SOFTWARE DESIGN

for

Encost Smart Graph Project

Version 1.0

Prepared by: Student 5
SoftFlux Engineer

SoftFlux

April 5, 2024

Contents

Revision History

Name	Date	Reason for Changes	Version

1 Introduction/Purpose

1.1 Purpose

The purpose of this document is to specify the Software Design Specifications (SDS) based on the Encost Smart Graph Project (ESGP) software requirements. This document will define and describe the software architecture of ESGP. Component, process, use case and class diagrams will be used to demonstrate how it's components and the interfaces between them will interact. This document is to used to compare and test with during development to determine whether this software has been successfully implemented base the contents of this document.

1.2 Document Conventions

This document uses the following conventions:

- ESGP: Encost Smart Graph Project
- SDS: Software Design Specifications
- SRS: Software Requirements Specification

1.3 Intended Audience and Reading Suggestions

This document is intended for any, developer, tester, and project manager. Here are the potential uses for each of the reader types:

- Developer: The developer who wants to read, change, modify or add new design specifications into the existing program, should firstly consult this document and update the design specifications in an appropriate manner so as to not destroy the actual meaning of them and pass the information correctly to the next phases of the development process.
- Tester: The tester needs this document in order to develop tests which check that the design for this system, as given in this document, are actually met by any program which some developer claims meets the design specifications.
- Project Manager: The Project Manager can use this document to get an idea of how this software is meant to be developed and used so that they can better manage the project as a whole.

1.4 Project Scope

Encost is a emerging Smart Home development company, who are dedicated to manufacturing a variety of Smart Home and IoT solutions, including Wifi Routers, Smart Hubs and Controllers, Smart Light Bulbs, Smart Appliances, and Smart Whiteware. Encost is focused on understanding how their smart devices are being used and connected within New Zealand households. They have worked alongside energy companies and users, to gather data on smart device usage in 100 homes across New Zealand from April 2020 to April 2022, creating the Encost Smart Homes Dataset.

The Encost Smart Graph Project (ESGP) is a software system that enables the visualisation of Encost's devices using a graph data structure. With access to the Encost Smart Homes Dataset, ESGP enables users to explore all of the devices within the dataset, along with their connection to one another. It also provides verified users with summary statistics on device distribution, location, and connectivity.

2 Specialized Requirements Specification

The following is to clarify any ambiguities, missing detail, missing features and so on in the SRS.

2.1 Functional Requirements

2.1.1 Welcome Prompt

When the application begins the console will display a welcome message followed but a prompt asking the user to input whether they are a Community user or an Encost

user. The application should store this, saying if they are an "encost-unverified" user or a "community" user. If the user is an "encost-unverified" user the user will be prompted with an account login. Else the user is prompted with ESGP Feature Options **excluding** features for an *Encost user*.

2.1.2 ESGP Account Login

An "encost-unverified" user is first prompted to input their username, once entered they're then prompted to input their password. Once both are entered check if the username and password match a username and encrypted password stored in the application. If the login is incorrect loop back to the beginning of this process. Else if the login is correct then update user type to "encost-verified" and go to ESGP feature options for an *Encost user*.

2.1.3 ESGP Feature Options

If user is a **community user** they should only be prompted with one option: visualising graph representation of the data. Thus, community users should be restricted from accessing any other option. If the user is an **Encost User** they should be prompted with two additional options. Therefore, an Encost user should be prompted wit three options; (a) loading a custom dataset; (b) visualising a graph representation of the data; or (c) viewing summary statistics; Once the user has input one of the feature options the system should go to that feature.

