

---

# SOFTWARE MAINTENANCE DOCUMENT

for

Encost Smart Graph Project

Version 1.0

Prepared by: Ben Lee  
SoftFlux Engineer

SoftFlux

June 11, 2023

# Contents

<b>1</b>	<b>Introduction/Purpose</b>	<b>3</b>
1.1	Purpose . . . . .	3
1.2	Document Conventions . . . . .	3
1.3	Intended Audience and Reading Suggestions . . . . .	3
1.3.1	Intended Audience . . . . .	3
1.3.2	Reading Suggestions . . . . .	3
1.4	Project Scope . . . . .	4
<b>2</b>	<b>Specialized Requirements Specification</b>	<b>4</b>
2.1	Classes . . . . .	4
2.1.1	DataSummary . . . . .	4
2.2	Requirements . . . . .	4
2.2.1	Calculate Device Distribution . . . . .	4
<b>3</b>	<b>Maintenance</b>	<b>4</b>
3.1	Product Backlog . . . . .	4
3.1.1	Categorising Users . . . . .	4
3.1.2	ESGP Account Login . . . . .	4
3.1.3	ESGP Feature Options . . . . .	4
3.1.4	Loading the Encost Smart Homes Dataset . . . . .	5
3.1.5	Categorising Smart Home Devices . . . . .	5
3.1.6	Building a Device Graph . . . . .	5
3.1.7	Graph Visualisation . . . . .	5
3.2	Modification - Updating GraphStream . . . . .	5
3.2.1	User Story . . . . .	5
3.2.2	Problem/modification request . . . . .	5
3.2.3	Problem/modification analysis . . . . .	5
3.2.4	Acceptance/rejection . . . . .	5
3.3	Modification - Calculating Device Distribution . . . . .	6
3.3.1	User Story . . . . .	6
3.3.2	Problem/modification request . . . . .	6
3.3.3	Problem/modification analysis . . . . .	6
3.3.4	Acceptance/rejection . . . . .	6
3.3.5	Modification implementation . . . . .	6
3.3.6	Maintenance review/acceptance . . . . .	8
3.4	Modification - Household Filter . . . . .	9
3.4.1	User Story . . . . .	9
3.4.2	Problem/modification request . . . . .	9
3.4.3	Problem/modification analysis . . . . .	9
3.4.4	Acceptance/rejection . . . . .	9
3.4.5	Modification implementation . . . . .	9
3.4.6	Maintenance review/acceptance . . . . .	12
<b>4</b>	<b>CI/CD Pipeline</b>	<b>13</b>
<b>5</b>	<b>Conclusion</b>	<b>14</b>

# Revision History

Name	Date	Reason for Changes	Version

## 1 Introduction/Purpose

### 1.1 Purpose

This Software Maintenance document’s purpose is to inform developers of any modifications that may need to be made to the system. Any modifications should be implemented with care, this time must be spent of program comprehension as well as understanding requirements and design philosophies stated in the SRS, SDS and functional software test plan documentation. This document will include Product Backlog and modification details and this will be written in a way to avoid any ambiguities.

### 1.2 Document Conventions

- ESGP: Encost Smart Graph Project
- CLI: Command Line Interface
- SRS: Software Requirement Specifications
- SDS: Software Design Specifications

### 1.3 Intended Audience and Reading Suggestions

#### 1.3.1 Intended Audience

- Product Owner: The product owner will update this document for any modifications that can be made to the system.
- Development Team: A team member will require this document to complete maintenance on the system.

#### 1.3.2 Reading Suggestions

All team members are expected to have read and studied the SRS, SDS, Test Planning and Development Planning documentation. All functional requirements are used in the software, all non-functional requirements must be considered and utilised when optimising the code. Any diagrams and design principles/patterns shown and SDS will be implemented in the Development Planning documentation. The Test Plan will be used to design all tests and used to create check functions that validate data. The Development Planning document is important for understanding the code for any and all maintenance purposes.

## 1.4 Project Scope

Encost uses the ESGP project to better understand how its devices are being used and linked in New Zealand households. The Encost Smart Homes Dataset, a dataset including details about 100 New Zealand homes and the Encost devices being used, will be able to be visualised and summarised through the ESGP project. The 100 New Zealand households that agreed to have their data collected will also receive the project so they may view a visualisation of the Encost Smart households Dataset.

