Shane O'Neill

X15048209  Project report

DISTRIBUTED SYSTEMS

Table of Contents

[Use of JmDNS 2](#_Toc37004579)

[Use of gRPC: 2](#_Toc37004580)

[Proto Files 2](#_Toc37004581)

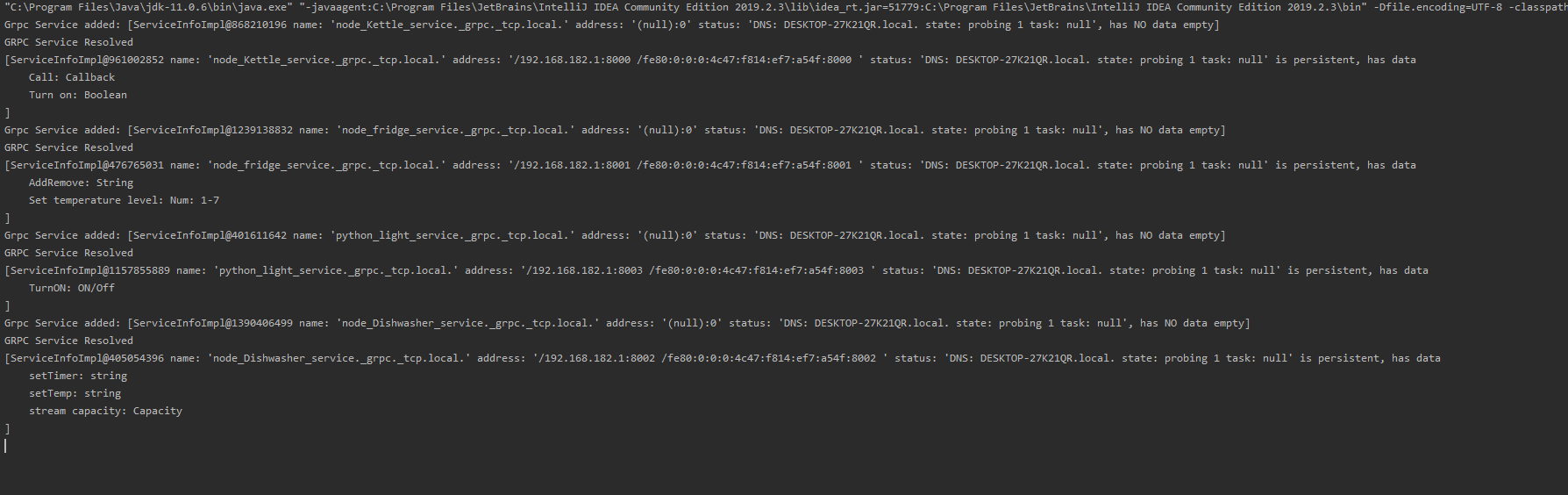
[RPC implementation - Services 7](#_Toc37004582)

[GUI 16](#_Toc37004583)

[GitHub Link: 16](#_Toc37004584)

# Use of JmDNS

Services are registered in JMDNS:



# Use of gRPC:

For my project I decided to create a smart kitchen using gRPC and smart devices. The devices I chose to develop were a light, kettle, fridge and dishwasher.

For each device I created a new proto file outlining the rpc’s to be consumed and the request and response types.

## Proto Files

Here you can see the light bulb proto that implements a simple RPC with a request and a response:

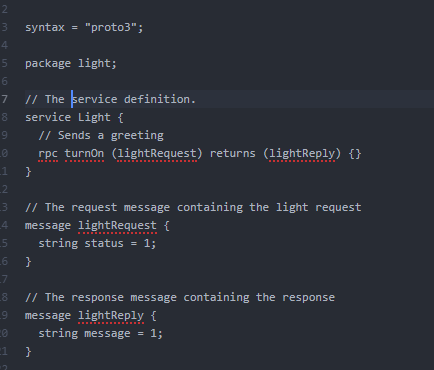


Fig.1 *light.proto code*

This proto allows a request to made to the light service to turn on and off the light.