2.1.4 Reading From A Dataset

The system should have a pointer to the location of the current dataset. The default dataset is The Encost Homes Dataset which should already be in the system (e.g. within the projects folder). The system should then be able to read the dataset specified by the loctation line by line extracting relevant device information. The layout of the dataset should be displayed something like this:

deviceID,dateConnected,deviceName,deviceType,householdID, routerConnection,rends,receives
EWR-1234,01/04/22,Enost Router 360,Router,WKO-1234,,Yes,Yes
ELB-4567,01/04/22,Encost Smart Bulb B22,Light bulb,WKO-1234,EWR-1234,No,Yes

2.1.5 Loading Custom Datasets

The system should prompt the user to enter the full file path of their custom dataset. The system should attempt to read this dataset based upon the format in Reading From A Dataset. If the read is successful then it should save the dataset as the current file location. Else if the read fails (dataset is incorrectly formatted) then the system should prompt the user that the dataset is not compatible and allow them to: (a) use the Encost Smart Homes Dataset or (b) enter a different file path.

2.1.6 Processing Datasets

The system should categorize each device based on each line of the dataset. Device categories are shown in a table in SRS (4.7.1). The system should create an Object for each device. The Object should hold all information about the graph.

2.1.7 Building A Graph Data Type

Each Encost Smart Device object from the dataset should be stored within the graph data structure that can be used by the GraphStream library. The graph data type should have the objects as nodes in the graph and the connection between objects should be the edges. All unique data points should be included in the graph. Also, all households should be represented in the graph.

2.1.8 Graph Visualisation

When the user selects "visualising a graph representation of the data" from the ESGP Feature Options in a UI window a visualisation of the graph data structure should be shown. This graph visualisation must be implemented by using the GraphSteam Library, show all nodes and show all connections between nodes (e.g. edges). The graph must distinguish between different device categories visually (e.g. different colour). It must also illustrate each device's ability to send and receive commands (e.g. shapes).

2.1.9 Summary Statistics

When the "viewing summary statistic" ESGP Feature Option is selected the system should display three sets of statistics. These statistics are: Device distribution, Device Location and Device Connectivity. For the calculations for these sets of statistics refer to the SRS (4.9-4.11). These statistics should be displayed in a clear and concise manner in the console.

2.2 Non-Functional Requirements

2.2.1 Performance Requirements

In each case of calculating device summary statistics and the processing and loading of a dataset each should **not** take more then ten seconds. Whereas, graph visualisation should take no more then five seconds. In each case the user should receive feedback after one second of loading.

2.2.2 Security Requirements

Encost employees should be able to login to the system to access additional features. Ten usernames and password are stored in the system for this purpose, passwords are

encrypted before they are stored. Encost should make sure these usernames and passwords are not leaked. A password entered during login should be encrypted and compared against the encrypted passwords stored in the system. All datasets that are used by the system must be anonymised before used by the system.

2.2.3 Software Quality Attributes

For reliability data loss must be prevented as much as possible and backups of source code should be made. For interoperability UTF8 encoding should be used and the product must be available in windows 10, java 1.8.0 or higher and GraphStream 1.3. For Usability textual layout must be clear and readable, language must be concise and easy to understand, user input options must be easy to understand and Error messages and feedback should be informative.

3 Software Architecture

This chapter discusses the software architecture of the project which determines the fundamental structure of the system as well as the actions that must be taken to produce this system's structure.