# 2 Specialized Requirements Specification

## 2.1 Classes

### 2.1.1 DataSummary

The DataSummary class from the SDS should be implemented with some modifications, these include using hash maps to count types and categories of devices, adding a new function is to be added called printDeviceDistribution which will print the counters to the console app. DataSummary should also utilise the Singleton design pattern.

## 2.2 Requirements

### 2.2.1 Calculate Device Distribution

Calculating device distribution has been a requirement that has been changed from high priority to medium priority, however we will be locking it as a high priority requirement as Encost users were expecting this features, and as of the last release Encost users had no additional features that community users didn't have. The implications of this mean that this not being a feature in the last release of ESGP is a bug that needs to be fixed.

# 3 Maintenance

## 3.1 Product Backlog

### 3.1.1 Categorising Users

#### Description

Users should be allowed to specify whether they are Community or Encost users. Encost Users will need to login, then they will have access to more features than Community Users.

### 3.1.2 ESGP Account Login

#### Description

Employees of Encost should be able to log in to the system and access additional functionalities. For this purpose, a set of login credentials will be issued.

### 3.1.3 ESGP Feature Options

#### Description

Verified Users should be able to choose between “1” loading a custom dataset, “2” visualising a graph representation of the data, and “3” viewing summary statistics. Community Users should be able to choose between “1” loading a custom dataset.

### 3.1.4 Loading the Encost Smart Homes Dataset

#### Description

The Encost Smart Homes Dataset should be read and processed by the system.

### 3.1.5 Categorising Smart Home Devices

#### Description

Each Encost Smart Device should be able to be grouped into one of five categories by the system. These categories will be used to display graphs and summarise statistics.

### 3.1.6 Building a Device Graph

#### Description

To store all of the Encost Smart Device Objects, the system should develop a graph data structure. This graph will be used to display Encost Smart Devices and provide summary data. A device class will need to be created when creating the graph data type class.

### 3.1.7 Graph Visualisation

#### Description

The system should provide the user with a visual representation of the graph data structure.

## 3.2 Modification - Updating GraphStream

### 3.2.1 User Story

“The client (Encost) has decided that the system should use the most up-to-date version of GraphStream”

### 3.2.2 Problem/modification request

Investigate and update GraphStream to the latest version, modifying any code that is affected by the update.

### 3.2.3 Problem/modification analysis

**Maintenance Category:** Adaptive

**Impact Analysis**

- **Verify the problem:** GraphStream is not up to date
- **Implementation options:** Update GraphStream jar files/dependencies from 1.3 to the latest version(2.0).
- **Effort:** Alter lines of code affected by the update and learn about the changes from 1.3 to 2.0.
- **Resources:** Learning may take 10-15 minutes, while changing the code should only take 10 minutes. If coded according to the Test Plan the tests should only take 5 minutes.

### 3.2.4 Acceptance/rejection

Rejected. This Modification has been rejected as it is unnecessary to update as the Graph already has the necessary features.

### 3.3 Modification - Calculating Device Distribution

#### 3.3.1 User Story

“The Encost users have noticed that the summary statistics aren’t working. They would like to be able to see a textual summary of the device distribution (SRS 4.9 Calculating Device Distribution)”

#### 3.3.2 Problem/modification request

Identify the issue and implement Device Distribution for summary statistics.

#### 3.3.3 Problem/modification analysis

Maintenance Category: Corrective  
Impact Analysis

- **Verify the problem:** Device Distribution is not being calculated and displayed to the user.
- **Implementation options:** Add the DataSummary class according to the SDS with some modifications. Create a function for printing the device distribution statistics to the console. A test will also need to be added for calculating device distribution.
- **Effort:** An estimated 100-200 lines of code will be implemented. Time will be spent understanding code before inserting the function, and setting up the test.
- **Resources:** This could take an estimated 10 hours to implement.

#### 3.3.4 Acceptance/rejection

Accepted

#### 3.3.5 Modification implementation

Implementation includes, adding the DataSummary class to the system, adding the functions included in Figure 3.1 and calling the functions from the ConsoleApp class. In ConsoleApp where the users input is checked for encost users feature options, showSummary function should be invoked. showSummary should get an instance of DataSummary, pass the devices, then call calculateDeviceDistribution and printDeviceDistribution.

