FUNCTIONAL SOFTWARE TEST PLAN

for

Encost Smart Graph Project

Version 1.0

Prepared by: Student 1
SoftFlux Engineer

SoftFlux

May 5, 2024

Contents

Revision History

Name	Date	Reason for Changes	Version
Student 1	5/05/24	Finished creating document	V1

1 Introduction/Purpose

1.1 Purpose

The purpose of this document is to provide a test plan for the Encost Smart Graph Project (ESGP) aligning with the requirements outlined in the Software Requirements Specifications document and the proposed design outlined in the Software Design Specification (SDS) document for Student 5. This document will show what tests are necessary for the software to perform the high priority requirements to a satisfactory level. This document will also show potential pseudo-code for calculating the device distribution, with relevant tests for 100% branch coverage, and mutation test for the proposed pseudo-code. This document is used to create tests to determine whether the software has been implemented according to the proposed SDS.

1.2 Document Conventions

This document uses the following conventions:

ESGP: Encost Smart Graph Project

ESHD: Encost Smart Homes Dataset

SRS: Software Requirements Specification

SDS: Software Design Specification

1.3 Intended Audience and Reading Suggestions

This document is intended for any developer, tester, Quality Assurance Engineer or stakeholder in the project. Here are the potential uses for each of the reader types:

- Developer: The developer who wants to create the software or potentially change, modify or add new features into the existing software, should consult this document to ensure that the tests are still valid with no functionality changes, and that the software passes these tests.
- Tester: The tester needs this document to develop the tests that check the designs for this system, as represented in this document.
- Quality Assurance Engineer: The Quality Assurance Engineers for this software need this document to ensure that the software passes these tests to ensure it is fit for purpose.
- Stakeholder: The Stakeholders of this software may wish to review the tests of the design presented in this document and determines if these tests for the application meets all the suitable requirements and if the software developer has implemented the software so that it passes these tests.

1.4 Project Scope

Encost is a Smart Home development company that aims to investigate the usage and connectivity of their devices. Encost Smart Graph Project (ESGP) is being developed to visualize Encost's devices using a graph data structure.

2 Specialized Requirements Specification

Details missing from the SDS with proposed fixes:

- There is no way to access the userType for testing, therefore a getUserType() method should be implemented to make this possible.
- There is no output to the console when selecting visualise graph, therefore a message saying "Graph data visualisation open in new window" should be implemented.
- It is not specified in the SDS whether or not capital letters are valid inputs, therefore it has been assumed that uppercase and lowercase letters are valid.

- It is not specified in the SDS how to access the graph in GraphDataType, therefore it is assumed that a method called getGraph() is added, which is used in Back-end.displayGraph() to display the graph, and is also used to test that devices are correctly going into the graph.

3 Black-box Testing

3.1 Categorizing Users

3.1.1 Description

Users should be able to select whether they are a Community User or an Encost User. As described in the SDS, users will enter an 'a' to select community user and 'b' for an Encost User. These values are not case sensitive therefore 'A' and 'B' could also be used respectively. An internal public method, welcomePrompt(), will be created which takes the input and changes the userType to be 'community', 'encost-unverified' or 'invalid'. The method will ask the user to input an option and will accept one char as input from standard input which represents the user-type entered by the user ('a' or 'b'). Any value input from the user other than 'a', 'A', 'b' or 'B' will result in a string with the value "Invalid Input" to be output to the console, and userType being set to ''.