Below is the Kettle proto file that implements server-side streaming:

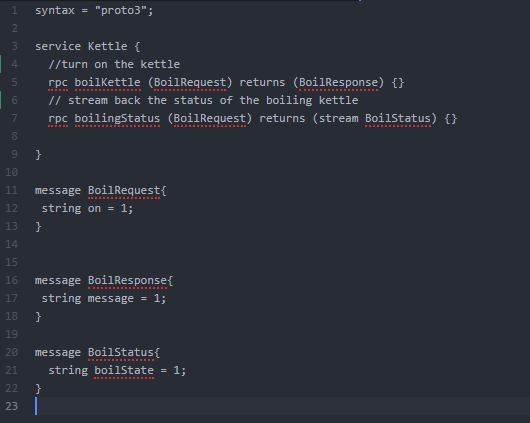


Fig.2 *Kettle proto code*

This proto allows the kettle to be turned on using a simple RPC and then the request for boiling status streams a response from the server back to the client.

This is the Proto file for the Dishwasher that implements client-side streaming:

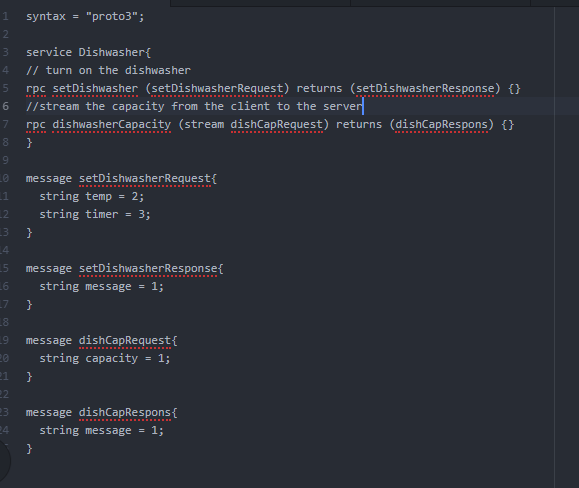


Fig.3 *Dishwasher proto code*

This proto file allows for a simple rpc to start the dishwaser with the temperature and timer – and the second rpc streams the capacity from the client to the server to show how full the dishwasher is.

The Final proto is the Fridge proto that implements bidirectional streaming:

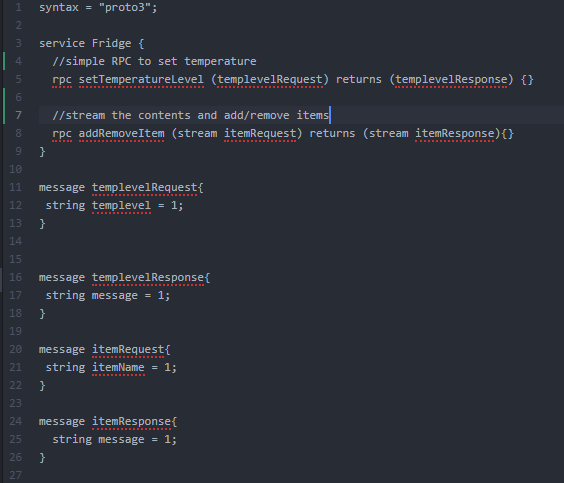


Fig.4 *Fridge proto code*

This proto allows for a simple RPC to set the temperature level in the Fridge (that dial in the fridge that you never use) and a bidirectional stream to allow you to add and remove items from the fridge and list the contents.

## RPC implementation - Services

**Light**

Client:

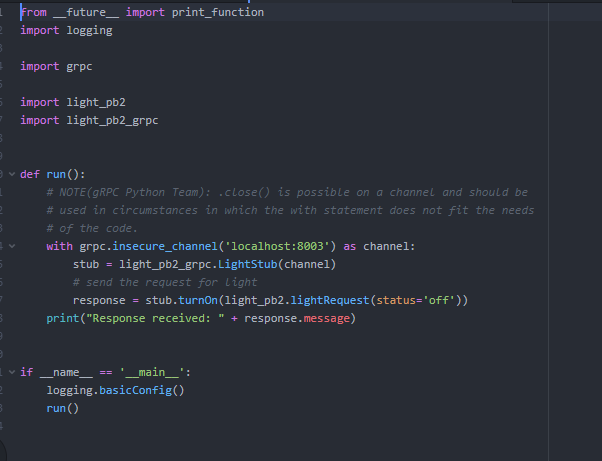


Fig.5 *light Client code*

Server:



Fig.6 *light server*

Functionality:

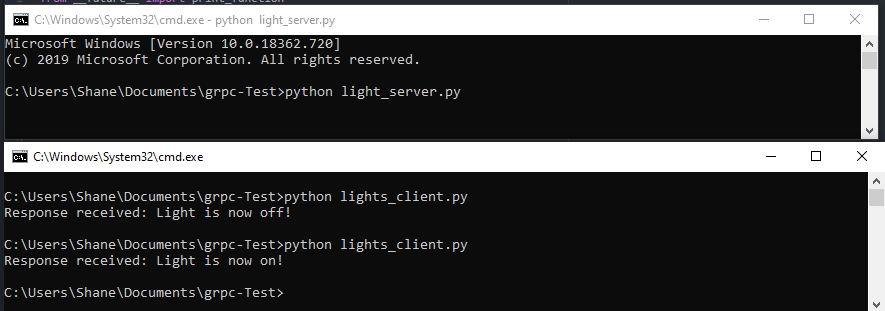


Fig.7 *terminals for Light*

The server allows a request to be made and sends back a message with the state of the light.

**Kettle:**

Client:

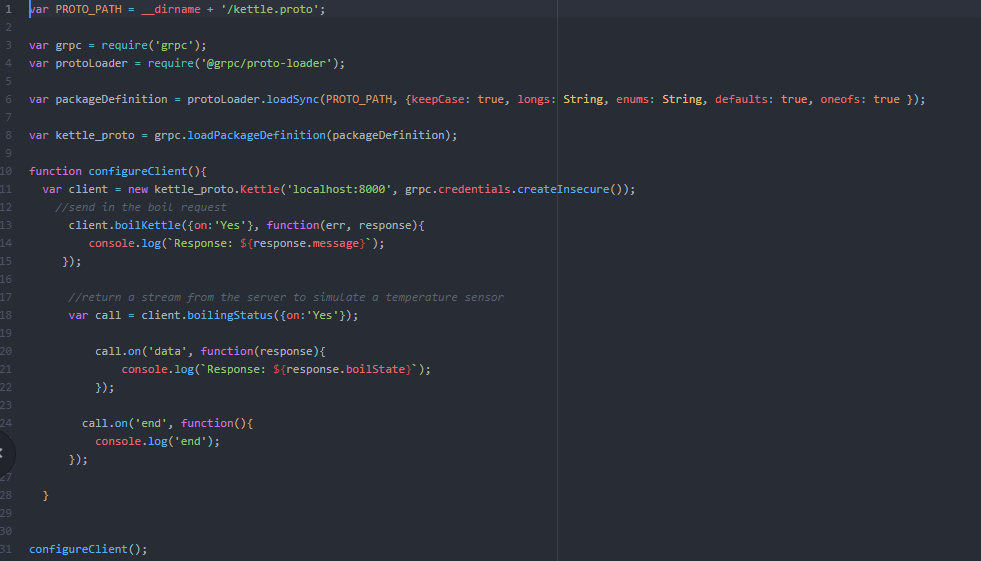


Fig.8 Kettle client

Server:

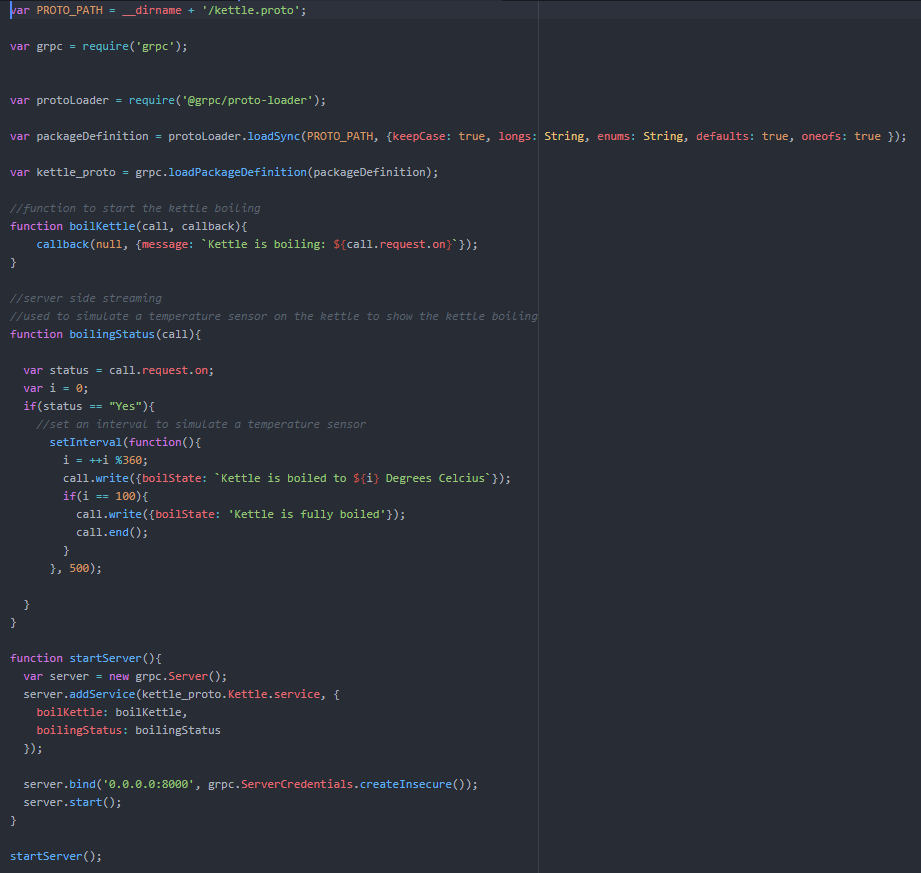
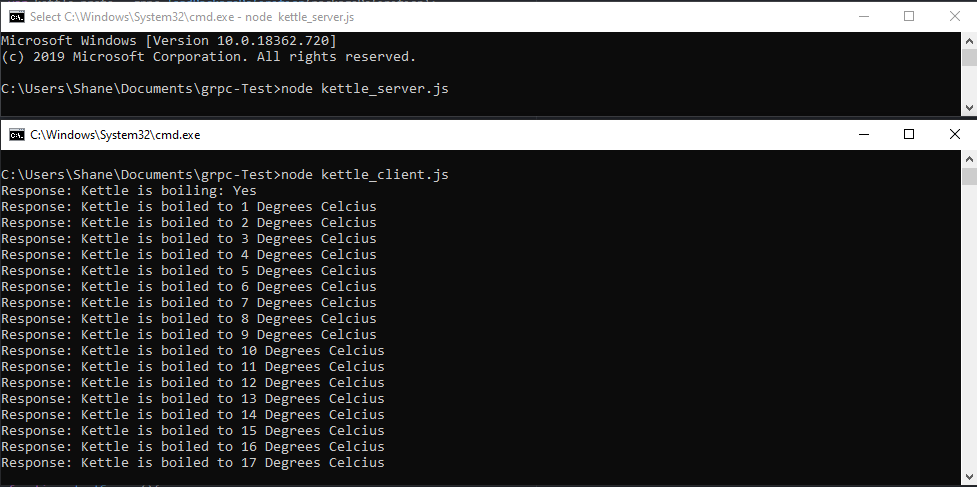


Fig.9 Kettle Server

Functionality:



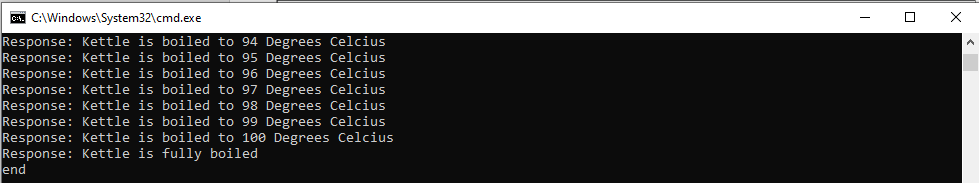


Fig.10 Kettle terminal

This appliance is turned on and a stream of responses is sent from the server to the client.

Using a timer interval to simulate a temperature gauge heating up the temperature is streamed back to the client.

Once the kettle is boiled to 100 the stream ends and the kettle is boiled.