DataSummary
- instance:DataSummary - devices:Device[]
+ DataSummary() + getInstance():DataSummary + setDevices(Device[] devices) + clearDevices() + calculateDeviceDistriubution(Device[] devices): Map + printDeviceDistriubution(Map counters)

Figure 3.1: Data Summary Class

Figure 3.1 references the DataSummary class from the SDS document. It was been updated to suit the needs of the modifications and requirements. As of the publishing of this document only device distribution statistics are needed. Pseudocode will be given as the number of lines for these functions to far too large.

Pseudocode for DataSummary class:

---

**Algorithm 1** calculateDeviceDistribution pseudocode

---

Set up LinkedHashMaps counters for device categories and types  
Set up an array list for inserted device id's  
**for** Each device in the device list **do**  
    Get the device's id, category and type  
    **if** device id **or** category **or** type are empty **then**  
        Continue to the next device  
    **end if**  
    **if** Inserted device list contains device id **then**  
        Continue to the next device  
    **end if**  
    Add device's id to the inserted device list  
    **if** counter contains device category **then**  
        Add one to that counter  
    **end if**  
    **if** counter contains device type **then**  
        Add one to that counter  
    **end if**  
**end for**  
Return the counters

---

---

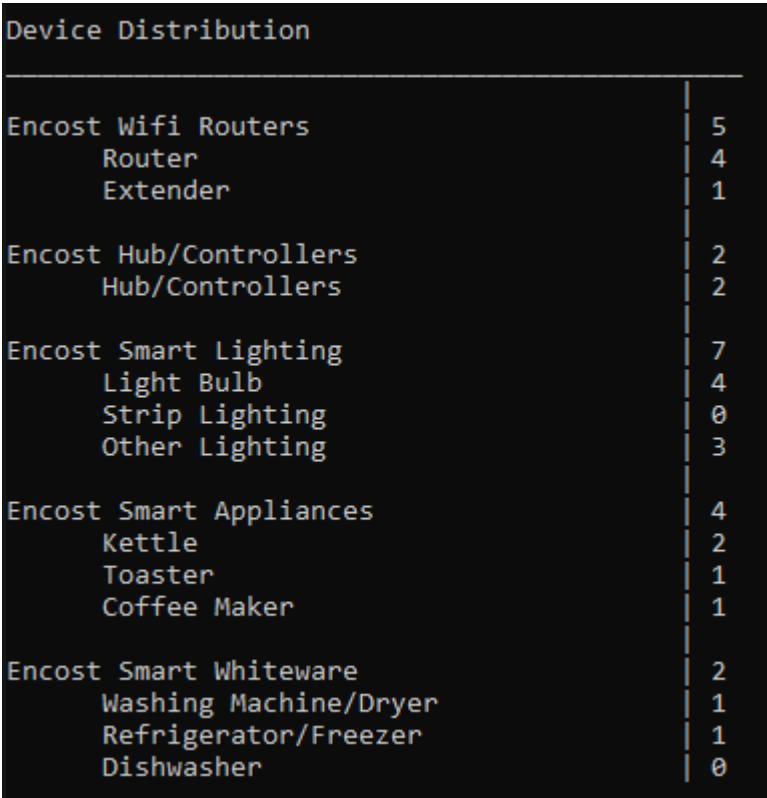
**Algorithm 2** printDeviceDistribution pseudocode

---

Print table title  
**for** each counter key in counters **do**  
    get counter's device and count  
    Switch for all possible device category or type  
    **if** category is not empty **then**  
        Print with format for category  
    **else**  
        Print with format for type  
    **end if**  
**end for**

---

Format for printing device distribution:



Device Distribution	
Encost Wifi Routers	5
Router	4
Extender	1
Encost Hub/Controllers	2
Hub/Controllers	2
Encost Smart Lighting	7
Light Bulb	4
Strip Lighting	0
Other Lighting	3
Encost Smart Appliances	4
Kettle	2
Toaster	1
Coffee Maker	1
Encost Smart Whiteware	2
Washing Machine/Dryer	1
Refrigerator/Freezer	1
Dishwasher	0