3.1.2 Functional Requirements Tested

SRS 4.1 REQ-2 The system should store the user-type that the user has selected (community or encost-unverified);

3.1.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Expected inputs, edge cases, boundary cases

3.1.4 Test Cases

Input	Expected userType	Expected Output
‘a’	“community”	“ESGP Feature Options:”
‘b’	“encost-unverified”	“Welcome Encost User please login”
‘A’	“community”	“ESGP Feature Options:”
‘B’	“encost-unverified”	“Welcome Encost User please login”
‘’	“”	“Invalid Input”
‘1’	“”	“Invalid Input”
‘?’	“”	“Invalid Input”
‘c’	“”	“Invalid Input”
‘C’	“”	“Invalid Input”
“abc”	“”	“Invalid Input”

Table 3.1: Test cases for Categorizing Users

3.2 ESGP Account Login

3.2.1 Description

Encost employees should be able to login to the system to access additional features. As described in the SDS, users will enter their username, then their password. An internal public method, loginPrompt(), will be created which takes the input and changes the userType to be ‘encost-verified’ and output a personalised message: ”welcome <Username>”. The method will ask the user to input their username, then their password as input from standard input. Any value input from the user other than a valid username and valid password for that username will result in a string with the value “Invalid username or password please try again” to be output to the console, and userType not being set, staying as ‘encost-unverified’.

3.2.2 Functional Requirements Tested

SRS 4.2 REQ-3 If the username and/or password are invalid, the system should inform the user that they have entered an invalid username and/or password and prompt them to enter their credentials again;

SRS 4.2 REQ-4 If the username and password are valid, the system should update the user-type to be “encost-verified” and should provide the user with the ESGP Feature Options;

3.2.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Expected inputs, edge cases, boundary cases

3.2.4 Test Cases

Example usernames and passwords, where all passwords given below in practice would be hashed variants of the text provided

Username	Password
'encostUserA'	"password789"
'encostUserB'	"password234"

Table 3.2: Valid Username and Password combinations

Input Username	Input Password	Expected Output User-type	Expected Output Console Message
"encostUserA"	"password789"	"encost-verified"	"Welcome encostUserA"
"encostUserB"	"password234"	"encost-verified"	"Welcome encostUserB"
"encostUserA"	"WrongPassword"	"encost-unverified"	"Invalid username or password please try again"
"WrongUsername"	"password789"	"encost-unverified"	"Invalid username or password please try again"
"WrongUsername"	"WrongPassword"	"encost-unverified"	"Invalid username or password please try again'
"encostUserA"	"password234"	"encost-unverified"	"Invalid username or password please try again"
""	"password789"	"encost-unverified"	"Invalid username or password please try again'
"encostUserA"	""	"encost-unverified"	"Invalid username or password please try again"
""	"WrongPassword"	"encost-unverified"	"Invalid username or password please try again'
"WrongUsername"	""	"encost-unverified"	"Invalid username or password please try again"
"'	""	"encost-unverified"	'Invalid username or password please try again'

Table 3.3: Test cases for ESGP Account Login

3.3 ESGP Feature Options

3.3.1 Description

Users should be able to select whether they want to (a) load a custom dataset; (b) visualise a graph representation of the data; or (c) view summary statistics. As described in the SDS, community users will enter an 'a' to visualise a graph representation of the data and Encost users will enter an 'a' to load a custom dataset, 'b' to visualise a graph representation of the data or a 'c' to view summary statistics. An internal public method, esgpOptionsPrompt(), will be created which takes the input and outputs a message for the given choice, running the method for that choice. The method will ask the user to input their choice as input from standard input. Any value input from the user other than 'a', 'b' or 'c', for encost users or 'a' for community users (or the Capital version of those letters), will result in a string with the value "Invalid Input" to be output to the console.

3.3.2 Functional Requirements Tested

SRS 4.3 REQ-2 Once the user has selected a feature, the system should provide them with the prompt/information for that feature.