3.1 Component Architecture Diagram

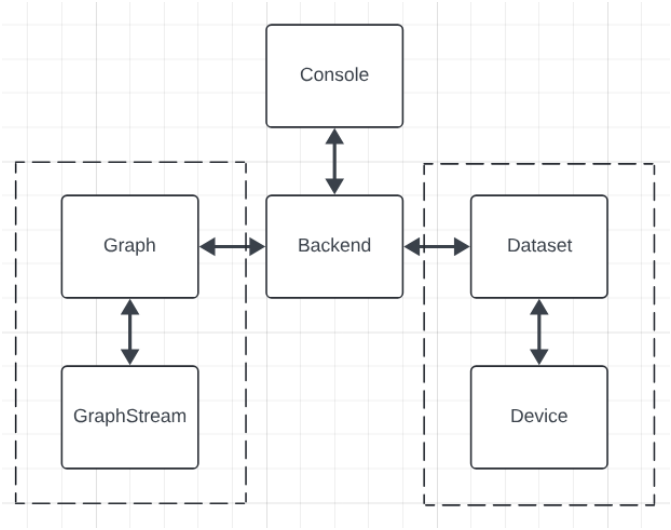

Figure 3.1: Components diagram

Figure 3.1 shows that all communication in the system relies on the back-end. This is because the console sends and receives information from the back-end of the system. The back-end then passes this information to the other components. This starts with the welcome message where the back-end sends a message to the console and receives the user type from the user input. Then the back-end either displays a login or ESGP feature options. Login will be done in the back-end which will be responsible for encrypting the password and checking login information with the stored logins. Then the user chooses the feature they want in the console after options are displayed by the back-end. If the feature is loading a custom data set then then the back-end processes the data and puts that information into the dataset and device components. If the feature is visualising a graph representation of the data then the back-end uses the dataset to transfer it to a graph data type which then uses GraphStream to display it in a UI window. If the feature is summary statistics then the back-end uses the dataset component to calculate and display stats to the console.

In conclusion this shows why the back-end must be efficient, know where to sent information to and receive information from and what it needs to calculate. As, the back-end is the backbone of the system.

3.2 Process Diagram

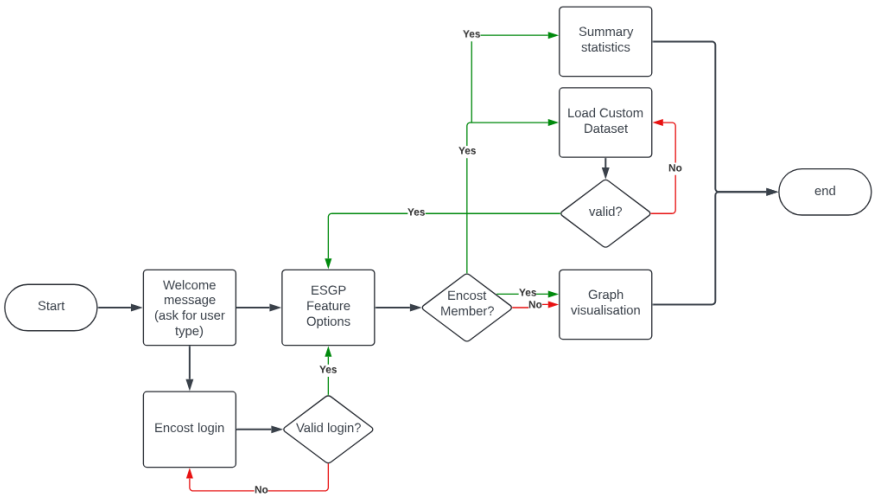

Figure 3.2: Process diagram

Figure 3.2 shows the process of checks the system uses to allow the user to progress along the system. Firstly when the user starts the application the user will be prompted with a welcome message which will ask for their user type. If the user type is a Encost user they will be prompted with a login. If this login is not valid they will be prompted to login again. Else if the user type is a community user or the user is a valid encost user they will be prompted with some feature options. Encost users can input all three

options summary statistics, load custom data set and graph visualisation. Whereas, community users will only be prompted with graph visualisation as their input and can't access the other options. If a user selects summary statistics they will be shown the summary statistics of the current dataset. If the user selects load custom data they can use their own dataset. If valid they will be taken back to EGSP feature options allowing them do view details of there dataset. If not valid ask them to try again or use the default dataset. If the user selects graph visualisation then a UI window will be displayed of the current dataset.

This diagram shows the importance of each function in the system to allow the user to progress to the end of the process. Each of these functions is control by the back-end once again showing the importance of the back-end of the system.