Figure 3.2: Device Distribution Print

### 3.3.6 Maintenance review/acceptance

Manual testing has been done, this was completed by comparing the output from printDeviceDistribution with the dataset values, see Figure 3.2. A test has been created to test the branch coverage and error checks of calculateDeviceDistribution.

```
/**
 * Tests the branch coverage of calculateDeviceDistribution function
 */
@Test
@DisplayName("test branch coverage for CalculateDeviceDistribution")
public void testCalculateDeviceDistribution() {
    DataSummary summary = DataSummary.getInstance();

    Device devicesExpected[] = {
        new Device(deviceID:"1", dateConnected:"23/01/23", deviceName:"1", deviceType:"Router",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"2", dateConnected:"23/01/23", deviceName:"2", deviceType:"Extender",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"3", dateConnected:"23/01/23", deviceName:"3", deviceType:"Light Bulb",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"4", dateConnected:"23/01/23", deviceName:"4", deviceType:"Strip Lighting",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"5", dateConnected:"23/01/23", deviceName:"5", deviceType:"Kettle",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"6", dateConnected:"23/01/23", deviceName:"6", deviceType:"Toaster",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false)
    };

    Device[] devicesBranch = {
        new Device(deviceID:"1", dateConnected:"23/01/23", deviceName:"1", deviceType:"Router",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"2", dateConnected:"23/01/23", deviceName:"2", deviceType:"Extender",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"3", dateConnected:"23/01/23", deviceName:"3", deviceType:"Light Bulb",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"4", dateConnected:"23/01/23", deviceName:"4", deviceType:"Strip Lighting",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"5", dateConnected:"23/01/23", deviceName:"5", deviceType:"Kettle",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"6", dateConnected:"23/01/23", deviceName:"6", deviceType:"Toaster",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"5", dateConnected:"23/01/23", deviceName:"5", deviceType:"Kettle",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false),
        new Device(deviceID:"", dateConnected:"23/01/23", deviceName:"", deviceType:"",
            householdID:"householdID", routerConnection:"routerConnection", sends:true, receives:false)
    };

    Map<String,Integer> countersExpected = new LinkedHashMap<>();
    Map<String,Integer> countersBranch = new LinkedHashMap<>();
    countersExpected = summary.calculateDeviceDistribution(devicesExpected);
    countersBranch = summary.calculateDeviceDistribution(devicesBranch);
    assertEquals(countersExpected, countersBranch);
}
```

Figure 3.3: Device Distribution Test



```
←[36m←[0m
←[36m+--←[0m ←[36mJUnit Jupiter←[0m ←[32m[OK]←[0m
←[36m'--←[0m ←[36mJUnit Vintage←[0m ←[32m[OK]←[0m
←[36m'--←[0m ←[36mESGPTTest←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mloadingEncostDatasetEmptyStringFilePath←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mbuildGraphValidFileA←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mbuildGraphValidFileB←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mloadingEncostDatasetInvalidFilePath←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mbuildGraphEmptyStringFilePath←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mcategoriseDevicesEncostDataset←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mtestCalculateDeviceDistribution←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mloadingEncostDatasetValidFilePath←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34maccountLoginValidUsernameValidPassword←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34maccountLoginValidUsernameInvalidPassword←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34maccountLoginInvalidUsernameValidPassword←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mloadingEncostDatasetNullStringFilePath←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34mcategoriseDevicesInvalidFileFormat←[0m ←[32m[OK]←[0m
←[36m +--←[0m ←[34maccountLoginInvalidUsernameInvalidPassword←[0m ←[32m[OK]←[0m
←[36m'--←[0m ←[34mbuildGraphInvalidFilePath←[0m ←[32m[OK]←[0m

Test run finished after 66 ms
[ 3 containers found ]
[ 0 containers skipped ]
[ 3 containers started ]
[ 0 containers aborted ]
[ 3 containers successful ]
[ 0 containers failed ]
[ 15 tests found ]
[ 0 tests skipped ]
[ 15 tests started ]
[ 0 tests aborted ]
[ 15 tests successful ]
[ 0 tests failed ]
```

Figure 3.4: Device Distribution Test Result

3.4 Modification - Household Filter

3.4.1 User Story

“The Community users are having trouble telling which household is theirs in the graph visualisation. They would like the additional feature of being able to see a graph visualisation for their household only”

3.4.2 Problem/modification request

Implement a filter to show specific households in the graph.

