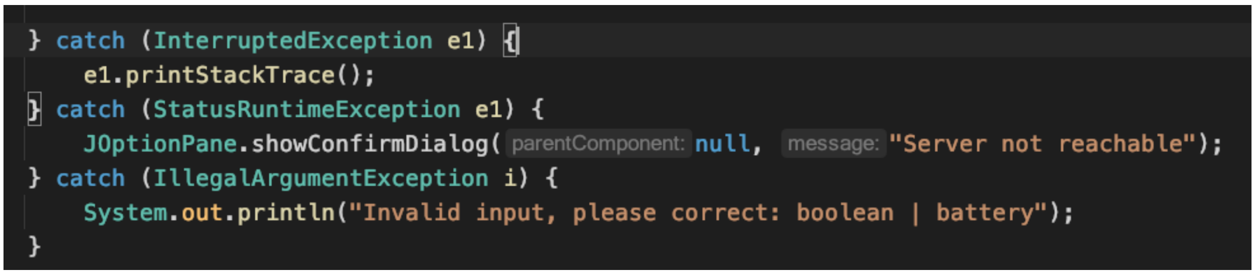
```java
ServiceInfo serviceInfo;
String service_type = "_collar._tcp.local.";
//Now get the service info - all we are supplying is the service type
serviceInfo = ServiceDiscovery.run(service_type);
//Use the serviceInfo to retrieve the port
int port = serviceInfo.getPort();
String host = "localhost";

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

Fig5 Code Snippet for client side calling of service discovery

Once we are able to get the port number of the specific service and we can pass the ip address (for the purposes of this project localhost will be sufficient) and finally build our channel! Now if the server is up and running, the client method can be called and the service will be added and resolved.

Remote Error Handling:

An example of error handling in the project is in CollarServiceGUI.java. In the server streaming method where we activate the LED light, when trying to build our request if we pass an invalid argument, the exception is caught. Similarly if the request is sent but there is no response from the server incoming that is handled too (*Fig6*).

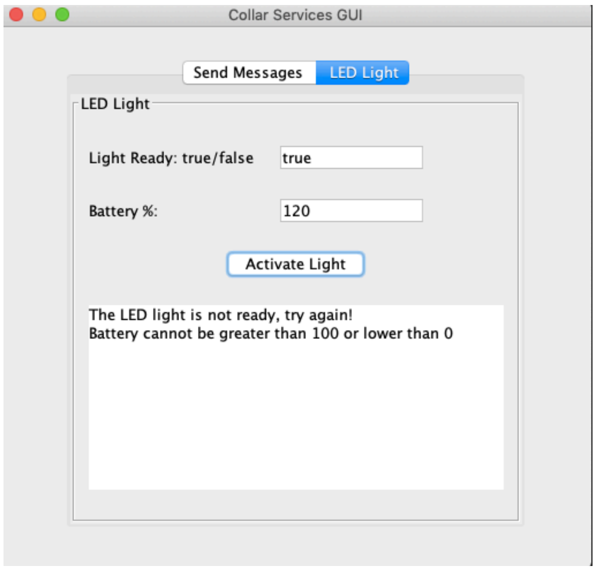