3.3.3 Test Type

- **Level of test:** Black-box testing, unit test
- **Test technique:** Expected inputs, edge cases, boundary cases

3.3.4 Test Cases

Input	User-type	Expected Console Output
‘a’	“encost-verified”	“Enter full path of custom dataset:”
‘A’	“encost-verified”	“Enter full path of custom dataset:”
‘b’	“encost-verified”	“Graph data visualisation open in new window”
‘B’	“encost-verified”	“Graph data visualisation open in new window”
‘c’	“encost-verified”	“==> Summary Statistics <==”
‘C’	“encost-verified”	“==> Summary Statistics <==”
‘a’	“community”	“Graph data visualisation open in new window”
‘A’	“community”	“Graph data visualisation open in new window”
‘b’	“community”	“Invalid Input”
‘B’	“community”	“Invalid Input”
‘c’	“community”	“Invalid Input”
‘C’	“community”	“Invalid Input”
‘’	All user types	“Invalid Input”
‘9’	All user types	“Invalid Input”
‘e’	All user types	“Invalid Input”
‘E’	All user types	“Invalid Input”
‘!’	All user types	“Invalid Input”

Table 3.4: Test cases for ESGP Feature Options

3.4 Loading the Encost Smart Homes Dataset

3.4.1 Description

The system should be able to read and process the Encost Smart Homes Dataset. As described in the SRS, the ESHD will be located inside the system, in a default location. An internal method, createDataSet(), will be created inside the Dataset class that uses the default location, from filepath to get the ESHD and then loads the file line by line into dataLine, turning the input data into instances of device which are stored in an array.

3.4.2 Functional Requirements Tested

SRS 4.4 REQ-3 The system should be able to read the Encost Smart Homes Dataset line by line and extract the relevant device information.

3.4.3 Test Type

- **Level of test:** Black-box testing, Integration test
- **Test technique:** Expected inputs

3.4.4 Test Cases

Loading the Encost Smart Homes Dataset cannot be tested in isolation due to the data being loaded into an array of devices immediately after being read from the ESHD in createDataSet(). Therefore many of the tests below in 'Categorizing Smart Home Devices' also test 'Loading the Encost Smart Homes Dataset'.

3.5 Categorizing Smart Home Devices

3.5.1 Description

The system should be able to categorize each Encost Smart Device into one of five categories. These categories will be used for the graph visualisation and summary statistics. As described in the SDS, an internal public method, createDataSet(), will be created which takes the filepath, loads the data, and puts each line into a Device object. The device object then calculates the category using the device type.

3.5.2 Functional Requirements Tested

SRS 4.6 REQ-1 The system should determine the device category for each device, based on the information provided on each line of the Encost Smart Homes Dataset

SRS 4.6 REQ-2 The system should create an Object for each device. This object should hold all of the information for that device.

3.5.3 Test Type

- **Level of test:** Black-box testing, integration tests
- **Test technique:** Expected inputs, edge cases, boundary cases

3.5.4 Small ESHD Test Data

Device ID	Date Connected	Device name	Device type	Household ID	Router Connection	Sends	Receives
EWR-1234	01/04/22	Encost Router 360	Router	WKO-1234	-	Yes	Yes
ELB-4567	01/04/22	Encost Smart Bulb B22 (multi colour)	Light bulb	WKO-1234	EWR-1234	No	Yes
EK-9876	07/05/22	Encost Smart Jug	Kettle	WKO-1234	EWR-1234	No	Yes
EHC-2468	01/04/22	Encost Smart Hub 2.0	Hub/ Controller	WKO-1234	EWR-1234	Yes	Yes

Table 3.5: Small ESHD Test Data

3.5.5 Test Cases: using small ESHD Test Data

Method	Expected output
dataset.getDevices().length	4
dataset.getDevices().get(0).getDeviceID()	"EWR-1234"
dataset.getDevices().get(0).getDateConnected().toString()	"Fri Apr 04 00:00:00 2022"
dataset.getDevices().get(0).getName()	"Encost Router 360"
dataset.getDevices().get(0).getDeviceType()	"Router"
dataset.getDevices().get(0).getHouseID()	"WKO-1234"
dataset.getDevices().get(0).getRouterConnection()	"-"
dataset.getDevices().get(0).getSends()	true
dataset.getDevices().get(0).getReceives()	true
dataset.getDevices().get(0).getCategory()	"Encost Wifi Routers"