3.4.3 Problem/modification analysis

Maintenance Category: Perfective  
Impact Analysis

- **Verify the problem:** User would like to be able to visualise their household only when viewing the graph.
- **Implementation options:** The user should be prompted whether they would like to use a filter on not, and if so they should input a valid household id.
- **Effort:** It may take an estimated 20-50 lines of code with some additional studying of the code to understand where and how to implement this modification(lines of code, additional learning)
- **Resources:** This should take no longer than 2 hours.

3.4.4 Acceptance/rejection

Accepted

3.4.5 Modification implementation

Implementation includes, adding a prompt for the user to select if they would like to use a household filter or not, this should be added in the displayGraph function in the ConsoleApp class. A check for the filter should added in the addNodes and addEdges functions in the GraphVisualiser class, also another function called setFilter should be added along with a private variable to store the users filter.

```

/*
 * Creates a new GraphVisualiser object to visuale the data using the GraphStream Library
 */
private void displayGraph(){

    GraphVisualiser graph = new GraphVisualiser();
    while(true){
        System.out.println(x:"\nFilter:");
        System.out.println(x:"    (1) No Filter");
        System.out.println(x:"    (2) Household Filter");
        System.out.print(s:">> ");
        String option = System.console().readLine();
        if(option.compareTo(anotherString:"1") == 0){
            graph.setFilter(household:"");
            break;
        }
        // if selected the second item
        else if(option.compareTo(anotherString:"2") == 0){
            //set household filter
            ArrayList<String> uniqueIDs = new ArrayList<>();
            //get all unique ids from devices
            for(Device d : deviceGraph.getDevices()){
                if(!uniqueIDs.contains(d.getHouseholdID())){
                    uniqueIDs.add(d.getHouseholdID());
                }
            }
            //read the household id from user
            System.out.println(x:"\nEnter Household ID");
            System.out.print(s:">> ");
            String householdID = System.console().readLine();
            //if the users input exists then set filter otherwise break;
            if(uniqueIDs.contains(householdID)){
                graph.setFilter(householdID);
            }
            else{
                System.out.println(householdID + " does not exist\n");
                continue;
            }
            break;
        }
        // if option is invalid
        else{
            System.out.println(x:"\nPlease select a valid option\n");
        }
    }
    graph.convertGraph(deviceGraph);
    graph.visualiseGraph();
}

```

Figure 3.5: displayGraph

```

/*
 * Adds the devices (nodes) to the graph
 * @param deviceGraph, the DeviceGraph object holding the device information
 */
private void addNodes(DeviceGraph deviceGraph){

    // loops through all the devices to add as nodes
    for(int i = 0; i < deviceGraph.getDevices().length; i++){
        Device d = deviceGraph.getDevices()[i];
        // if filter is not empty add the node that have the household filter as id
        if(!filter.isEmpty()){
            if(d.getHouseholdID().equals(filter)){
                Node node = graph.addNode(d.getDeviceID());
                node.setAttribute("ui.class", d.getDeviceCategory().toString());
                node.setAttribute("ui.label", d.getDeviceName());
            }
        }
        else{
            Node node = graph.addNode(d.getDeviceID());
            node.setAttribute("ui.class", d.getDeviceCategory().toString());
            node.setAttribute("ui.label", d.getDeviceName());
        }
    }
}

```

Figure 3.6: addNodes

```

/*
 * Adds the connections (edges) between the devices to the graph
 * @param deviceGraph, the DeviceGraph object holding the device information
 */
private void addEdges(DeviceGraph deviceGraph){

    // loops through all the devices to find its neighbours
    for(int i = 0; i < deviceGraph.getDevices().length; i++){
        Device d = deviceGraph.getDevices()[i];
        // if filter is not empty add the edges that have the household filter as id
        if(!filter.isEmpty()){
            if(d.getHouseholdID().equals(filter)){
                Device[] neighbours = deviceGraph.getNeighbours(d.getDeviceID());
                // loop through all the device neighbours
                for(int j = 0; j < neighbours.length; j++){
                    if(neighbours[j].getHouseholdID().equals(filter)){
                        graph.addEdge(d.getDeviceID() + ":" + neighbours[j].getDeviceID(),
                                      d.getDeviceID(), neighbours[j].getDeviceID(), true);
                    }
                }
            }
        }
        else{
            Device[] neighbours = deviceGraph.getNeighbours(d.getDeviceID());
            // loop through all the device neighbours
            for(int j = 0; j < neighbours.length; j++){
                graph.addEdge(d.getDeviceID() + ":" + neighbours[j].getDeviceID(),
                              d.getDeviceID(), neighbours[j].getDeviceID(), true);
            }
        }
    }
}