3.3 Use Case Diagram

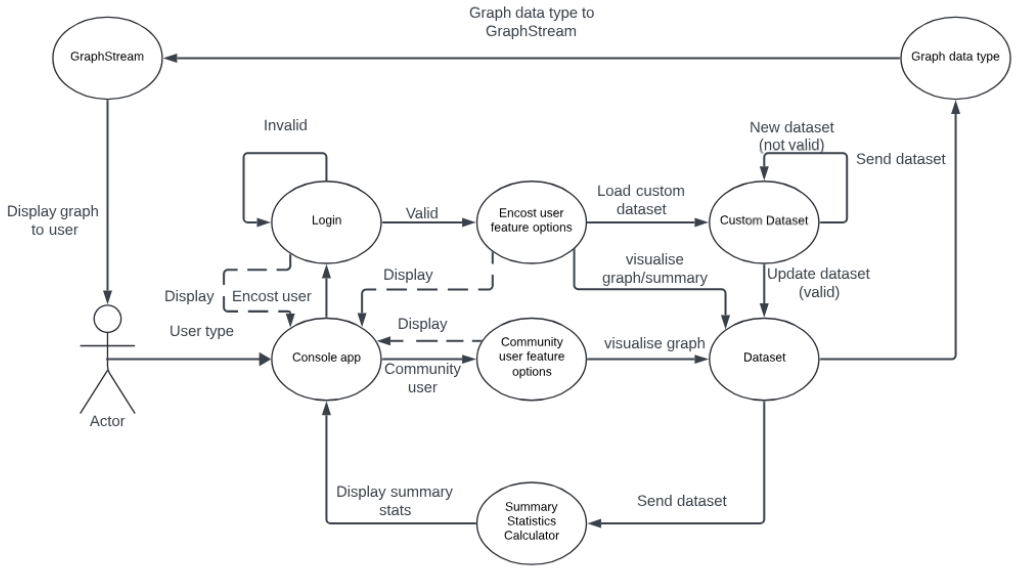

Figure 3.3: Use Case diagram

Figure 3.3 shows the scope of the system as well as it's interaction with the user. First the user is given a welcome message with allows the user to enter a user type into the console. If the user type is a community user it goes to a community feature option which is only the visualise graph option. If the user type is Encost then a login is displayed to the console which can't be moved pass until a valid user login has been entered. Once there is a valid login all the feature options are displayed. Both visualise graph and summary statistics go straight to the data set to get what they need. Meanwhile, the load custom dataset loads a custom data set which is then set as the current data set if valid. Then if graph needs to be visualised then data set is turned into a graph data

type which gets sent to GraphStream to send the user a visual graph in a UI window. If summary statistics are needed then the dataset is sent to the calculator in the back-end and then the statistics are displayed in the console.

It's important to show how the system interacts with the user as it makes it easier to form restrains and limits on where the user can go and what they do. As, the user should not be able to see what happens in the back-end, they should only see what the system updates to the console. This is why this diagram is good cause it shows you how information should be updated to the user.