```
} catch (InterruptedException e1) {
    e1.printStackTrace();
} catch (StatusRuntimeException e1) {
    JOptionPane.showConfirmDialog( parentComponent: null,  message: "Server not reachable");
} catch (IllegalArgumentException i) {
    System.out.println("Invalid input, please correct: boolean | battery");
}
```

Fig6 CollarServiceGUI.java Errors are handled

User Input Validation

An example of user input validation. If the battery % passed cannot be lower than 0% or higher than 100%. If it is, a message will be shown to the user *Fig7*



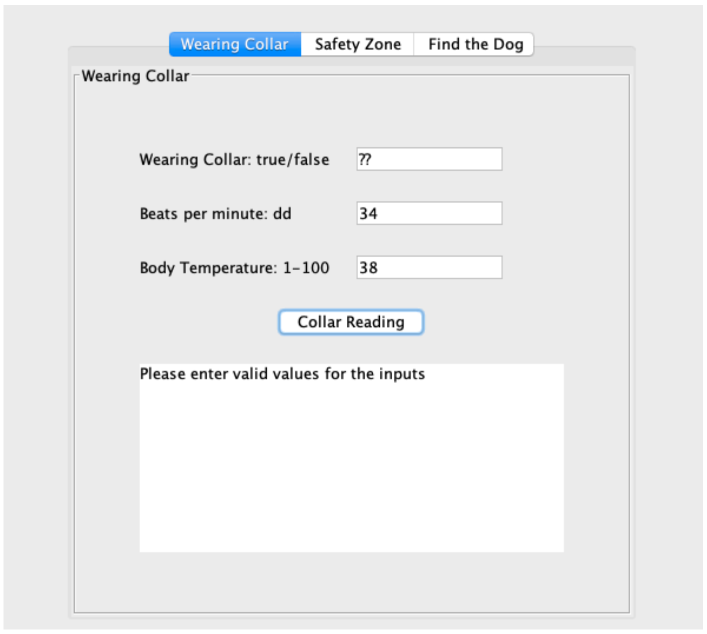
Light Ready: true/false: true

Battery %: 120

```
The LED light is not ready, try again!
Battery cannot be greater than 100 or lower than 0
```

Fig7 CollarService GUI -User Input Validation

Similarly when user input of the type is not recognised the Gui will tell the user *Fig8*.



Wearing Collar: true/false ??

Beats per minute: dd 34

Body Temperature: 1-100 38

Please enter valid values for the inputs

Fig8 Wearing Collar GUI - User Input Validation

There are many more cases of user input error handling within the project.

Client Graphical User Interface (GUI):

1 DogTracker GUI: *src/main/java/com.github/clientgui/DogTrackerGUI.java*

1.1 Is the dog wearing the collar - (collar status).

Input:

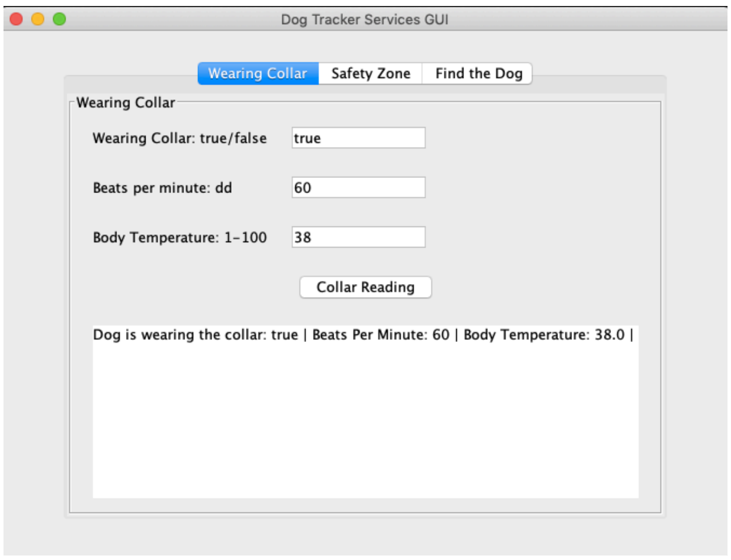
• Wearing Collar:	true
• Beats per minute:	60
• Body Temperature:	38

Expected output:

- Dog is wearing the collar: true | Beats Per Minute: 60 | Body Temperature: 38.0 |

Result:

- Functioning as expected(*Fig9*)



Wearing Collar

Wearing Collar: true/false: true

Beats per minute: dd: 60

Body Temperature: 1-100: 38