```

Figure 3.7: addEdges

After selecting the graph visualisation option, the user is prompted if they would like to view

the graph or add a household filter. If they select to add a household filter, they should be prompted to type the household id to console, see Figure 3.8. The graph should then display devices with the specified household id, if it exists.

```
Data loaded successfully.

Community Features:
  (1) Data visualisation
>> 1

Filter:
  (1) No Filter
  (2) Household Filter
>> 2

Enter Household ID
>> WKO-1234
```

Figure 3.8: Filter Prompt

3.4.6 Maintenance review/acceptance

Manual testing has been done on this modification, it was done by comparing the filtered and unfiltered graphs, Figure and reference this. We can see that only the devices that belong to 'WKO-1234' appear when the filter is set, all 'WKO-1234's devices are present in the unfiltered graph.

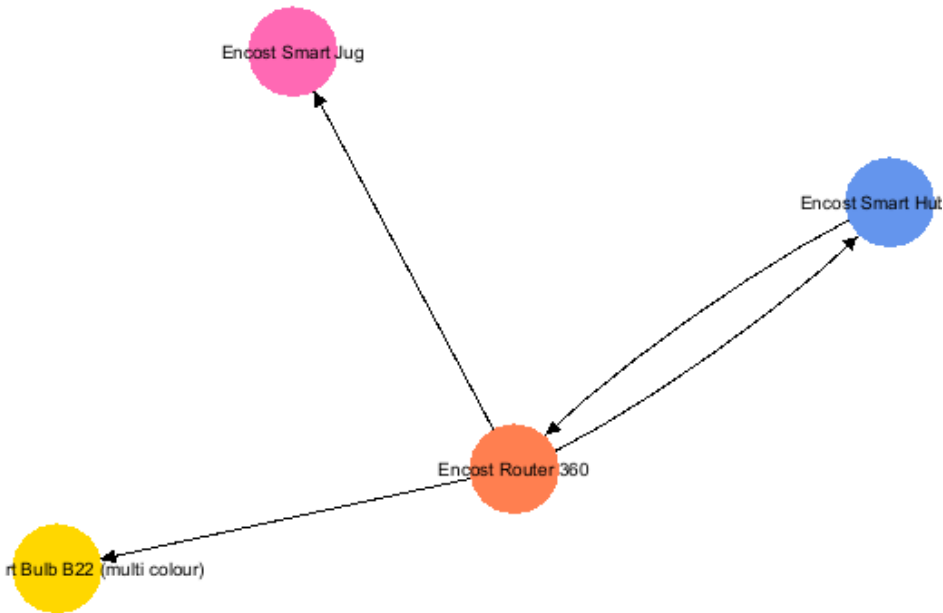


Figure 3.9: Filtered Graph

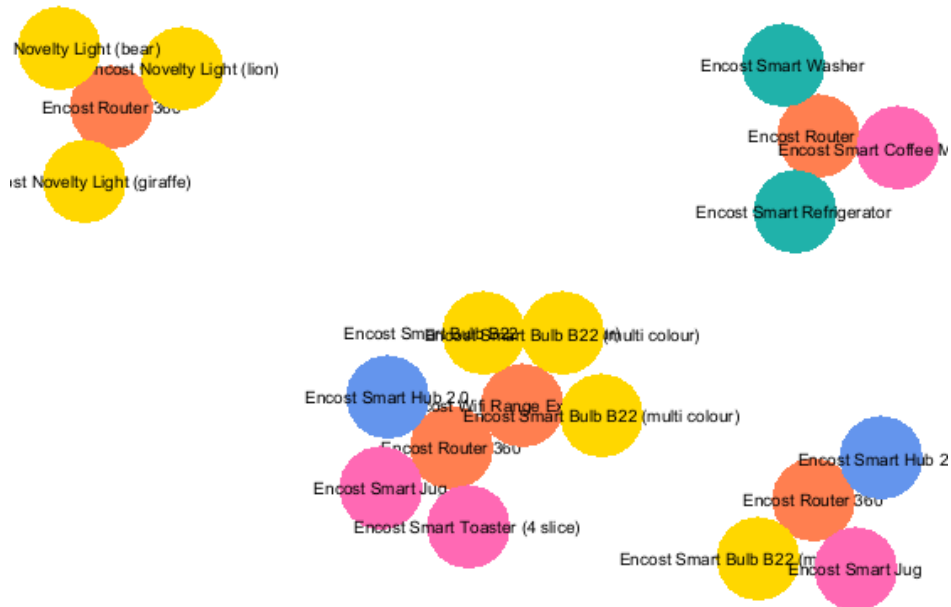


Figure 3.10: Unfiltered Graph

## 4 CI/CD Pipeline

The CI/CD Pipeline will have 5 Stages. Stage 1 will build the application using the the GraphStream dependencies, this line can be found in the README file or in Figure 4.1. Stage 2 will Build the tests using the junit dependencies, this line can also be found in the README file or in Figure 4.1. Stage 3 will Run the tests, if all the tests pass then we can continue with deployment, it is imperative that all tests succeed before deployment. Stage 4 commits all files to GitLab to ensure a secure version control that is always up to date. Stage 5 deploys the application, in this case it will just run the application on the local device.

```

1  @echo off
2  echo:
3  echo (1) Build Application (compiling the application...)
4  javac -cp "gs-core-1.3.jar;gs-ui-1.3.jar" Device.java DeviceGraph.java GraphVisualiser.java
5  ConsoleApp.java FileParser.java UserVerifier.java ESGP.java DataSummary.java
6  IF %ERRORLEVEL% NEQ 0 (