**Dishwasher**

Client:

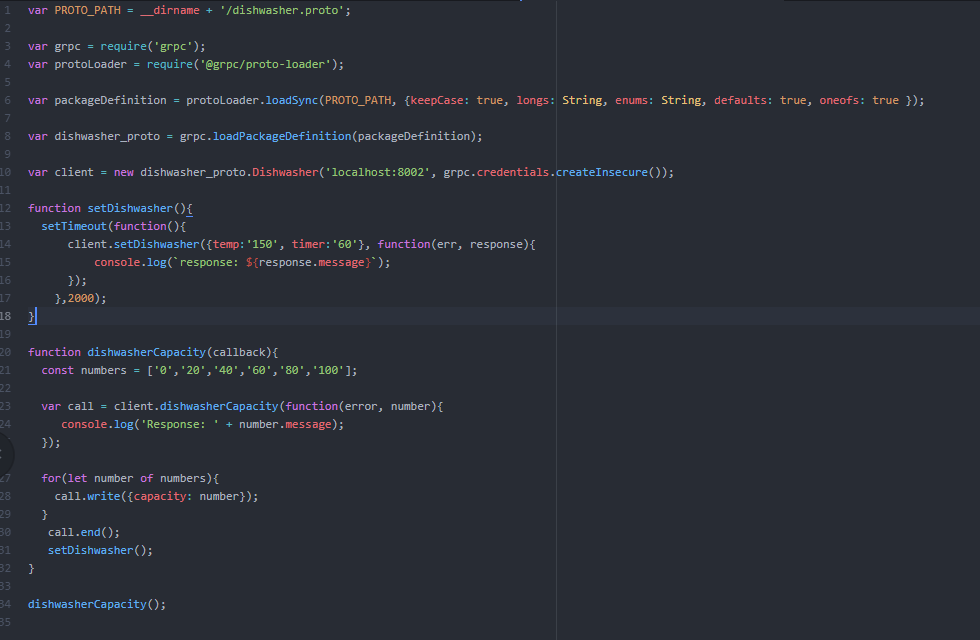


Fig.11 Dishwasher Client

Server:

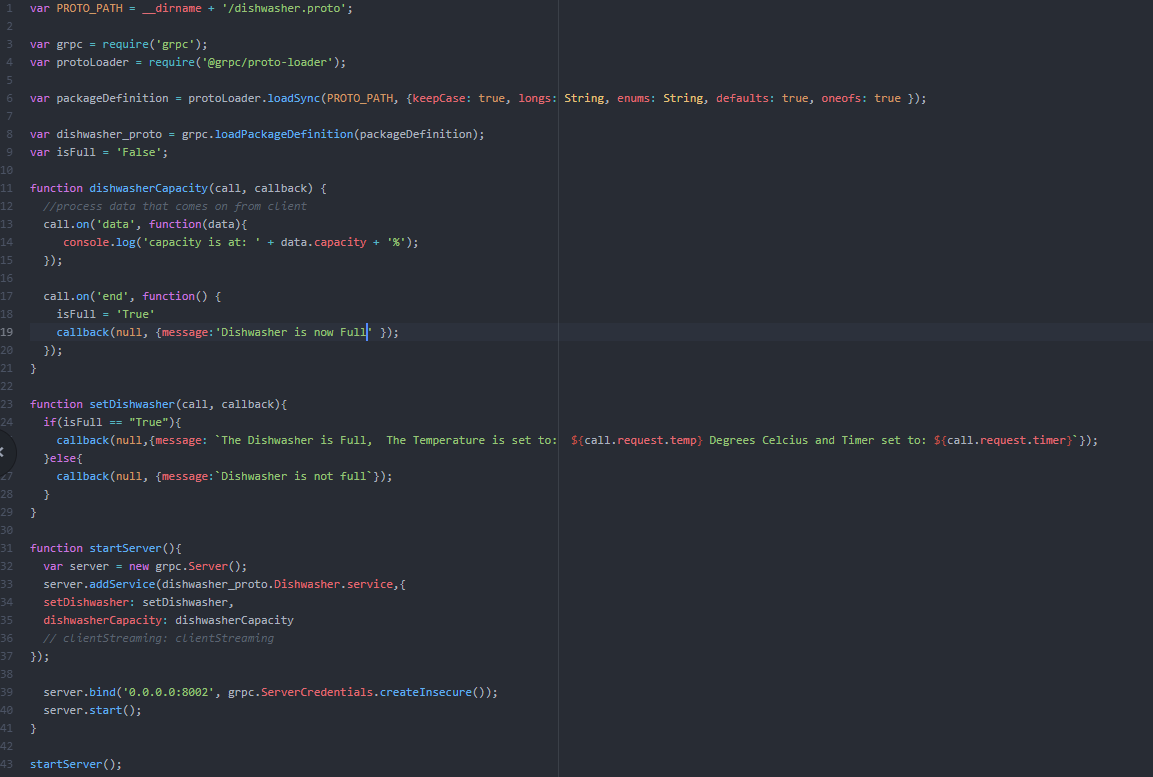


Fig.12 Dishwasher Server

Functionality:

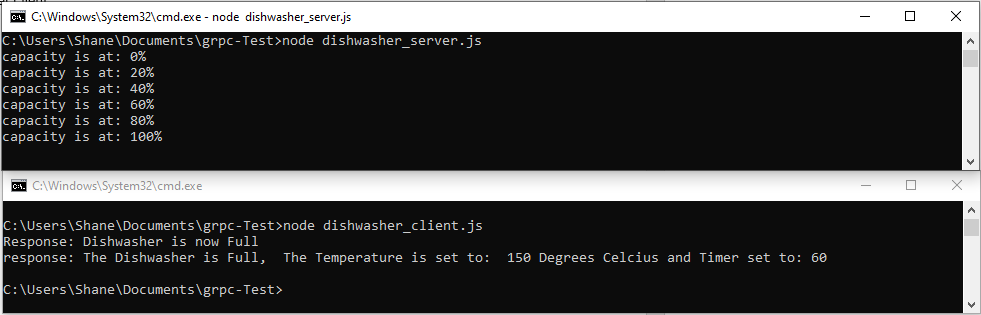


Fig.13 dishwasher terminal

The client Streams the capacity of the dishwasher to the server and then when its full it sets a flag to True:isFull

Then the client requests that the dishwasher starts for a certain time at a certain temperature.

**Fridge**

Client:

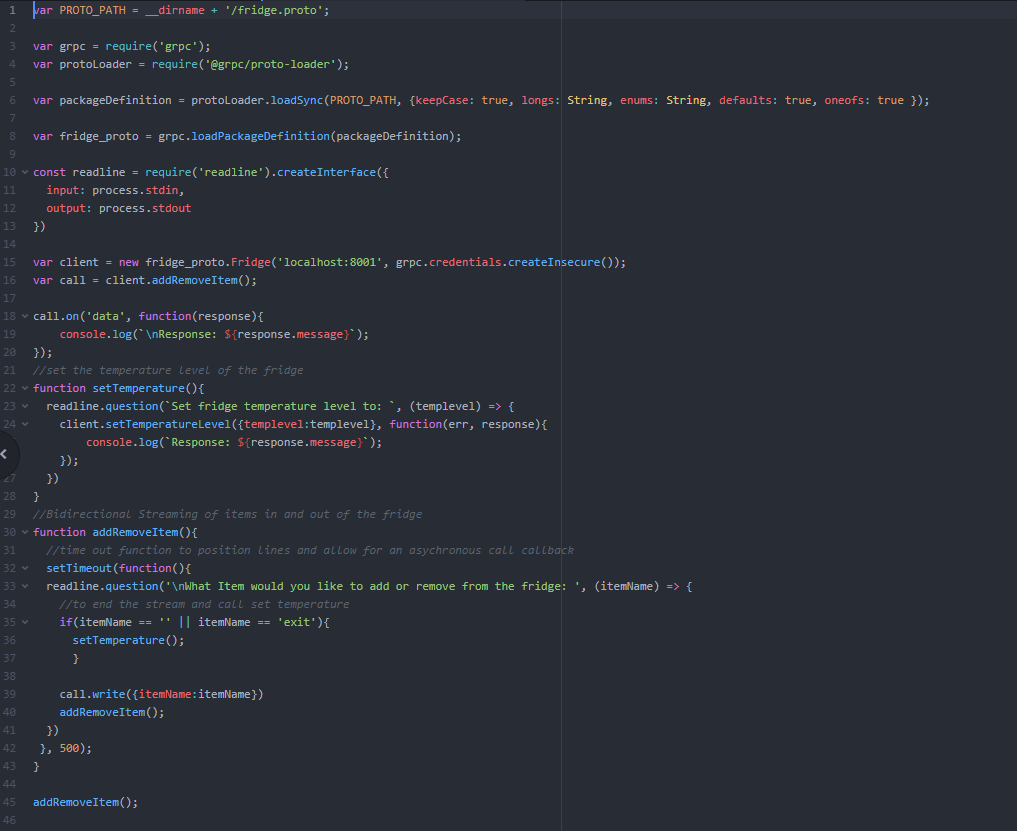
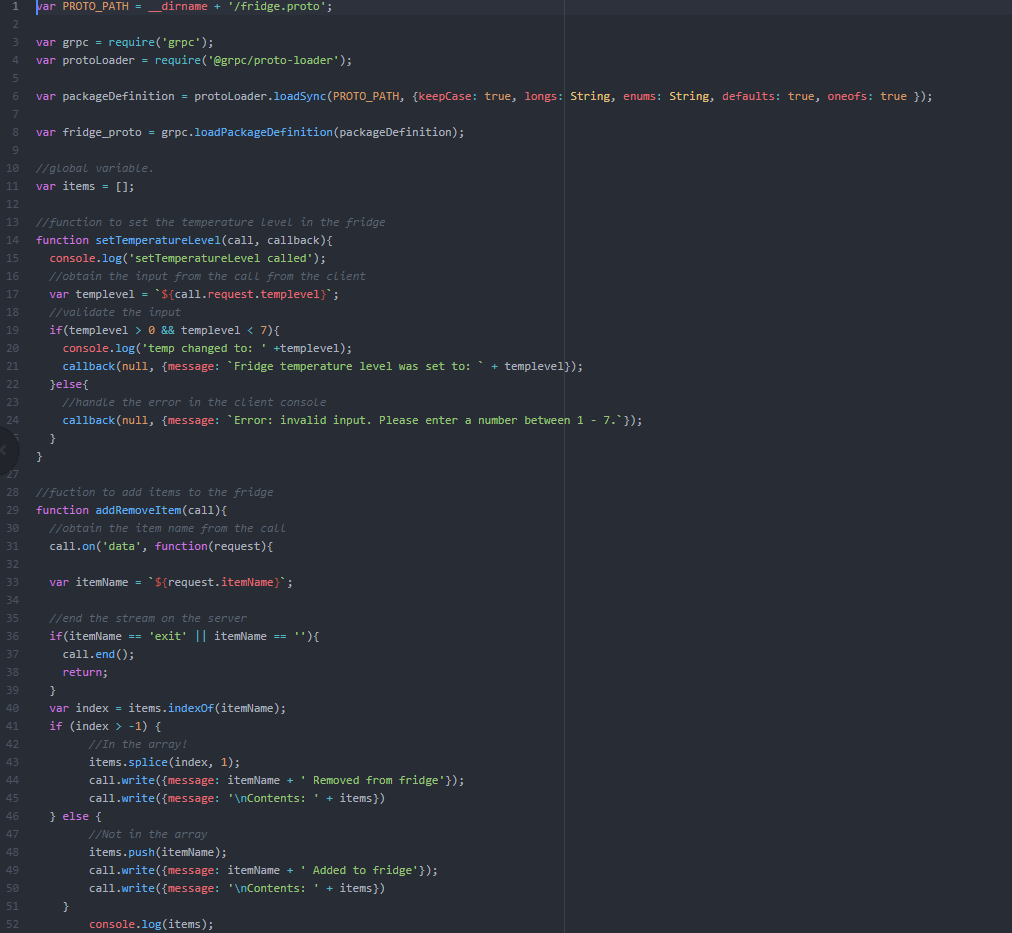


Fig.14 Fridge Client

Server:



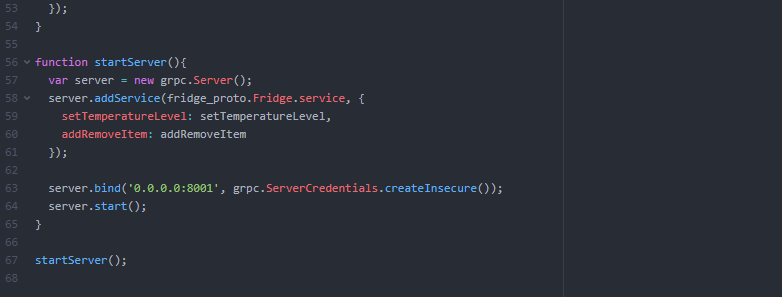


Fig.15 Fridge Server

Functionality:

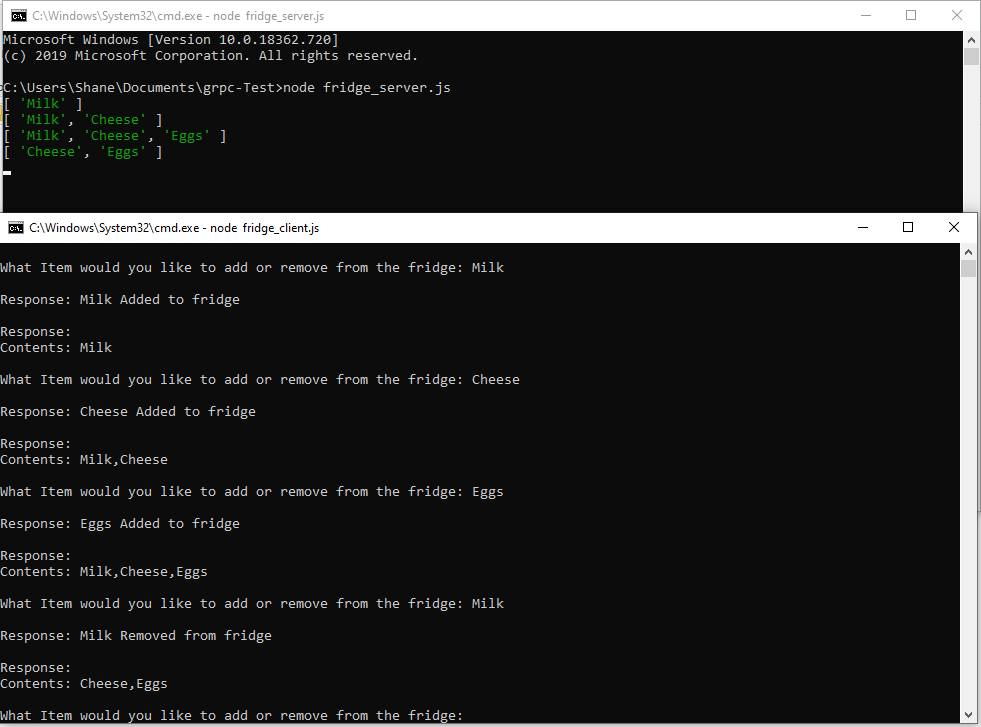


Fig.16 Fridge terminal – add/remove

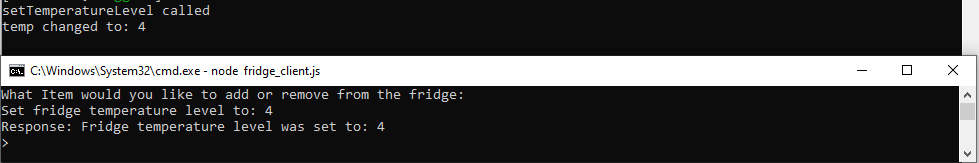


Fig.17 Fridge terminal – set temperature

The Server and client are streaming bidirectionally – the client asks the user to add/remove an item to the fridge.

The item is then streamed to the server and stored in an array that is streamed back to the client – which is then printed to the screen. If the user streams an item that is already in the fridge then it is removed from the array.

The items array is printed to the screen on the server as the items are added and removed.

If exit or an empty string is streamed, then the set temperature level function is called and the user can set the temperature of the fridge.

# GUI

A Very basic HTML Gui is shown here below listing the services and their ports:

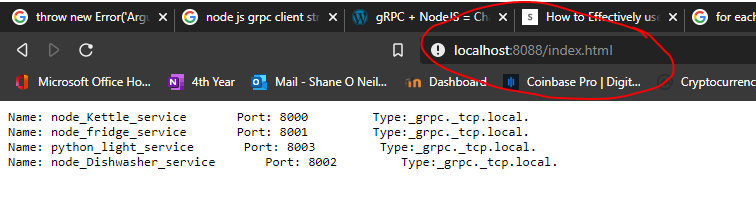


Fig.18 HTML GUI

# GitHub Link:

<https://github.com/JellieBeanz/grpc-DistSys>