4 Component Design

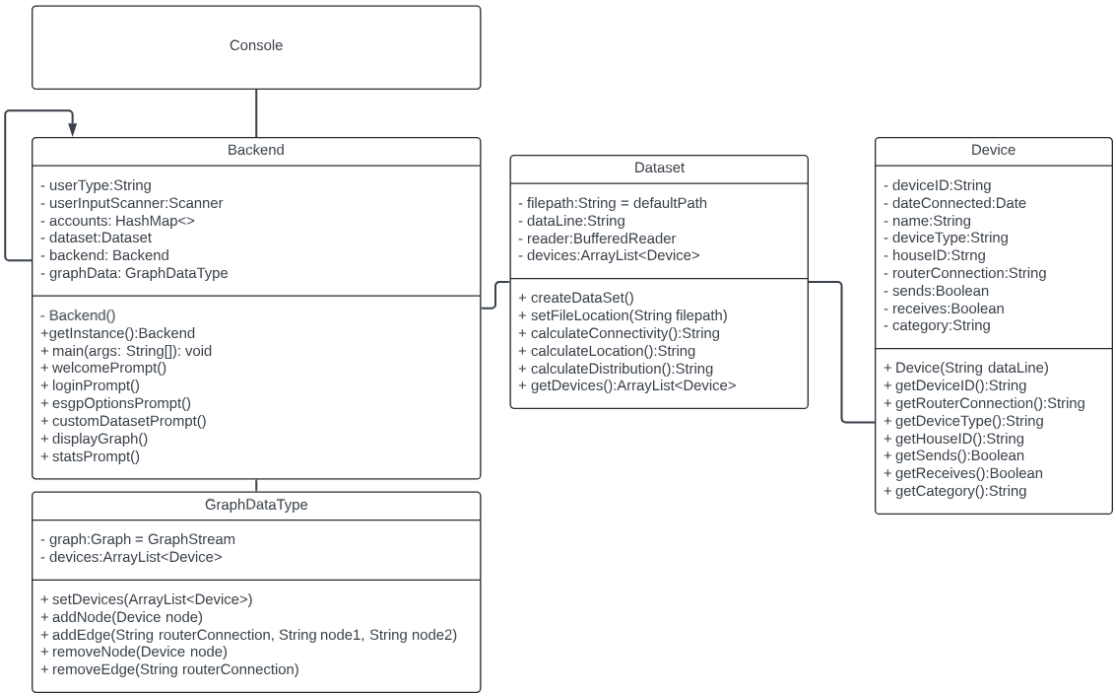

Figure 4.1: UML Class Diagram

4.1 Console

The console is used to communicate back and forth with the user as you can see in the wire-frames in section 3.4. This communication informs the back-end what actions to take next. The console is also the entry point of the system.

Implementation

The console calls the main method of the back-end of the program. Then the back-end uses standard i/o output to display prompts to the user. Then the console waits for a response from the user using a scanner in the back-end. The user responses determine the next actions of the back-end. This makes sure all communication with the user is done is one place allowing it to be clear and concise.

4.2 Back-end

The Back-end is responsible for communication with the user. Then spreading this information and updating the rest of the system. The Back-end also receives responses from the system and displays them to the user. This means the Back-end is the most important component as it is responsible for the communication of the system. Meaning the rest of the system can't function properly with out it.

Attributes

- **userType**

  userType is a string which is responsible for storing the type of user that the user is "community", "encost-verified" or "encost-unverified".

- **userInputScanner**

  userInputScanner is a scanner that allows the system to read an input from the console.

- **accounts**

  accounts is a hashmap that stores the ten usernames and encrypted passwords.

- **dataset** dataset holds the dataset being used.

- **graphData** graphData holds the GraphDataType being used.

- **backend**

  backend is used to create a singleton of the Backend class.

Functions

- **Backend()** Backend() is a private constructor used to initialise the variables and create a singleton also creates the default dataset.

- **getInstance()**

  getInstance() is a public method used to get the instance of the singleton so only one is created.

- **main(String[] args)**

  The main function is a public static method used by the console to call the back-end and is used to call the other functions in the correct order. Should call constructor at start.

- **welcomePrompt()**

  The welcome prompt function is a public method used to welcome the user then asks for a user type. The input is then stored in the userType attribute.

- **loginPrompt()**

  loginPrompt() is a public method used to display a login prompt that asks the user for a username and password. Then encrypts the entered password and compares the username and encrypted password with the ones in the hashmap. If valid continue if not ask for username and password again.

- **esgpOptionsPrompt()**

  esgpOptionsPrompt() is a public method that displays the ESGP feature options and then runs the option chosen in the input.

- **customDatasetPrompt()**

  customDatasetPrompt() is a public method that displays a prompt to the user to enter a file path. If valid file path and file then save file path, create custom dataset and set as back-end dataset. If not valid ask user if they want to use default dataset or a different custom dataset if so try again.

- **displayGraph()**

  displayGraph() is a public function that displays creates a GraphDataType from the dataset and displays it in a UI window using GraphStream.

- **statsPrompt()**

  statsPrompt is a public method that runs the stat calculations and displays it all to the console in a clear and concise manner.