Table 3.6: Test cases for Loading the Encost Smart Homes Dataset and Categorizing Smart Home Devices

3.5.6 Test Cases

Device type	Expected device category
“Router”	“Encost Wifi Routers”
“Light bulb”	“Encost Smart Lighting”
“Kettle”	“Encost Smart Appliances”
“Router”	“Encost Hubs/Controllers”

Table 3.7: Test cases for Categorizing Smart Home Devices

3.6 Building a Graph Data Type

3.6.1 Description

The system should create a graph data structure to store all of the Encost Smart Device Objects. This graph will be used to visualise the Encost Smart Devices. As described in the SDS, the GraphDataType class has an internal public method setDevices, that takes the devices from Dataset and uses them to addNode and AddEdge to the graph.

3.6.2 Functional Requirements Tested

SRS 4.7 REQ-1 Each Encost Smart Device Object should be stored in the graph data structure. The objects should be the nodes in the graph. The connection between objects should be the edges;

3.6.3 Test Type

- **Level of test:** Black-box testing, integration testing
- **Test technique:** Expected inputs, edge cases, boundary cases

3.6.4 Small ESHD Test Data

3.6.5 Test Cases

Device ID	Date Connected	Device name	Device type	Household ID	Router Connection	Sends	Receives
EWR-1234	01/04/22	Encost Router 360	Router	WKO-1234	-	Yes	Yes
ELB-4567	01/04/22	Encost Smart Bulb B22 (multi colour)	Light bulb	WKO-1234	EWR-1234	No	Yes
EK-9876	07/05/22	Encost Smart Jug	Kettle	WKO-1234	EWR-1234	No	Yes
EHC-2468	01/04/22	Encost Smart Hub 2.0	Hub/ Controller	WKO-1234	EWR-1234	Yes	Yes

Table 3.8: Small ESHD Test Data

Input	Methods	Expected Output
Small ESHD Test Data	GraphDataType.setDevices();	All correct nodes, each node matching 1 device
Small ESHD Test Data	GraphDataType.setDevices();	All correct edges, should be 3 edges connected to the encost router
Single device from Small ESHD Test Data	GraphDataType.addNode();	Graph contains a single node matching the devices from the ESHD

Table 3.9: Test cases for Building a Graph Data Type

3.7 Graph Visualisation

3.7.1 Description

The system should allow the user to view a visualisation of the graph data structure. As described in the SDS, the backend calls the method displayGraph which calls the getGraph method on graphData (of type GraphDataType) and calls the Graphstream display method on the graph.

3.7.2 Functional Requirements Tested

Due to the nature of the tests it is not possible to test this with JUnit, this is because these requirements rely on either knowing the code (REQ-1), or visually inspecting the output graph. We can assume that the graphstream library has been thoroughly tested therefore the display method should work as expected.

4 White-box testing

4.1 Calculating Device Distribution Pseudocode

The pseudo-code below follows the design from the SDS of student 5.

Pseudocode for Dataset.calculateDistribution():

```
Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
        if device category in categoryNum
                increment that category value by 1
                if device type in categoryTypeNum with device category
                        increment that type value by 1
                else
                        add the type to categoryTypeNum with value 1
        else if device category not in categoryNum
                add category to categoryNum with value 1
                add the category and type to categoryTypeNum with value 1
        endif
endfor
Output a header: "Number of devices in each category:"
for each (category, value) pair in the categoryNum
        Output the category with the value: "category: value"
endfor
Output device distribution header: "Device Distribution:"
Output a header: "Number of devices for each device type in category:"
for each (category, (type, value)) pair in the dictionary
        Output the category: "category: <category>"
        for each (type, value)
                Output the type with the pair: "Type: value"
        endfor
endfor
```

4.2 Branch Coverage Testing

Branch coverage testing is testing the branches within the code to ensure that all parts of the code work as intended. The pseudo-code above can be 100% branch tested using a single test case.