```
Dog is wearing the collar: true | Beats Per Minute: 60 | Body Temperature: 38.0 |
```

Fig9 Wearing Collar method - Input Output

1.2 Alerts if the dog strays out of pre-defined safe zone.

Input:

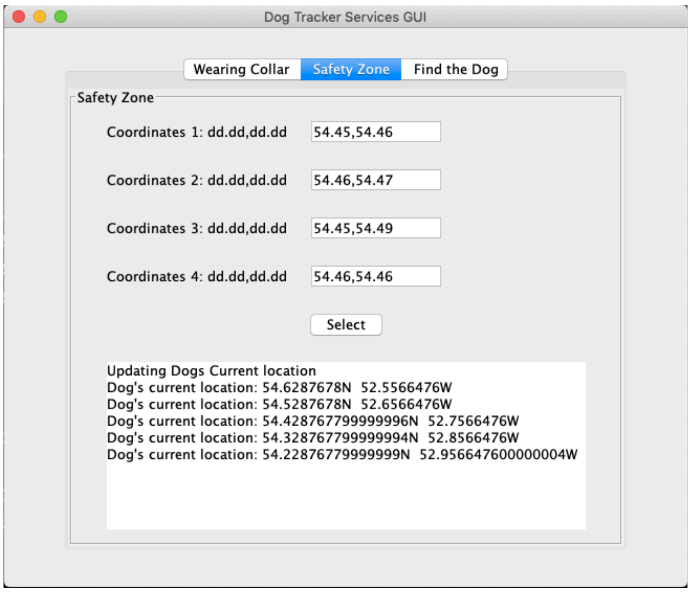
- Coordinates 1:   54.45,54.46
- Coordinates 2:   54.46,54.47
- Coordinates 3:   54.45,54.49
- Coordinates 4:   54.46,54.46

Expected output:

- Updating Dogs Current location
- Dog's current location: 54.6287678N 52.5566476W
- Dog's current location: 54.5287678N 52.6566476W
- Dog's current location: 54.428767799999996N 52.7566476W
- Dog's current location: 54.328767799999994N 52.8566476W
- Dog's current location: 54.22876779999999N 52.956647600000004W

Result:

- Functioning as expected(*Fig10*)



Coordinates 1: dd.dd,dd.dd 54.45,54.46

Coordinates 2: dd.dd,dd.dd 54.46,54.47

Coordinates 3: dd.dd,dd.dd 54.45,54.49

Coordinates 4: dd.dd,dd.dd 54.46,54.46

```
Updating Dogs Current location
Dog's current location: 54.6287678N  52.5566476W
Dog's current location: 54.5287678N  52.6566476W
Dog's current location: 54.428767799999996N  52.7566476W
Dog's current location: 54.328767799999994N  52.8566476W
Dog's current location: 54.22876779999999N  52.956647600000004W
```

Fig10 Safety Zone method - Input Output

1.3 Find the dog.

Input:

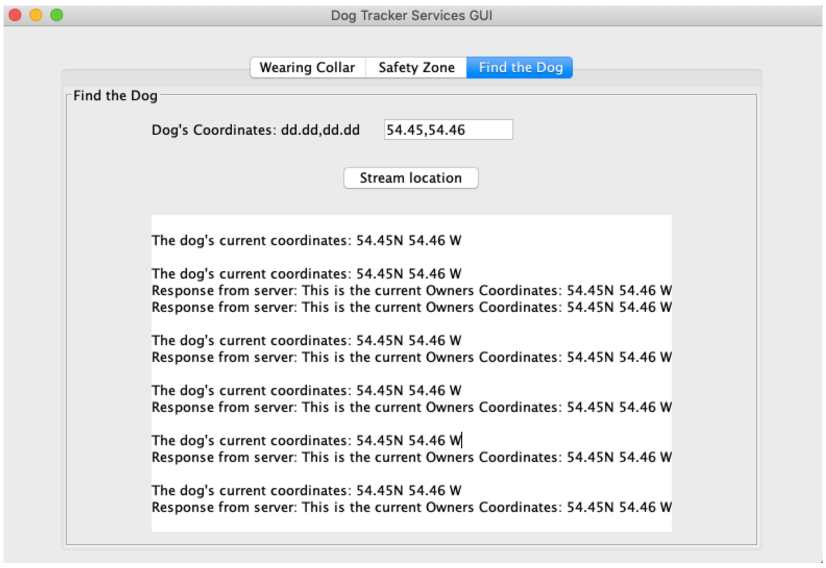
- Dog's Coordinates: 54.45,54.46

Expected output:

- The dog's current coordinates: 54.45N 54.46 W
- The dog's current coordinates: 54.45N 54.46 W
- Response from server: This is the current Owners Coordinates: 54.45N 54.46 W
- Response from server: This is the current Owners Coordinates: 54.45N 54.46 W
- The dog's current coordinates: 54.45N 54.46 W
- Response from server: This is the current Owners Coordinates: 54.45N 54.46 W
- The dog's current coordinates: 54.45N 54.46 W
- Response from server: This is the current Owners Coordinates: 54.45N 54.46 W
- The dog's current coordinates: 54.45N 54.46 W
- Response from server: This is the current Owners Coordinates: 54.45N 54.46 W
- The dog's current coordinates: 54.45N 54.46 W
- Response from server: This is the current Owners Coordinates: 54.45N 54.46 W

Result:

- This is a Bi-Directional Streaming method so the dog's current coordinates and the current owner's coordinates can vary on execution.
- Functioning as expected(*Fig11*)



Find the Dog

Dog's Coordinates: dd.dd,dd.dd 54.45,54.46