7      echo Build failed, exiting pipeline
8      set /p DUMMY=Hit ENTER to finish...
9      EXIT
10 ) ELSE (
11     echo Application Build succeeded
12 )
13 echo:
14 echo (2) Build Tests (compiling the tests...)
15 javac -cp "gs-core-1.3.jar;gs-ui-1.3.jar;junit-platform-console-standalone-1.8.2.jar" *.java
16 IF %ERRORLEVEL% NEQ 0 (
17     echo Build failed, exiting pipeline
18     set /p DUMMY=Hit ENTER to finish...
19     EXIT
20 ) ELSE (
21     echo Test Build succeeded
22 )
23 echo:
24 echo (3) Run Tests (running the tests...)
25 java -jar junit-platform-console-standalone-1.8.2.jar -cp ".;gs-core-1.3.jar;gs-ui-1.3.jar" -c ESGPTest
26 IF %ERRORLEVEL% NEQ 0 (
27     echo Tests failed, exiting pipeline
28     set /p DUMMY=Hit ENTER to finish...
29     EXIT
30 ) ELSE (
31     echo Tests Passed
32 )
33 echo:
34 echo (4) Release (Committing to GitLab...)
35 Git add .
36 Git commit -m "CI/CD pipeline Automated Commit"
37 IF %ERRORLEVEL% NEQ 0 (
38     echo Release failed, exiting pipeline
39     set /p DUMMY=Hit ENTER to finish...
40     EXIT
41 ) ELSE (
42     echo Release succeeded
43 )
44 echo:
45 echo (5) Deploy (running the Application...)
46 java -cp ".;gs-core-1.3.jar;gs-ui-1.3.jar" ESGP
47 IF %ERRORLEVEL% NEQ 0 (
48     echo Deployment failed, exiting pipeline
49     set /p DUMMY=Hit ENTER to finish...
50     EXIT
51 ) ELSE (
52     echo Deployment succeeded
53 )

```

Figure 4.1: CI/CD Pipeline

## 5 Conclusion

Modifications have been made to ESGP. The DataSummary class has been added, this has been implemented according to the SDS document with some adjustments. An option to show data on specific households has been added, as well as calculating device distribution for Summary Statistics. The CI/CD Pipeline will be helpful during any modifications as progression will be hastened as it removes the need to type out each step for compiling, running, testing and committing to the command line. This documentation will be used to refer to all past and future modifications after the roll out.