Implementation

The back-end must only have one version of itself so should be implemented as a singleton called in the main function. The console uses the the back-end to get prompts and gives the back-end inputs from the user making the back-end respond accordingly. It uses functions from the Dataset class to get it's own dataset and calculate the summary statistics. It also uses functions from the GraphDataType class to create a graph and display it in a UI window. This ensures all information is passed and checked in this component, this will provide consistency, prevent loss as well as preventing invalid or unwanted changes to data.

4.3 Dataset

The Dataset component is used to create a collection of devices based upon a file path. The file must have a valid format aligned with the variables. It also is able to calculates it's own dataset summary statistics.

Attributes

- **filepath** The filepath is a string attribute which is set to the default filepath the file path location of the Encost Smart Homes Dataset. This is used to load a dataset.
- **dataLine**

  dataLine is a string attribute which holds a line from a file that contains device data.
- **reader**

  reader is a BufferedReader used to read from a file containing a dataset.
- **devices**

  devices is an ArrayList of the Device component this holds the current dataset.

Functions

- **createDataSet()**

  createDataSet() is a public method that creates an ArrayList of the Device component from a file that holds lines of data from a dataset. Each line is it's own Device. Then stores this in the devices attribute.
- **setFileLocation(String filepath)**

  setFileLocation(String filepath) is a public method that allows another components to set the file path by passing in a string.
- **calculateConnectivity()**

  calculateConnectivity() is a public method that calculates the connectivity statistics and return them clear and concisely in a String.
- **calculateLocation()**

  calculateLocation() is a public method that calculates the location statistics and return them clear and concisely in a String.
- **calculateDistribution()**

  calculateDistribution() is a public method that calculates the distribution statistics and return them clear and concisely in a String.
- **getDevices()** getDevices() is a public method that returns the attribute devices so other components get access to it.

Implementation

The Dataset component is used to create a dataset from a file. The dataset is made up of Device object in an ArrayList. This is done by splitting each line of the file into it's own Device object. The file is found through the file path which is either the default file path (file path location of the Encost Smart Homes Dataset) or it is a custom file path set by the back-end from a user input. Also, allows components to view it's summary stats. All functions must be accessible to the back-end. This is important because the back-end will use these Functions to display information to the user.

4.4 GraphDataType

GraphDataType is used to display a UI Window that is a user interface which displays to the user when they select the Graph Visualisation option from ESGP Feature Options.

Attributes

- **graph**

  graph holds a external library object called GraphStream as a graph object which allows a graph to be displayed in a UI window

- **devices**

  devices holds the ArrayList of Device objects from the dataset.

Functions

- **setDevices(ArrayList¡Device¿)**

  setDevices(ArrayList¡Device¿) is a public method that allows you to give GraphDataType the dataset.

- **addNode(Device node)**

  addNode(Device node) is a public method which adds a node to the graph. The node represents a device.

- **addEdge(String routerConnection, String node1, String node2)**

  addEdge(String routerConnection, String node1, String node2) is a public method which adds a edge to the graph. The edge represents a router connection.

- **removeNode(Device node)**

  removeNode(Device node) is a public method which removes a node from the graph. Therefore, removing a device from the graph.

- **removeEdge(String routerConnection)**

  removeEdge(String routerConnection) is a public method which removes a edge from the graph. Therefore, removing a router connection from the graph.

Implementation

GraphDataType will use the GraphStream library to display the graph in a UI Window using the dataset. It will have nodes for devices and edges for connections. Each dataset there should only have one graph. This is important because there should only be each UI window should only use one dataset.

4.5 Device

The Device component allows the system to store device data from lines in a dataset file.

Attributes

- **deviceID**

  deviceID is a private attribute that holds the id of the device as a string.

- **dateConnected**

  dateConnected is a private attribute that holds the data a device was connected as a date.

- **name**

  name is a private attribute that holds the name of a device as a string.

- **deviceType**

  deviceType is a private attribute that holds the type of device it is as a string.

- **houseID**

  houseID is a private attribute that holds the name of the house it is connected to.