```
The dog's current coordinates: 54.45N 54.46 W

The dog's current coordinates: 54.45N 54.46 W
Response from server: This is the current Owners Coordinates: 54.45N 54.46 W
Response from server: This is the current Owners Coordinates: 54.45N 54.46 W

The dog's current coordinates: 54.45N 54.46 W
Response from server: This is the current Owners Coordinates: 54.45N 54.46 W

The dog's current coordinates: 54.45N 54.46 W
Response from server: This is the current Owners Coordinates: 54.45N 54.46 W

The dog's current coordinates: 54.45N 54.46 W
Response from server: This is the current Owners Coordinates: 54.45N 54.46 W

The dog's current coordinates: 54.45N 54.46 W
Response from server: This is the current Owners Coordinates: 54.45N 54.46 W
```

Fig11 Find the Dog method - Input Output

2 HealthService GUI: *src/main/java/com.github/clientgui/HealthServiceGUI.java*

2.1 Log the dogs heart-rate in bpm(beat per minute)

Input:

• Heartbeat Sensor Ready:	true
• Previous Usage:	20.40

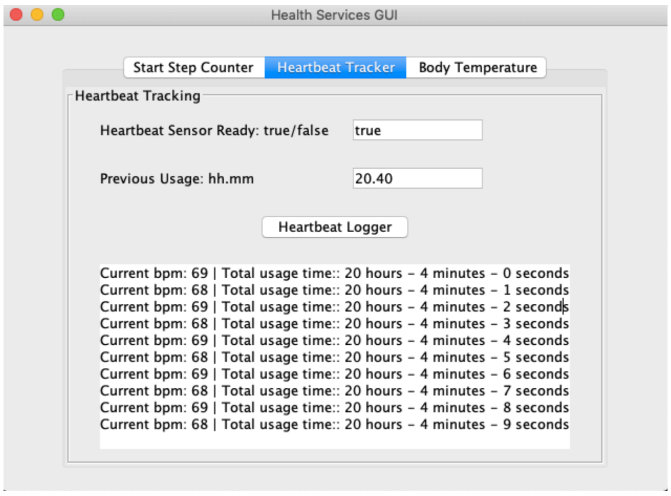
Expected output:

- Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 0 seconds
- Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 1 seconds
- Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 2 seconds

- Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 3 seconds
- Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 4 seconds
- Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 5 seconds
- Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 6 seconds
- Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 7 seconds
- Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 8 seconds
- Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 9 seconds

Result:

- Functioning as expected(*Fig12*)



Heartbeat Sensor Ready: true/false: true

Previous Usage: hh.mm: 20.40