This test case has the following devices in the ArrayList devices as per the SDS for Student 5 as follows:

Category	Type
"Encost Smart Appliances"	"Kettle"
"Encost Smart Appliances"	"Kettle"
"Encost Smart Appliances"	"Toaster"

Table 4.1: Test case for branch coverage

4.2.1 Branches Covered by device 1

```
Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
        if device category in categoryNum
                increment that category value by 1
                if device type in categoryTypeNum with device category
                        increment that type value by 1
                else
                        add the type to categoryTypeNum with value 1
        else if device category not in categoryNum
                add category to categoryNum with value 1
                add the category and type to categoryTypeNum with value 1
        endif
endfor
```

4.2.2 Branches Covered by device 2

```
Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
        if device category in categoryNum
                increment that category value by 1
                if device type in categoryTypeNum with device category
```

```
                    increment that type value by 1
                else
                    add the type to categoryTypeNum with value 1
            else if device category not in categoryNum
                add category to categoryNum with value 1
                add the category and type to categoryTypeNum with value 1
            endif
endfor
```

4.2.3 Branches Covered by device 3

```
Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
            if device category in categoryNum
                increment that category value by 1
                if device type in categoryTypeNum with device category
                    increment that type value by 1
                else
                    add the type to categoryTypeNum with value 1
            else if device category not in categoryNum
                add category to categoryNum with value 1
                add the category and type to categoryTypeNum with value 1
            endif
endfor
```

4.2.4 All Branches Covered by Test

```
Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
            if device category in categoryNum
                increment that category value by 1
                if device type in categoryTypeNum with device category
                    increment that type value by 1
                else
                    add the type to categoryTypeNum with value 1
            else if device category not in categoryNum
                add category to categoryNum with value 1
                add the category and type to categoryTypeNum with value 1
```

```
        endif
endfor
Output device distribution header: "Device Distribution:"
Output a header: "Number of devices in each category:"
for each (category, value) pair in the categoryNum
        Output the category with the value: "category: value"
endfor
Output a header: "Number of devices for each device type in category:"
for each (category, (type, value)) pair in the dictionary
        Output the category: "category: <category>"
        for each (type, value)
            Output the type with the pair: "Type: value"
        endfor
endfor
```

4.2.5 Expected output from the test

Device Distribution: Number of devices in each category: Encost Smart Appliances: 3 Number of devices for each type in category: Category: Encost Smart Appliances Kettle: 2 Toaster: 1

Table 4.2: Test case output for branch coverage

5 Mutation Testing

Mutation testing is adding 1 error to the code and checking if the test picks it up. For the pseudo-code above, there are 4 mutation tests given below.

5.1 Mutant #1

Change the pseudo-code by changing increment by 1 to set to 1 as shown below in green.

```
   Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
        if device category in categoryNum
            Set that category value to 1
            if device type in categoryTypeNum with device category
                increment that type value by 1
            else
                add the type to categoryTypeNum with value 1
        else if device category not in categoryNum
            add category to categoryNum with value 1
            add the category and type to categoryTypeNum with value 1
        endif
endfor
Output device distribution header: "Device Distribution:"
Output a header: "Number of devices in each category:"
for each (category, value) pair in the categoryNum
        Output the category with the value: "category: value"
endfor
Output a header: "Number of devices for each device type in category:"
for each (category, (type, value)) pair in the dictionary
        Output the category: "category: <category>"
        for each (type, value)
            Output the type with the pair: "Type: value"
        endfor
endfor
```

Device Distribution: Number of devices in each category: Encost Smart Appliances: 1 Number of devices for each type in category: Category: Encost Smart Appliances Kettle: 2 Toaster: 1