- **routerConnection**

  routerConnection is a private attribute that holds the router a device is connected to as a string.

- **sends**

  sends is a private attribute that holds whether or not a device can send information in a boolean.

- **receives**

  recieves is a private attribute that holds whether or not a device can receive information in a boolean

- **category**

  category is a private attribute that holds what category a device is part of as a string.

Functions

- **Device(String dataLine)**

  Device(String dataLine) is a public constructor that intailises all the attributes include finding what category the device is.

- **getDeviceID()**

  getDeviceID() is a public method that returns the attribute deviceID.

- **getRouterConnection()**

  getRouterConnection() is a public method that returns the attribute routerConnection.

- **getDeviceType()**

  getDeviceType() is a public method that returns the attribute deviceType.

- **getHouseID()**

  getHouseID() is a public method that returns the attribute houseID.

- **getSends()**

  getSends() is a public method that returns the attribute sends.

- **getReceives()**

  getReceives() is a public method that returns the attribute receives.

- **getCategory()**

  getCategory() is a public method that returns the attribute category.

Implementation
A new device object will be created for each line of data in the dataset file, this ensures encapsulation, which allows access control and prevents unwanted changes.

4.6 Off-the-shelf Technologies

4.6.1 GraphStream

GraphStream an external Java library for building and visualising dynamic graphs. This library will serve the purpose of building graph of Encost smart home devices. GraphStream is designed to handle any change that will occur over time in the graph ensuring the addition or removal of any devices to and from the dataset are handled correctly and effectively. The GraphStream libraries visualisation options are ideal for what information we want to be displayed on the graph. Allowing us to make each category of device visually distinguishable on the graph, as well as being able the connection and communication between devices through edges and visually displaying whether a device can send/receive.

4.6.2 Git

Git enables version control, this allows developers to track previous versions of source code and reverse any changes when necessary or recover source code if an error has occurred preventing data loss. Git also enables collaboration for developers, it means developers can work on the same code files simultaneously. This is also helpful for maintenance so a version of the source code can be saved daily.

4.7 UI & Wire-frame Diagrams

This section will mainly cover the wire-frames as each process is already explained in section 2.

4.7.1 Welcome Prompt

Figure 4.2 shows the wire-frame for the welcome prompt.

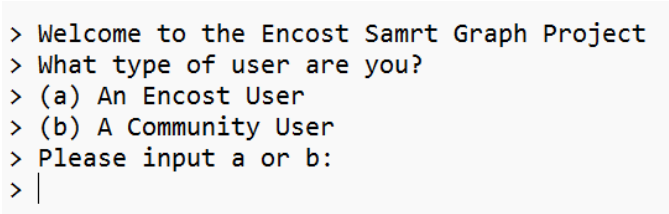

Figure 4.2: Welcome Prompt Wire-frame

1. The welcome prompt first welcomes the user
2. Asks the user to enter a user type either community or Encost user
3. User types what user they are
4. Based on user type the the system moves to next function

This wire-frame helps the programmer understand how the system should interact with the user during the welcome function.

4.7.2 Login Prompt

Figure 4.3 shows the wire-frame for the login prompt.

```
> Welcome Encost User please login
> Input your username:
> EncostUser1
> Input your password:
> WrongPassword
> Invalid username or password please try again
> Input your username:
> EncostUser1
> Input your password:
> Password1
> Welcome EncostUser1
>
```

Figure 4.3: Login Prompt Wire-frame

1. Welcome Encost User
2. Ask for username
3. Ask for password
4. If username or password are not valid then repeat steps 2 and 3
5. If login is valid then welcome user
6. Move to ESGP Feature Options

This wire-frame helps the programmer understand how the system should interact with the user during the login function.