```
Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 0 seconds
Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 1 seconds
Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 2 seconds
Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 3 seconds
Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 4 seconds
Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 5 seconds
Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 6 seconds
Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 7 seconds
Current bpm: 69 | Total usage time:: 20 hours - 4 minutes - 8 seconds
Current bpm: 68 | Total usage time:: 20 hours - 4 minutes - 9 seconds
```

Fig12 Find the Dog method - Input Output

2.2 Dog step counter

Input:

- Step Counter Active: true
- Previous Steps: 3500

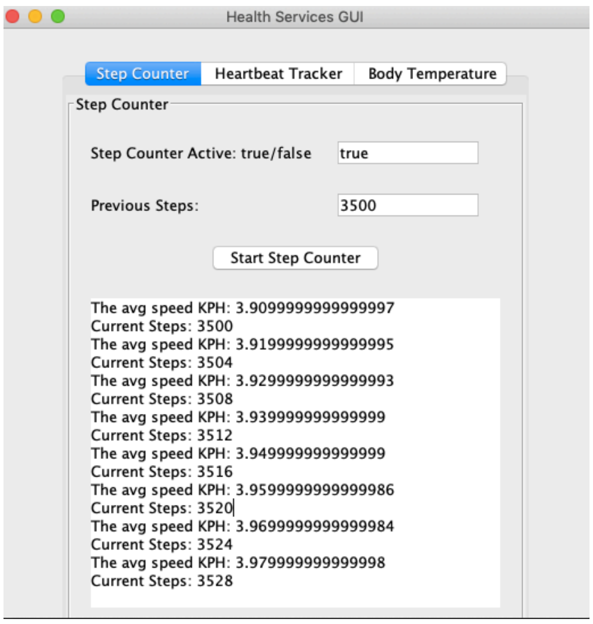
Expected output:

- The avg speed KPH: 3.9099999999999997
- Current Steps: 3500

- The avg speed KPH: 3.9199999999999995
- Current Steps: 3504
- The avg speed KPH: 3.9299999999999993
- Current Steps: 3508
- The avg speed KPH: 3.939999999999999
- Current Steps: 3512
- The avg speed KPH: 3.949999999999999
- Current Steps: 3516
- The avg speed KPH: 3.9599999999999986
- Current Steps: 3520
- The avg speed KPH: 3.9699999999999984
- Current Steps: 3524
- The avg speed KPH: 3.979999999999998
- Current Steps: 3528

Result:

- Functioning as expected(*Fig13*)



Step Counter Active: true/false: true

Previous Steps: 3500

```
The avg speed KPH: 3.9099999999999997
Current Steps: 3500
The avg speed KPH: 3.9199999999999995
Current Steps: 3504
The avg speed KPH: 3.9299999999999993
Current Steps: 3508
The avg speed KPH: 3.939999999999999
Current Steps: 3512
The avg speed KPH: 3.949999999999999
Current Steps: 3516
The avg speed KPH: 3.9599999999999986
Current Steps: 3520
The avg speed KPH: 3.9699999999999984
Current Steps: 3524
The avg speed KPH: 3.979999999999998
Current Steps: 3528
```

Fig13 Step Counter method - Input Output

2.3 Dog's temperature

Input:

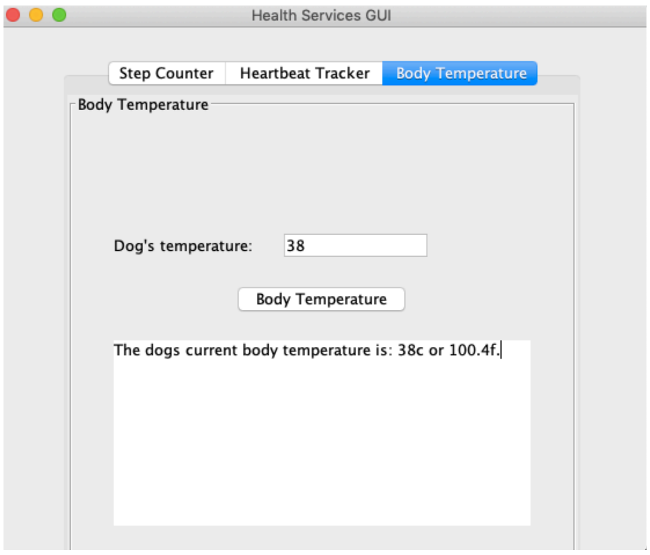
- Dog's Temperature: 38

Expected output:

- The dogs current body temperature is: 38c or 100.4f.

Result:

- Functioning as expected(*Fig14*)



Dog's temperature: 38