Table 5.1: Output for mutation test 1

5.2 Mutant #2

Change the pseudo-code by removing an else statement as shown below, with the else to remove in red.

```
  Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
        if device category in categoryNum
                increment that category value by 1
                if device type in categoryTypeNum with device category
                        increment that type value by 1
                else
                        add the type to categoryTypeNum with value 1
        else if device category not in categoryNum
                add category to categoryNum with value 1
                add the category and type to categoryTypeNum with value 1
        endif
endfor
Output device distribution header: "Device Distribution:"
Output a header: "Number of devices in each category:"
for each (category, value) pair in the categoryNum
        Output the category with the value: "category: value"
endfor
Output a header: "Number of devices for each device type in category:"
for each (category, (type, value)) pair in the dictionary
        Output the category: "category: <category>"
        for each (type, value)
                Output the type with the pair: "Type: value"
        endfor
endfor
```

Device Distribution: Number of devices in each category: Encost Smart Appliances: 3 Number of devices for each type in category: Category: Encost Smart Appliances Kettle: 1 Toaster: 1

Table 5.2: Output for mutation test 2

5.3 Mutant #3

Change the pseudo-code by swapping categoryTypeNum with categoryNum in an if statement as shown below in green.

```
  Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
        if device category in categoryNum
                increment that category value by 1
                if device type in categoryNum with device category
                        increment that type value by 1
                else
                        add the type to categoryTypeNum with value 1
        else if device category not in categoryNum
                add category to categoryNum with value 1
                add the category and type to categoryTypeNum with value 1
        endif
endfor
Output device distribution header: "Device Distribution:"
Output a header: "Number of devices in each category:"
for each (category, value) pair in the categoryNum
        Output the category with the value: "category: value"
endfor
Output a header: "Number of devices for each device type in category:"
for each (category, (type, value)) pair in the dictionary
        Output the category: "category: <category>"
        for each (type, value)
                Output the type with the pair: "Type: value"
```

```
        endfor
endfor
```

Device Distribution: Number of devices in each category: Encost Smart Appliances: 5 Number of devices for each type in category: Category: Encost Smart Appliances Kettle: 2 Toaster: 1

Table 5.3: Output for mutation test 3

5.4 Mutant #4

Change the pseudo-code by setting the value to 0 instead of 1 as shown below in green.

```
  Create empty dictionary categoryNum of (category, value) pairs
Create empty dictionary categoryTypeNum of (category, dictionary(type, value)) pairs
Get devices
for Device in devices
        if device category in categoryNum
                increment that category value by 1
                if device type in categoryTypeNum with device category
                        increment that type value by 1
                else
                        add the type to categoryTypeNum with value 1
        else if device category not in categoryNum
                add category to categoryNum with value 0
                add the category and type to categoryTypeNum with value 1
        endif
endfor
Output device distribution header: "Device Distribution:"
Output a header: "Number of devices in each category:"
for each (category, value) pair in the categoryNum
        Output the category with the value: "category: value"
endfor
Output a header: "Number of devices for each device type in category:"
for each (category, (type, value)) pair in the dictionary
```

Output the category: "category: <category>"
for each (type, value)
Output the type with the pair: "Type: value"
endfor
endfor

Device Distribution: Number of devices in each category: Encost Smart Appliances: 2 Number of devices for each type in category: Category: Encost Smart Appliances Kettle: 2 Toaster: 1

Table 5.4: Output for mutation test 4

5.5 Mutation Score

Using the input and output shown above we can see in the table below which mutations pass and which fail.

Mutation Test 1	Mutation Test 2	Mutation Test 3	Mutation Test 4
Pass	Pass	Pass	Pass

Table 5.5: Mutation Test, pass or fail

5.5.1 Calculating mutation score

Using the input and output combinations above we can calculate the mutation score:

mutation score = (caught mutants) / (mutants)

Therefore:
mutation score = 4 / 4 = 100%