4.7.3 ESPG Feature Options

Figure 4.4 shows the wire-frame for the ESPG Feature Options.

```
> ESGP Feature Options:
> (a) loading a custom dataset
> (b) visualising a graph representation of the data
> (c) viewing summary statistics
> Input the feature you would like to use a, b or c:
>
```

Figure 4.4: ESPG Feature Options Wire-frame

1. If user is an Encost user display all options as shown in figure 3.6
2. If user is a community user display only visualise graph option
3. User selects an option
4. Based on option entered move to that function

This wire-frame helps the programmer understand how the system should interact with the user during the ESPG Feature Options function.

4.7.4 Load Custom Dataset Prompt

Figure 4.5 shows the wire-frame for the load custom dataset prompt.

```
> Enter full path of custom dataset:
> /home/example/path/datasetError.text
> invalid dataset
> Would you like to try:
> (a) another custom dataset
> (b) the default dataset
> Please enter a or b:
> a
> /home/example/path/dataset.text
> Dataset has been saved
>
```

Figure 4.5: Load Custom Dataset Wire-frame

1. Ask for the users custom data full path
2. User enters path
3. If data not valid prompt for new dataset or default dataset
4. If valid save custom dataset
5. Move to feature options

This wire-frame helps the programmer understand how the system should interact with the user during the load custom dataset function.

4.7.5 Visualise Graph

Figure 4.6 shows the wire-frame for the visual graph.

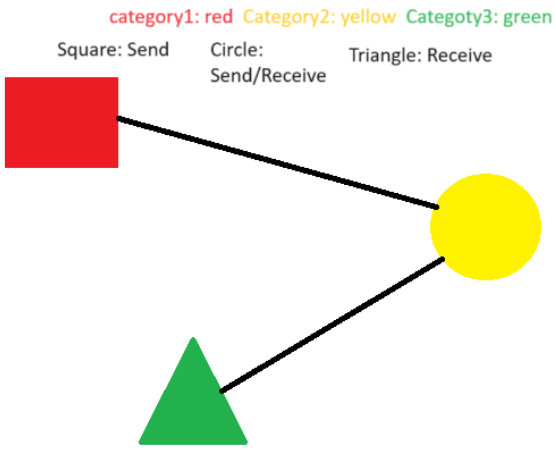

Figure 4.6: Graph Wire-frame

1. graph is displayed in a UI window
2. Each household should be represented by their own group of nodes
3. Each node of a certain category should have it's own visual difference (e.g. colour)
4. Each node should display if it can send/receive (e.g. shapes)
5. Each unique datapoint should be displayed
6. The graph must also display the connections between devices (e.g. edges)

This wire-frame helps the programmer understand how the graph should be displayed to the user via the UI window.

4.7.6 Summary Statistics

Figure 4.7 shows the wire-frame for the summary statistics prompt.

```
> ==> Summary Statistics <==
>
> Device Distribution:
> Number of devices in each category
> category1: 1
> ...
> Number of devices for each device type:
> type1: 2
> ...
>
> Device Location:
> Number of households in each reigon of NZ:
> WKO: 1
> ...
> ... (rest of location stats)
>
> Device  Connectivity:
> Average number of devices that an Encost Wifi Router is connected to: 1
> ... (rest of connectivity stats)
```

Figure 4.7: Summary Statistics Wire-frame

1. Display Summary Statistics
2. First display device distribution statistics
3. Second display device location statistics
4. Lastly display device connectivity statistics
5. Make sure it's displayed clearly and concisely

This wire-frame helps the programmer understand how the system should interact with the user during the summary statistics functions.

5 Conclusion

In conclusion this SDS for Encost's smart graph project has covered the functional and non-functional requirements as well as the software architecture, component design and the user interfaces. It uses diagrams to discuss how the components should interact and what they should look like. These diagrams include a Component Diagram, a Process Diagram, a Use Case Diagram and a UML Class Diagram. This SDS also gives a rough idea of what the UI interfaces should look like using wire-frames. Also, it expands on GraphStream and other external technologies used for the project. The use of this SDS should give confidence that the requirements have been meet and should provide value to Encost and it's users.

6 References

- Software Requirements Specification: Link to SRS
- GraphStream: Link to GraphStream