```
The dogs current body temperature is: 38c or 100.4f.
```

Fig14 Body Temperature method - Input Output

3 CollarService GUI: *src/main/java/com.github/clientgui/CollarServiceGUI.java*

3.1 Dog come home Input:

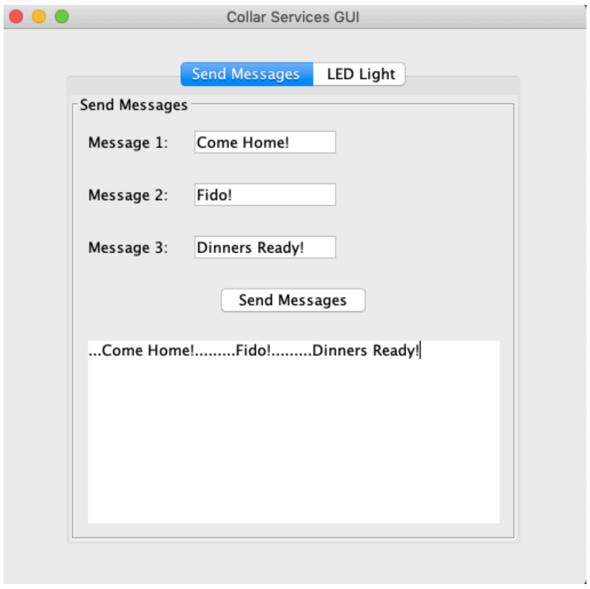
- Message 1: Come Home!
- Message 2: Fido!
- Message 3: Dinners Ready!

Expected output:

- ...Come Home!.........Fido!.........Dinners Ready!

Result:

- Functioning as expected(*Fig15*)



Message 1: Come Home!
Message 2: Fido!
Message 3: Dinners Ready!

```
...Come Home!.........Fido!.........Dinners Ready!
```

Fig15 Send Messages method - Input Output

3.2 Turn on LED light and alert owner its dark outside

Input:

- Light Ready: true
- Battery %:   90

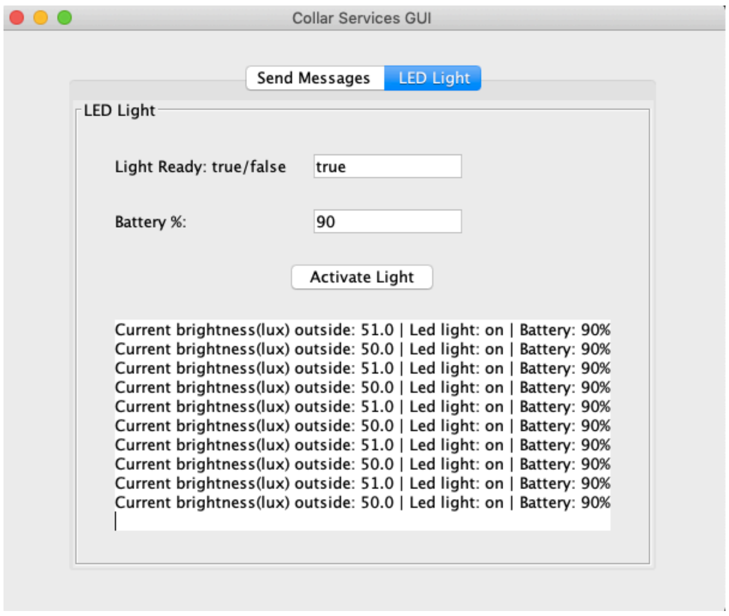
Expected output:

- Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
- Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
- Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
- Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
- Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%

- Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
- Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
- Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
- Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
- Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%

Result:

- Functioning as expected(*Fig16*)



Light Ready: true/false: true

Battery %: 90

```
Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 51.0 | Led light: on | Battery: 90%
Current brightness(lux) outside: 50.0 | Led light: on | Battery: 90%
```

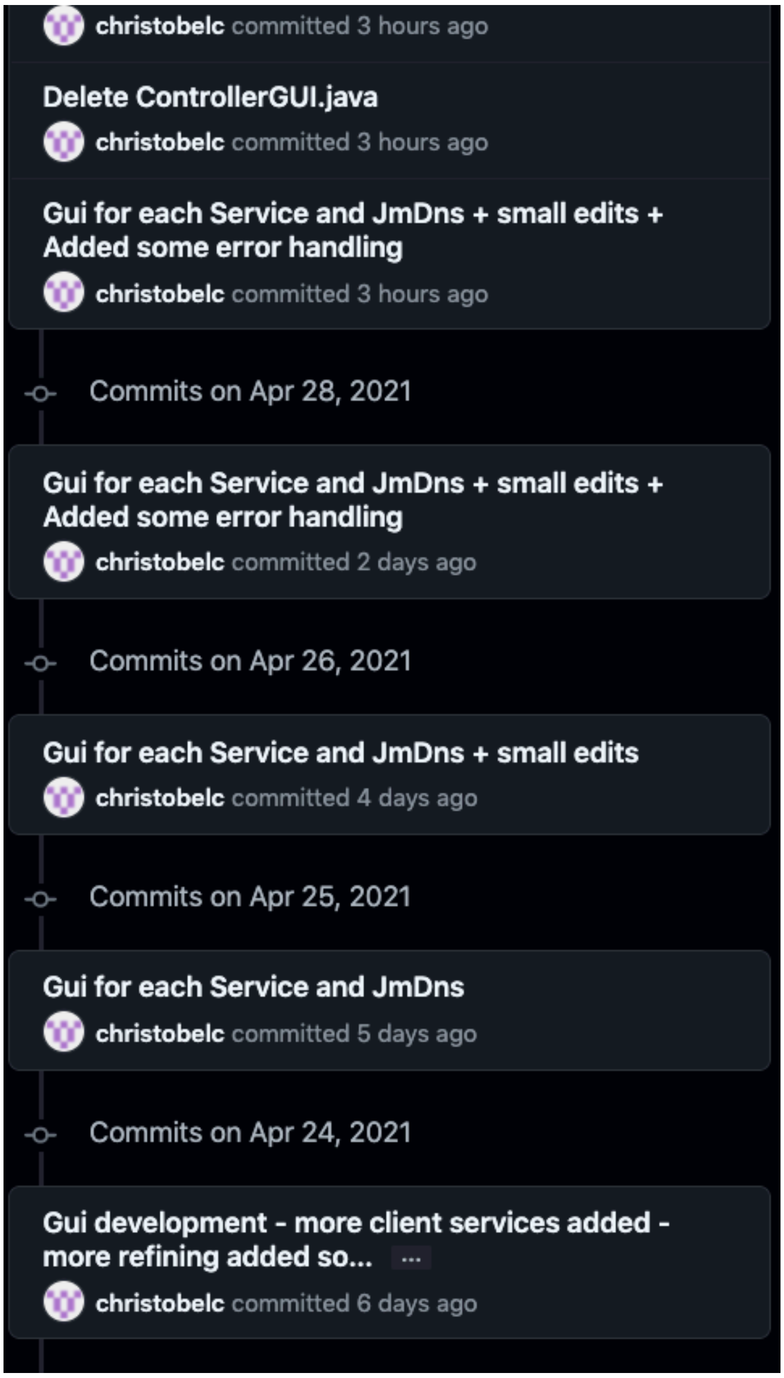
Fig16 LED Light method - Input Output

GUI Design:

The gui was created by using the sample gui as a template and also using some tutorials(*ref2*)

Github: *https://github.com/christobelc/dogcollar-grpc/tree/master/src/main*

Commits (as of writing): Fig17, Fig18



christobelc committed 3 hours ago

Delete ControllerGUI.java
christobelc committed 3 hours ago

Gui for each Service and JmDns + small edits + Added some error handling
christobelc committed 3 hours ago

Commits on Apr 28, 2021

Gui for each Service and JmDns + small edits + Added some error handling
christobelc committed 2 days ago

Commits on Apr 26, 2021

Gui for each Service and JmDns + small edits
christobelc committed 4 days ago

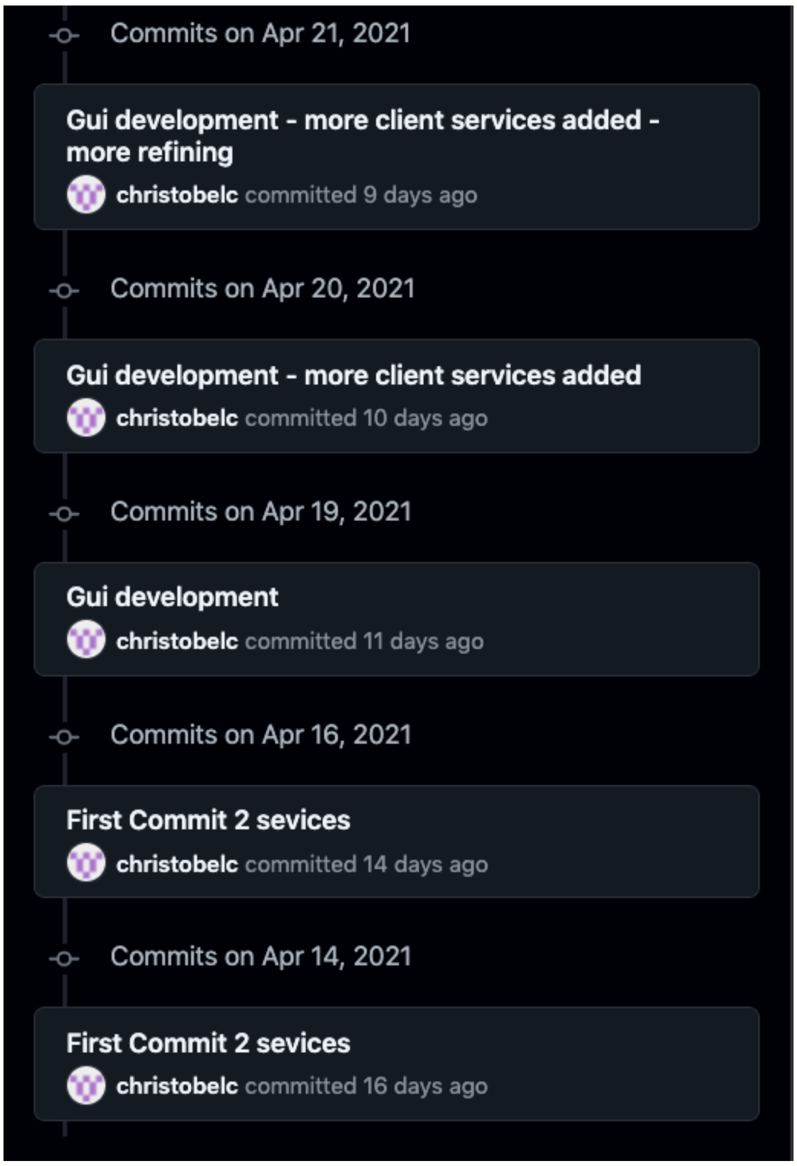
Commits on Apr 25, 2021

Gui for each Service and JmDns
christobelc committed 5 days ago

Commits on Apr 24, 2021

Gui development - more client services added - more refining added so... …
christobelc committed 6 days ago

Fig17 Github Commits


Commits on Apr 21, 2021

Gui development - more client services added - more refining
christobelc committed 9 days ago

Commits on Apr 20, 2021

Gui development - more client services added
christobelc committed 10 days ago

Commits on Apr 19, 2021

Gui development
christobelc committed 11 days ago

Commits on Apr 16, 2021

First Commit 2 sevices
christobelc committed 14 days ago

Commits on Apr 14, 2021

First Commit 2 sevices
christobelc committed 16 days ago

Fig18 Github Commits

Since the first commit there has been lots of edits on the Github repository. I have deleted and added some files. Also take note only the source files have only been uploaded, the automatically generated code such at the stub have files, build and configuration files have not been uploaded. Those files are included in the project build.

References:

1. Dogs Exercise: *https://www.thesprucepets.com/great-ways-to-exercise-with-dogs-1117865*
2. GUI Swing tutorial: *https://www.guru99.com/java-swing-gui.html*